

Access und Bilder

HARALD LANGER

Überblick

Behandelt werden soll

- Erfassen, verwalten und darstellen von Bildern in Access-Datenbanken
- Arbeiten mit den Imaging-Controls
 - Scannen, bearbeiten, verwalten und darstellen von Bildern

Gliederung

- Darstellung in Formularen und Berichten bei unterschiedlichen Speichervarianten
- Pfad zum externen Bild (ca. 30 Min)
- Interne Binärspeicherung (ca. 30 Min)
- Scannen, bearbeiten und verwalten mit den Imaging-Controls (ca. 20 Min)

Fachausdrücke

- Bildsteuerelement: Steuerelement in Access zur Darstellung von Bildern
- ScanControl: Steuerung für Kodak-Bildscan (W2k) / Steuerelement für Wang-Bildscan (NT4) / Kodak Image Scan Control (Win98)
- EditControl: Steuerung für Kodak-Bildbearbeitung (W2k und Win98) / Steuerelement für Wang-Bildbearbeitung (NT4)
- AdminControl: Steuerung für Kodak-Bildverwaltung (W2k und Win98) / Steuerelement für Wang-Bildverwaltung (NT4)
- ThumbnailControl: Kodak-Bildminiaturansichtssteuerung (W2k und Win98) / Steuerelement für Wang-Bildminiaturansicht (NT4)

Externe Bilder

Trennung von Bild und Datenbank

Der vermutlich gravierendste Nachteil beim Zugriff auf externe Bilddateien ist, daß die gespeicherten Pfade aktuell gehalten werden müssen. Was nutzt die schönste Bildverwaltung, wenn die Dateien an einen anderen Ort verschoben wurden?

Um dem entgegenzuwirken, hat es sich als wirksam erwiesen, den Namen des Bildes und den Pfad zum Bild getrennt in der Datenbank zu hinterlegen. Kann zusätzlich auf Gruppierungsmerkmale, wie „Kategorie“ o.ä., zurückgegriffen werden, lassen sich Bildpfade relativ einfach via Aktualisierungsabfrage anpassen.

Beispieldatenbank

Tabellen

tbl100Bildpfade

Die Trennung von Bildpfad und Bildname erleichtert nicht nur die Verwendung im lokalen, sondern auch im Internet-Betrieb. Allerdings sollten für den Web-Auftritt zusätzlich zu den lokalen auch noch die Internet-Pfade hinterlegt werden (im Beispiel ist im Feld „txtPFADWEB“ ein Standardwert hinterlegt).

Formulare

frm100Bildpfade

Bildauswahl

1. Die Auswahl des Bildes erfolgt zunächst mittels dem sicher allseits bekannten Klassenmodul „FileDialog“ von Karsten Pries. An dieser Stelle meinen Dank an Karsten.
2. Der Rückgabewert von „FileDialog.FileName“ wird mittels der Prozeduren „DividePath(FileDialog.FileName)“ und „DivideFileName(FileDialog.FileName)“ in Pfad und Bildname zerlegt (die Prozeduren sind im „mdlGlobal“ enthalten) und den Textfeldern im Formular zugewiesen.
3. Das Bild wird angezeigt, indem der Eigenschaft „Picture“ des Bildsteuerelementes ebenfalls der Wert „FileDialog.FileName“ übergeben wird.
4. Anschließend werden aus dem Bildsteuerelement die Eigenschaften „ImageHeight“ und „ImageWidth“ ausgelesen und in Pixel umgewandelt.
 1. „ImageHeight“ und „ImageWidth“ enthalten Angaben in Twips. Nun könnte man selbstverständlich die Twips anstelle von Pixeln in der Datenbank speichern und sich damit etwas Umrechenarbeit sparen. Dagegen spricht, daß eigentlich nur Access mit Twips umgehen kann. Soll das Bild z.B. auch noch im Web verwendet werden, dann können nur Pixel als Breite und Höhe übergeben werden. Auch wenn die Abmessungen

manuell über ein externes Programm wie z.B. ACDSee ermittelt werden, erhält man von dort immer nur Angaben in Pixel.

```
Private Sub btnGetPic_Click()  
Dim fd As New FileDialog  
With fd  
    .DialogTitle = "Bild auswählen"  
    .Filter1Text = "Alle unterstützten Grafikformate"  
    .Filter1Suffix = "*.bmp;*.wmf;*.gif;*.jpg;*.jpeg"  
    .ShowOpen  
End With  
  
If fd.FileName <> "" Then  
    Me!txtBILD = DivideFileName(fd.FileName)  
    Me!txtPFAD = DividePath(fd.FileName)  
    Me!picBild.Picture = fd.FileName  
    Dim lngHoehe As Long, lngBreite As Long  
    lngHoehe = Me!picBild.ImageHeight  
    lngBreite = Me!picBild.ImageWidth  
    Me!lngHEIGHT = IIf(Int(lngHoehe / 15) < (lngHoehe / 15), Int(lngHoehe / 15) + 1, lngHoehe / 15)  
    Me!lngWIDTH = IIf(Int(lngBreite / 15) < (lngBreite / 15), Int(lngBreite / 15) + 1, lngBreite / 15)  
End If  
  
End Sub
```

Zur Umwandlung in Pixel werden die Twips durch 15 geteilt. Da INT/FIX grundsätzlich abrunden und CINT/CLNG kaufmännisch runden, wird in der Prozedur immer dann, wenn bei der Division Nachkommastellen herauskommen, der Integer-Wert um 1 erhöht.

Bildanzeige

Aktualisierung über das FORM_CURRENT-Ereignis

Das Bildsteuerelement behält die Eigenschaft „Picture“ solange bei, bis ich dieser Eigenschaft einen neuen Wert zuweise. Dazu bietet sich an, jeweils das Ereignis „Beim Anzeigen“ zu nutzen.

1. Zur Vermeidung von Laufzeitfehlern wird zunächst geprüft, ob zu dem angezeigten Datensatz (bei einem neuen Datensatz wäre das ja vermutlich der Fall) keine NULL-Werte in den Feldern mit dem Bildpfad und dem Bildnamen stehen. Laufzeitfehler 94 (Ungültige Verwendung von Null) wird dadurch umgangen.
2. Anschließend wird mit der DIR-Funktion gecheckt, ob das Bild auch körperlich in dem gespeicherten Pfad vorhanden ist und eine Alternative abgearbeitet, falls dem nicht so ist. Damit wird gleichzeitig dem Laufzeitfehler 2220 (Microsoft Access kann die Datei nicht öffnen.) vorgebeugt.
3. Im dritten Schritt wird der „Picture“-Eigenschaft des Bildsteuerelementes Bildpfad und Bildname zugewiesen.

Durch die Schritte 1 und 2 kann in dem Form_Current-Beispiel auf eine weitere Fehlerbehandlung verzichtet werden.

```

Private Sub Form_Current()
'On Error Resume Next
'Erstmal schauen, ob Bildpfad und Bildname vorhanden sind
If Not (IsNull(Me.txtBILD)) And Not (IsNull(Me.txtPFAD)) Then
'Wenn ja, dann Bildpfad und Bildname in einer Variablen zusammensetzen
Dim strBild$
strBild = Me!txtPFAD & Me!txtBILD
'Jetzt nachschauen, ob das Bild auch körperlich vorhanden ist
If Dir(strBild) = "" Then
'Wenn nein, dann Pfad zur aktuellen DB und zum DummyBild
'in die Variable speichern
strBild = DividePath(CurrentDb.Name) & "NotFound.bmp"
End If
'Und nun der Picture-Eigenschaft die Bildvariable zuweisen
Me.picBild.Picture = strBild
Else
'Ansonsten kein Bild anzeigen
Me.picBild.Picture = ""
End If
End Sub

```

Abschneiden / Zoomen / Dehnen

Zur Anpassung/Einpassung in das Bildsteuerelement stehen die 3 Modi Abschneiden, Zoomen oder Dehnen zur Verfügung. In VBA steht dafür die Methode „SizeMode“ zur Verfügung. Mit den drei Schaltflächen auf dem Formular lassen sich die einzelnen Modi auf das Bildsteuerelement anwenden.

```

SizeMode in VBA
Abschneiden = 0
Dehnen = 1
Zoomen = 3 (nicht etwa 2)

```

frm101Bildpfade

Dieses Formular ist weitgehend identisch mit „frm100Bildpfade“. Der Unterschied liegt in der Verwendung eines EditControls zur Ermittlung der Bilddimensionen.

Warum diese Alternative, wenn sich die Bildmaße auch schon aus dem Bildsteuerelement herauslesen lassen?

Access 97 hat, man mag es fast gar nicht glauben, so unwahrscheinlich klingt es, teilweise Probleme mit Bitmaps.

Dies führt zum einen dazu, daß Bitmaps verzerrt dargestellt werden - mag sein, daß das Auge das noch nicht einmal als gravierend wahrnimmt. Es kann aber auch dazu führen, daß die Abmessungen des Bildes völlig falsch interpretiert werden (*Beispiel „NotFound.bmp“*) - mit den entsprechenden Folgen in der Weiterbehandlung.

API-Guru Stephan Lebans hat festgestellt, daß dies an der Interpretation des BITMAPINFOHEADER¹ liegt. Befinden sich dort andere Informationen, als von Access 97 erwartet, werden die entsprechenden Bytes von Access 97 mit Fehlerdaten gefüllt. Stephan meint dazu: „Eine Menge von Bitmaps enthalten Müll oder einfach falsche Daten. Das führt zu falschen Resultaten.“

Das EditControl arbeitet in der Hinsicht korrekt und ermittelt auch für Bitmaps die tatsächlichen Bildmaße.

Was spricht dann dagegen, das EditControl grundsätzlich zu verwenden?

Zunächst einmal: Access kann alle Bilder darstellen, für die der entsprechende Microsoft-Filter installiert oder in der Anwendung vorhanden ist. Das sind mehr Bildtypen, als sie im EditControl darstellbar sind. Die folgende Tabelle enthält eine Gegenüberstellung der anzeigbaren Grafikformate:

	Microsoft-Filter	EditControl-Filter
*.awd	nein	x ²
*.bmp	x	x
*.cdr	x	nein
*.cgm	x	nein
*.drw	x	nein
*.dcx	nein	x
*.dxf	x	nein
*.emf	x	nein
*.eps	x	nein
*.fpx	x	nein
*.gif	x	x
*.jpeg / *.jpg	x	x
*.pcd	x	nein
*.pcx	x	x
*.pict	x	nein
*.png	x	nein
*.tga	x	nein
*.tiff / *.tif	x	x
*.wiff (Wang Image File Format)	nein	x
*.wmf	x	nein
*.wpg	x	nein

¹ BITMAPINFOHEADER->biXPelsPerMeter & biYPelsPerMeter

² nur Windows 95 / 98

*.xif (Xerox Image Format)

nein

x

Werden viele Bitmaps gespeichert, dann würde sich z.B. durchaus der Einsatz des EditControls lohnen. Sind es dagegen viele WMF's, dann ist es eigentlich nicht notwendig.

Das zweite Gegenargument ist Access XP!

In XP scheint das Problem bereinigt. Von Access 97 falsch interpretierte Bitmaps werden dort offensichtlich korrekt dargestellt (*Beispiel: NotFound.bmp*).

Im Beispielformular wird auf die Ermittlung der Bilddimensionen bei Auswahl des Bildes verzichtet.

frm102Bildpfade

Dieses letzte Formular in der Reihe verwendet eine Kombination aus automatischer Ermittlung der Bilddimensionen aus den Eigenschaften des Bildsteuerelementes und ebenfalls automatischer Korrektur via EditControl, falls es sich bei dem gewählten Bild um eine Bitmap handelt.

```
Private Sub btnGetPic_Click()
Dim fd As New FileDialog

With fd
    .DialogTitle = "Bild auswählen"
    .Filter1Text = "Alle unterstützten Grafikformate"
    .Filter1Suffix = "*.bmp;*.wmf;*.gif;*.jpg;*.jpeg"
    .ShowOpen
End With

If fd.FileName <> "" Then
    Me!txtBILD = DivideFileName(fd.FileName)
    Me!txtPFAD = DividePath(fd.FileName)
    Me!picBild.Picture = fd.FileName
    Dim lngHoehe&, lngBreite&
    lngHoehe = Me.picBild.ImageHeight
    lngBreite = Me.picBild.ImageWidth
    Me.lngHEIGHT = IIf(Int(lngHoehe / 15) < (lngHoehe / 15), Int(lngHoehe / 15) + 1, lngHoehe / 15)
    Me.lngWIDTH = IIf(Int(lngBreite / 15) < (lngBreite / 15), Int(lngBreite / 15) + 1, lngBreite / 15)
End If

If InStr(1, fd.FileName, ".bmp") > 0 Then
    With Me.imgEdit1
        .Visible = True
        .Image = fd.FileName
        .Display
    End With
    'Bilddimensionen aus dem EditControl auslesen
    Me.lngHEIGHT = Me.imgEdit1.ImageScaleHeight
    Me.lngWIDTH = Me.imgEdit1.ImageScaleWidth
    'EditControl unsichtbar schalten
    Me.imgEdit1.Visible = False
End If

End Sub
```

Was ist an dieser Stelle noch bei Verwendung des EditControls zu beachten, bevor wir uns an anderer Stelle damit beschäftigen?

Das EditControl zeigt das Bild so lange an, bis ein neues Bild hineingeladen wird. Die Anzeige kann über „Visible = False“ ausgeschaltet werden. Damit jedoch überhaupt ein Bild in das Control hineingeladen werden kann, muß es sichtbar sein – ansonsten kommt es zu einem Laufzeitfehler.

frm110Bildpfade

Sollen mehrere Bilder gleichzeitig dargestellt werden, bietet sich als naheliegende Formularvariante das Endlosformular an.

Im Beispielformular wird aber deutlich, daß ein Bildsteuerelement im Detailbereich eines Endlosformulars ziemlich sinnlos ist. Warum?

1. Das Bildsteuerelement läßt sich nicht an ein Tabellenfeld binden.
2. Ist ihm die Eigenschaft „Picture“ zugewiesen, behält es diese Eigenschaft, bis ihm dafür ein neuer Wert zugewiesen wird und
3. handelt es sich, auch wenn das Endlosformular vielleicht etwas anderes vorgaukelt, nur um „ein einziges“ Bildsteuerelement im Detailbereich und nicht um eines pro Datensatz.

Damit ist das FORM_CURRENT-Ereignis unbrauchbar, weil bereits mit dem ersten Datensatz die „Picture“-Eigenschaft belegt wird. Genauso unbrauchbar ist eine Schaltfläche, mit der das jeweils gewünschte Bild geladen wird, weil dann zwar dem Bildsteuerelement ein neuer „Picture“-Wert zugewiesen wird, aber der dann auch in dem einen, mehrfach dargestellten, Bildsteuerelement angezeigt wird.

frm111Bildpfade

Um dennoch das Bildsteuerelement im Endlosformular sinnvoll nutzen zu können, muß es in einen anderen Formularbereich plziert werden - im Beispiel ist es der Formulkopf. Nun kann auch das FORM_CURRENT-Ereignis wieder verwendet werden, um bei jedem Datensatzwechsel den Inhalt des Bildsteuerelementes anzupassen.

Für das Problem der gleichzeitigen Darstellung mehrerer Bilder muß jedoch eine andere Lösung gefunden werden.

frm120Bildbrowser

Dieses Formular stellt eine solche Lösung vor, wie sich mehrere Bilder neben- und untereinander anzeigen lassen.

Ich habe das Formular „Bildbrowser“ genannt, weil sich mit ihm

1. ein einzelnes Bild anzeigen läßt,

2. ein Verzeichnis komplett in die Datenbank eingelesen werden kann,
3. Bilder aus einem beliebigen Verzeichnis sowie
4. alle in der Datenbank als Textwerte gespeicherten Bilder anzeigen lassen.

Dazu sind auf dem (ungebundenen) Formular 40 Bildsteuerelemente plaziert und fortlaufend numeriert. Im Formularentwurf wird sichtbar, daß noch weitere Textfelder für ID, Beschreibung, Bezeichnung und Pfad, alle ebenfalls mit fortlaufender Numerierung, vorhanden sind.

Das Formular selber enthält einige Prozeduren, aber die wichtigsten sind in dem Modul „mdlBrowsePictures“ enthalten, weil die Anzeige im Formular über die Symbolleiste gesteuert wird.

Wie schon an anderer Stelle bemerkt, behält die „Picture“-Eigenschaft eines Bildsteuerelementes solange den zugewiesenen Wert, bis ihr ein neuer zugewiesen wird. Gerade wenn unterschiedliche Bildbestände durchbrowst werden sollen, ist es daher sinnvoll, zunächst diesen Wert auf einen Leerstring zurückzusetzen.

```
Public Function ClearPictures()
On Error Resume Next
Dim x%
For x = 1 To 40
Forms!frm120Bildbrowser("TN(" & x & ")").Picture = ""
Forms!frm120Bildbrowser("lbl(" & x & ")").Caption = ""
Forms!frm120Bildbrowser("lbl(" & x & ")").Visible = False
Forms!frm120Bildbrowser("ID(" & x & ")").Caption = 0
Forms!frm120Bildbrowser("Bez(" & x & ")").Caption = ""
Forms!frm120Bildbrowser("Pfad(" & x & ")").Caption = ""
Next
Debug.Print SysCmd(acSysCmdClearStatus)
End Function
```

Ein einzelnes Bild anzeigen

Die Prozedur „acGetPictures“ ist weitgehend identisch mit derjenigen, mit der in „frm100Bildpfade“ ein Bild ausgewählt wird.

Ein Verzeichnis in die Datenbank einlesen

Dafür wird eine völlig simple Routine verwendet. Nach Auswahl eines Verzeichnisses (im Beispiel wird das Modul „mdlBrowseFolder“ verwendet) werden mittels DIR-Funktion innerhalb einer Schleife alle Dateien mit den vorgegebenen Namenserweiterungen (bmp, tif, jpg, pcx, bmp, gif und wmf) an die Tabelle „tbl100Bildpfade“ angefügt. Dort können Sie anschließend weiterbearbeitet werden.

```
Public Function gDirToTable()
Dim strFolder$, DSG As Recordset
Dim DN$, Pfad$, x%, strExt$
Set DSG = CurrentDb.OpenRecordset("tbl100Bildpfade")

strFolder = BrowseFolder(DividePath(CurrentDb.Name), _
"Verzeichnis wählen, aus dem die Bilder angezeigt werden sollen.")
```

```

If strFolder = "" Then Exit Function

Pfad = strFolder

If Not (Right$(Pfad, 1)) = "\" Then
    Pfad = Pfad & "\"
End If

DN = Dir(Pfad & "*. *")
Do While DN <> ""
    strExt = Right$(DN, 3)
    If strExt = "bmp" Or strExt = "jpg" Or strExt = "tif" Or strExt = "pcx" Or strExt = "gif" Or strExt = "wmf"
        Then
            DSG.AddNew
            DSG!txtPFAD = Pfad
            DSG!txtBILD = DN
            DSG.Update
        End If
        DN = Dir
    Loop

MsgBox "Das Verzeichnis " & Pfad & " wurde eingelesen."

End Function

```

Diese wirklich simple Prozedur enthält nun keinerlei Möglichkeit, mit Pfad und Bildname auch gleichzeitig die Bilddimensionen zu ermitteln und zum jeweiligen Datensatz abzulegen. Sie sollte daher etwas modifiziert werden:

```

Public Function gfDirToTable()
Dim strFolder$, DSG As Recordset
Dim lngHoehe As Long, lngBreite As Long
Dim DN$, Pfad$, x%, strExt$

Set DSG = CurrentDb.OpenRecordset("tbl100Bildpfade")

strFolder = BrowseFolder(DividePath(CurrentDb.Name), _
    "Verzeichnis wählen, aus dem die Bilder angezeigt werden sollen.")

If strFolder = "" Then Exit Function

Pfad = strFolder

If Not (Right$(Pfad, 1)) = "\" Then
    Pfad = Pfad & "\"
End If

DN = Dir(Pfad & "*. *")
Do While DN <> ""
    strExt = Right$(DN, 3)
    If strExt = "bmp" Or strExt = "jpg" Or strExt = "tif" Or strExt = "pcx" Or strExt = "gif" Or strExt = "wmf"
        Then
            Forms!frm120Bildbrowser("TN(1)").Picture = Pfad & DN
            DSG.AddNew
            DSG!txtPFAD = Pfad
            DSG!txtBILD = DN
        End If
        DN = Dir
    Loop
End Function

```

```

IngHoehe = Forms!frm120Bildbrowser("TN(1)").ImageHeight
IngBreite = Forms!frm120Bildbrowser("TN(1)").ImageWidth
DSG!IngHEIGHT = IIf(Int(IngHoehe / 15) < (IngHoehe / 15), Int(IngHoehe / 15) + 1, IngHoehe / 15)
DSG!IngWIDTH = IIf(Int(IngBreite / 15) < (IngBreite / 15), Int(IngBreite / 15) + 1, IngBreite / 15)
DSG.Update
End If
DN = Dir
Loop

Forms!frm120Bildbrowser("TN(1)").Picture = ""

MsgBox "Das Verzeichnis " & Pfad & " wurde eingelesen."

End Function

```

Um Höhe und Breite der soeben eingelesenen Bilddatei auswerten und speichern zu können, wird das Bild in das erste Bildsteuerelement geladen. Die Twips werden mit der schon im Formular „frm100Bildpfade“ verwendeten Formel in Pixel umgewandelt. Am Ende des Einlesevorgangs wird die „Picture“-Eigenschaft des Bildsteuerelements wieder zurückgesetzt, damit nicht das letzte eingelesene Bild angezeigt bleibt.

Bei dieser Methode muß mit dem Problem der falschen Bitmap-Auswertung gerechnet (Höhe/Breite) und diese ggfs. manuell korrigiert werden.

Ein Verzeichnis anzeigen

Diese Prozedur unterscheidet sich eigentlich nur minimal von der, mit der ein Verzeichnis in die Datenbank eingelesen wird. Der einzige wirkliche Unterschied ist der, daß maximal 40 Bilder (eben so viele, wie Bildsteuerelemente auf dem Formular vorhanden sind) gelesen und angezeigt werden.

```

Public Function ReadDirToThumbs()
On Error Resume Next

Dim strFolder$, strExt$, DN$, x&, Pfad$, BP$(24)

strFolder = BrowseFolder(DividePath(CurrentDb.Name), _
    "Verzeichnis wählen, aus dem die Bilder angezeigt werden sollen.")

If strFolder = "" Then Exit Function

Pfad = strFolder

If Not (Right$(Pfad, 1)) = "\" Then
    Pfad = Pfad & "\"
End If

Debug.Print ClearPictures() 'Erstmal alle Bilder löschen

DN = Dir(Pfad & "*. *")
Do Until x = 40 Or DN = ""
    strExt = Right$(DN, 3)
    If strExt = "bmp" Or strExt = "jpg" Or strExt = "tif" Or strExt = "gif" Or strExt = "wmf" Then
        x = x + 1
        Forms!frm120Bildbrowser("TN(" & x & ")").Picture = Pfad & DN
        Forms!frm120Bildbrowser("Ibl(" & x & ")").Caption = DN
        Forms!frm120Bildbrowser("Ibl(" & x & ")").Visible = True
        Forms!frm120Bildbrowser("ID(" & x & ")").Caption = 0
    End If
Loop
End Function

```

```
Forms!frm120Bildbrowser("Bez(" & x & ")").Caption = ""  
Forms!frm120Bildbrowser("Pfad(" & x & ")").Caption = Pfad & DN
```

```
Debug.Print SysCmd(acSysCmdSetStatus, "Die Datei " & Pfad & DN & " wird eingelesen.")  
End If  
DN = Dir  
Loop  
Debug.Print x  
  
Forms!frm120Bildbrowser.Caption = "Browse Pictures - Bilder aus nicht gespeichertem " & _  
"Verzeichnis anzeigen"  
Debug.Print SysCmd(acSysCmdClearStatus)  
  
End Function
```

Damit die Zuweisung korrekt erfolgt, wird eine Zählervariable (x) bei jedem Schleifendurchlauf um 1 erhöht.

Gespeicherte Bilder anzeigen

Das ist die wohl komplexeste Aufgabe des Bildbrowsers. Warum?

1. Es sind maximal so viele Bilder darstellbar, wie Bildsteuerelemente auf dem Formular plaziert sind.
2. Sind mehr Bilder gespeichert, muß die Möglichkeit geschaffen werden, nach vorne „zu blättern“.
3. Komfortabel (so weit davon gesprochen werden kann) wird der Browser aber erst, wenn ich auch nach hinten, also zurück „blättern“ kann.

Für „frm120Bildbrowser“ ist zusätzlich noch die Möglichkeit vorhanden, nur durch Bilder eines bestimmten Dateityps zu browsen.

Bilder einlesen

Um zunächst die ersten 40 gespeicherten Bilder in die Bildsteuerelemente zu laden, wird die Prozedur „ReadTableToThumbs“ verwendet:

```
Public Function ReadTableToThumbs(numDateityp%)  
On Error GoTo Err_ReadTableToThumbs  
  
booDirTab = False 'Die Merkervariable setzen, um Next und Previous einzuschalten  
  
Dim DSG As Recordset, DSGFilter As Recordset  
Dim x%, BP$(40)  
Dim frmBrowse As Form  
  
Set DSG = CurrentDb.OpenRecordset("tbl100Bildpfade", dbOpenDynaset)  
  
Debug.Print ClearPictures() 'Erstmal alle Bilder löschen  
  
'Objektvariable für das Formular setzen  
Set frmBrowse = Forms!frm120Bildbrowser  
  
Select Case numDateityp  
Case 0 'alle
```

```

TypFilt = "*.*"
Case 1
DSG.Filter = "Right([txtBILD],3) = 'tif'"
TypFilt = "tif"
Case 3
DSG.Filter = "Right([txtBILD],3) = 'bmp'"
TypFilt = "bmp"
Case 6
DSG.Filter = "Right([txtBILD],3) = 'jpg'"
TypFilt = "jpg"
Case 11
DSG.Filter = "Right([txtBILD],3) = 'wmf'"
TypFilt = "wmf"
Case 12
DSG.Filter = "Right([txtBILD],3) = 'gif'"
TypFilt = "gif"
End Select

frmBrowse.Caption = "Browse Pictures - Bilder aus gespeicherten Pfaden anzeigen - Typ: " & TypFilt

Set DSGFilter = DSG.OpenRecordset()

'Sind zum gewählten Typ keine Datensätze vorhanden, dann beenden
If DSGFilter.BOF Then Exit Function

DSGFilter.MoveLast
AnzGes = DSGFilter.RecordCount
intAG = Int(AnzGes)
Debug.Print SysCmd(acSysCmdSetStatus, "Insgesamt " & AnzGes & " Dateipfade gespeichert.")

DSGFilter.MoveFirst
x = 0
Do Until x = 40 Or DSGFilter.EOF
    x = x + 1
    frmBrowse("TN(" & x & ")").Picture = DSGFilter!txtPFAD & DSGFilter!txtBILD
    If Not (IsNull(DSGFilter!cntID)) Then
        frmBrowse("lbl(" & x & ")").Caption = DSGFilter!cntID
    End If
Err_Continue:
    frmBrowse("lbl(" & x & ")").Visible = True
    frmBrowse("ID(" & x & ")").Caption = DSGFilter!cntID
    frmBrowse("Bez(" & x & ")").Caption = Nz(DSGFilter!txtBESCHREIBUNG, "")
    frmBrowse("Pfad(" & x & ")").Caption = Nz(DSGFilter!txtBILD, "")
    DSGFilter.MoveNext
Loop

DSG.Close: DSGFilter.Close

If AnzGes > 40 Then
    Vorschub = 0
End If

Exit Function

Err_ReadTableToThumbs:
Select Case Err
    Case 321, 2114 'Ungültiges Dateiformat

```

```

frmBrowse("lbl(" & x & ")").Caption = "Formatfehler"
Resume Err_Continue

Case Else
    MsgBox Err.Number & " " & Err.Description
    Resume Next
End Select

End Function

```

Zum korrekten Anzeigen der Bilder wird wieder eine Zählervariable (x) verwendet.

Im Deklarationsbereich des Moduls wird die Variable „Vorschub“ deklariert. Mit dem Wert dieser Variablen wird bei den „Blätter“-Prozeduren der Datensatzzeiger positioniert. In der Startprozedur wird dieser Wert explizit auf 0 zurückgesetzt.

Weiterblättern

Soweit ist ein großer Unterschied zur Prozedur, mit der ein beliebiges Verzeichnis eingelesen wird, noch nicht unbedingt erkennbar. Dieser zeigt sich erst, wenn „weitergeblättert“ wird.

```

Public Function NextPictures()
On Error Resume Next

'Ist die Variable "TRUE", also wurde aus Verzeichnis angezeigt, dann verlassen
If booDirTab = True Then Exit Function

Dim DSG As Recordset, DSGFilter As Recordset
Dim x%, BP$(40)
Set DSG = CurrentDb.OpenRecordset("tbl100Bildpfade", dbOpenDynaset)

'Den Typenfilter berücksichtigen
Select Case TypFilt
    Case Is <> "*"
        DSG.Filter = "Right([txtBILD],3) =" & TypFilt & ""
End Select

Set DSGFilter = DSG.OpenRecordset()

'Zum letzten Datensatz gehen und alle Datensätze auslesen
DSGFilter.MoveLast
AnzGes = DSGFilter.RecordCount
intAG = Int(AnzGes)
Debug.Print SysCmd(acSysCmdSetStatus, "Insgesamt " & AnzGes & " Dateipfade gespeichert.")

'Gesamtanzahl < (Seitenvorschub * 40), dann Prozedur verlassen
If AnzGes <= ((Vorschub + 1) * 40) Then Exit Function

'Zum ersten Datensatz zurückspringen
DSGFilter.MoveFirst

'Die globale Variable für den Seitenvorschub hochzählen
Vorschub = Vorschub + 1

'Jetzt alle Bilder löschen
Debug.Print ClearPictures()

'Datensatzzeiger um Seitenvorschub * 40 verschieben

```

```

DSGFilter.Move (Vorschub * 40)
x = 0
Do Until x = 40 Or DSGFilter.EOF
    x = x + 1
    Forms!frm120Bildbrowser("TN(" & x & ")").Picture = DSGFilter!txtPFAD & DSGFilter!txtBILD
    Forms!frm120Bildbrowser("lbl(" & x & ")").Caption = DSGFilter!cntID
    Forms!frm120Bildbrowser("lbl(" & x & ")").Visible = True
    Forms!frm120Bildbrowser("ID(" & x & ")").Caption = DSGFilter!cntID
    Forms!frm120Bildbrowser("Bez(" & x & ")").Caption = Nz(DSGFilter!txtBESCHREIBUNG, "")
    Forms!frm120Bildbrowser("Pfad(" & x & ")").Caption = Nz(DSGFilter!txtPFAD, "")
    DSGFilter.MoveNext
Loop

DSG.Close: DSGFilter.Close

Debug.Print SysCmd(acSysCmdSetStatus, "Angezeigt wid bis Bild " & ((Vorschub * 40) + x) & " von " &
    AnzGes)

End Function

```

Zunächst wird wieder das Recordset geöffnet und die Datensätze nach dem Filter für die Dateitypen gefiltert. Ist der Filter <> „*.““, also ein bestimmter Dateityp gewünscht, wird die in der Variablen „TypFilt“ gespeicherte Dateiendung als Filter für das Recordset verwendet. Die gefilterten Datensätze werden einer neuen Recordset-Variablen zugewiesen.

Anschließend kommt der spannende Teil, in dem die Variable für den Seitenvorschub um 1 erhöht wird. Nach dem Löschen der angezeigten Bilddaten wird der Datensatzzeiger auf den Datensatz, der sich aus (Seitenvorschub-Variable * 40) ergibt, positioniert. Ab dieser Position werden die Datensätze (erneut unter Verwendung der Zählervariablen x) den Formularelementen zugewiesen.

Wird nun weitergeblättert, wird erneut die Seitenvorschub-Variable um 1 erhöht und der Datensatzzeiger entsprechend weiterpositioniert.

Damit am Ende nicht über die letzten Bilder hinausgeblättert werden kann, wird die Prozedur verlassen, wenn die Gesamtanzahl der Datensätze < ((Seitenvorschub-Variable+1) * 40).

Zurückblättern

Der Unterschied zwischen vorwärts und rückwärts blättern liegt im wesentlichen darin, daß beim Zurückblättern die Seitenvorschub-Variable um 1 vermindert wird. Hat diese Variable den Wert 0 wird die Prozedur verlassen.

```

Public Function PreviousPictures()
On Error Resume Next

'Ist die Variable "TRUE", also wurde aus Verzeichnis angezeigt, dann verlassen
If booDirTab = True Then Exit Function

'Ist der Vorschub gleich 0, dann Prozedur verlassen
If Vorschub <= 0 Then Exit Function
'Die globale Variable hochzählen
Vorschub = Vorschub - 1

'Erstmal alle Bilder löschen
Debug.Print ClearPictures()

```

```
Dim DSG As Recordset. DSGFilter As Recordset
Dim x%, BP$(40)
Set DSG = CurrentDb.OpenRecordset("tbl100Bildpfade", dbOpenDynaset)
```

```
'Den Typenfilter berücksichtigen
Select Case TypFilt
  Case Is <> "*" *
    DSG.Filter = "Right([txtBILD],3) =" & TypFilt & ""
End Select
```

```
Set DSGFilter = DSG.OpenRecordset()
```

```
DSGFilter.Move (Vorschub * 40)
```

```
x = 0
```

```
Do Until x = 40 Or DSGFilter.EOF
```

```
  x = x + 1
```

```
  Forms!frm120Bildbrowser("TN(" & x & ")").Picture = DSGFilter!txtPFAD & DSGFilter!txtBILD
```

```
  Forms!frm120Bildbrowser("lbl(" & x & ")").Caption = DSGFilter!cntID
```

```
  Forms!frm120Bildbrowser("lbl(" & x & ")").Visible = True
```

```
  Forms!frm120Bildbrowser("ID(" & x & ")").Caption = DSGFilter!cntID
```

```
  Forms!frm120Bildbrowser("Bez(" & x & ")").Caption = Nz(DSGFilter!txtBESCHREIBUNG, "")
```

```
  Forms!frm120Bildbrowser("Pfad(" & x & ")").Caption = Nz(DSGFilter!txtPFAD, "")
```

```
  DSGFilter.MoveNext
```

```
Loop
```

```
DSG.Close: DSGFilter.Close
```

```
End Function
```

Berichte

rpt100Bildpfade

Um Bilder in Berichten darzustellen, wird prinzipiell genauso vorgegangen, wie in Formularen. In den Detailbereich wird ein Bildsteuerelement plziert. Damit dorthinein ein Bild geladen werden kann, sind im Bericht noch die Felder mit Bildpfad und Bildname notwendig. Anschließend wird der Eigenschaft „Picture“ des Bildsteuerelementes der zusammengesetzte Bildpfad und –name zugewiesen.

Die Zuweisung kann nur „Beim Formatieren“, aber nicht „Beim Drucken“ erfolgen.

```
Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)
```

```
  If Not (IsNull(Me.Bildstring)) Then
```

```
    Me.picBild.Picture = Me.Bildstring
```

```
  Else
```

```
    Me.picBild.Picture = ""
```

```
  End If
```

End Sub

Damit werden alle Bilder angezeigt, aber alle mit den gleichen Abmessungen. Wesentlich interessanter oder eleganter wäre es, die Bildabmessungen dynamisch zu verändern. Dies wird im folgenden Bericht getan.

rpt101Bildpfade

Ein Bildsteuerelement wird im Detailbereich plziert. Da es dynamisch an die Bildgröße angepaßt werden soll, kann es beliebig verkleinert werden. Die Eigenschaften „Vergrößerbar“ und „Verkleinerbar“ des Detailbereichs werden auf „JA“ eingestellt. Für ein Bildsteuerelement sind diese Eigenschaften leider nicht verfügbar.

Datenherkunft für alle Berichte ist die Abfrage „qry100Bildpfade“ . Damit das Bildsteuerelement später im Bericht korrekt vergrößert werden kann, enthält die Abfrage bereits die Formeln, um Höhe und Breite des Bildes wieder von Pixeln in Twips umzurechnen. Höhe bzw. Breite * 15 ergibt nicht 100%ig genau den Twipswert, sollte aber im allgemeinen ausreichen. (Um die Ungenauigkeiten zu vermeiden, könnten zusätzlich zu den Pixelangaben, auch noch die Twips in der Tabelle gespeichert werden.)

Nun muß nur noch die Prozedur für den Detailbereich verändert werden:

```
Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)
```

```
    If Not (IsNull(Me.Bildstring)) Then
        With Me!picBild
            .Height = Me!BildHoehe
            .Width = Me!BildBreite
            .Picture = Me!Bildstring
        End With
    Else
        With Me!picBild
            .Height = 0
            .Width = 0
            .Picture = ""
        End With
    End If
```

```
End Sub
```

rpt110Bildpfade + rpt111Bildpfade

Diese beiden Berichte zeigen in Access 97 (unter NT und W2k) ein Problem.

Mit der Vergrößerung des Bildsteuerelementes wird auch der Detailbereich entsprechend vergrößert. Sind nachfolgende Bildsteuerelemente kleiner, werden die Steuerelemente zwar kleiner angezeigt, aber der Detailbereich behält die letzte Höhe bzw. bisher höchste Höhe.

„rpt110Bildpfade“ zeigt was geschieht, wenn ein relativ großes Bild als erstes angezeigt wird. Für alle folgenden Bilder ist der Detailbereich so groß, wie für das erste Bild.

Als Workaround ist in „rpt111Bildpfade“ die Gruppierungsebene „lngHEIGHT“ mit aufsteigender Sortierung eingefügt. Dadurch werden die Bilder immer mit der kleinsten Höhe beginnend angezeigt. Befriedigend ist dies allerdings auch nicht.

In Access XP (ob in 2000 auch schon, kann ich so nicht sagen) muß dieser Workaround nicht mehr verwendet werden, da sich dort auch die Höhe des Detailbereichs dynamisch ändern läßt (in Access 97 führt der Versuch zu einem Fehler).

```
Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)

    If Not (IsNull(Me.Bildstring)) Then

        With Me!picBild
            .Height = Me!BildHoehe
            .Width = Me!BildBreite
            .Picture = Me!Bildstring
        End With

        With Me.Detailbereich
            .Height = Me!BildHoehe
        End With

    Else

        With Me!picBild
            .Height = 0
            .Width = 0
            .Picture = ""
        End With

    End If

End Sub
```

Problemfall Bildladedialog

Bei allen Bildformaten, für die Access einen Grafikfilter zur Darstellung zwischenschalten muß, poppt im allgemeinen und auch recht kurzfristig ein Ladedialog auf. Dieser Ladedialog kann bei schnellem Blättern in den Bildern dazu führen, daß Access schlicht und wenig ergreifend abstürzt.

Den Sinn des Ladedialoges selber habe ich noch nicht so recht verstanden. Zum Glück läßt er sich allerdings in der Registry ausschalten.

In den Schlüsseln [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\Graphics Filters\Import\] muß beim jeweiligen Grafikformat der Wert für „ShowProgressDialog“ auf „No“ gesetzt werden. Mit diesen Schlüsseln wird der Ladedialog in Access 97 und Access 2000 ausgeschaltet.

Um ihn auch in Access XP auszuschalten, muß dasselbe (zusätzlich) in [HKEY_CURRENT_USER\Software\Microsoft\Shared Tools\Graphics Filters\Import\] und dem jeweiligen Grafikformat gemacht werden.

Wer nicht selber Hand an die Registry legen möchte, kann sich zum gleichen Zweck die Datei FSA.exe (VB6-Runtime erforderlich) bei www.access-guru herunterladen und von dort für die gängigen Grafikformate in HKLM und HKCU den ShowProgressDialog auf beliebig „Yes“ oder „No“ setzen lassen.

Nach Ausschalten des Bildladedialogs stürzt Access auch bei schnellem Blättern nicht mehr ab.

Interne Bilder

OLE-Objekt und Binärspeicherung

OLE war zur Bilddarstellung nur in früheren Versionen von Access die einzige Möglichkeit. Heute sollten OLE-Objekte im herkömmlichen Sinne dafür nicht mehr verwendet werden, da sie die Datenbank unverhältnismäßig aufblähen. Für eine gleichzeitige Darstellung der Bilder sowohl lokal, als auch im Internet ist es völlig ungeeignet. Mit der direkten Speicherung von OLE-Objekten möchte ich mich daher hier auch nicht beschäftigen

Eine sehr interessante Variante dagegen ist, die Bilder zwar in einem Feld vom Typ *OLE-Objekt* zu speichern, aber nicht als OLE-Objekt selber, sondern lediglich mit seinen Binärwerten.

Dabei ergeben sich zwei Hauptprobleme:

1. Wie wandele ich das Objekt in seine Binärwerte um und
2. wie schaffe ich es, die gespeicherten Binärwerte weiterzuverwenden?

Zur Lösung der Probleme stellt Access die Methoden „AppendChunk“ und „GetChunk“ zur Verfügung.

„AppendChunk“ benötigt zur Umwandlung in den Binärwert die Angabe einer Quelle (...AppendChunk Quelle). „GetChunk“ will dagegen wissen, ab welchem Byte und wieviele Bytes insgesamt es zurückwandeln soll (...GetChunk(Offset, Anzahl Bytes).

Zur Verwendung von „AppendChunk“ und „GetChunk“ ein Datenbank-Beispiel:

Beispieldatenbank

Tabellen

tbl200Bilder

Das Feld „oleBILD“ ist vom Typ *OLE-Objekt*.

Formulare

frm200Bilder

Das Formular ist von der Art her ähnlich aufgebaut, wie „frm100Bildpfade“. Zur Darstellung der einzelnen Bilder wird ein simples Bildsteuerelement verwendet. Das Feld „oleBild“ ist nicht im Formularentwurf enthalten.

Die Schaltfläche, mit der ein neues Bild gespeichert wird, befindet sich im Formulkopf, weil im Beispiel nur Bilder angefügt werden.

Was geschieht, wenn auf die Schaltfläche gedrückt wird?

Ein Bild speichern

```
Private Sub btnGetPic_Click()  
On Error Resume Next  
Dim fd As New FileDialog  
  
With fd  
    .DialogTitle = "Bild auswählen"  
    .Filter1Text = "Alle unterstützten Grafikformate"  
    .Filter1Suffix = "*.bmp;*.wmf;*.gif;*.jpg;*.jpeg"  
    .ShowOpen  
End With  
  
If fd.FileName <> "" Then  
    Call BinaryImport(fd.FileName)  
    Me!txtBESCHREIBUNG.SetFocus
```

```
RunCommand acCmdRecordsGoToLast
```

```
Call BinExport(DividePath(CurrentDb.Name) & "temp" & Right(Me!txtDATEI, 4))  
Me!picBild.Picture = DividePath(CurrentDb.Name) & "temp" & Right(Me!txtDATEI, 4)
```

```
End If
```

```
End Sub
```

Mit dem FileDialog wird zunächst das Bild ausgewählt.

Anschließend wird über die Prozedur „BinaryImport“ (Dank an Günther Ritter) das Bild in die Binärdaten umgewandelt und diese im Tabellenfeld „oleBILD“ abgelegt. Die Prozedur erwartet als Übergabeargument einen Dateinamen; dafür dient „FileDialog.FileName“.

```
Function BinaryImport(strDatei As String)  
Dim DSG As Recordset  
Dim Nr As Long  
  
Set DSG = CurrentDb.OpenRecordset("tbl200Bilder")  
  
Nr = FreeFile  
  
Open strDatei For Binary As #Nr  
ReDim BinaryData(LOF(Nr))  
  
Get #Nr, , BinaryData()  
  
DSG.AddNew  
DSG!lngGROESSE = LOF(Nr)  
DSG!txtDATEI = strDatei  
DSG!oleBILD.AppendChunk BinaryData  
DSG.Update  
Close #Nr  
  
Erase BinaryData  
  
DSG.Close  
Set DSG = Nothing  
  
Me.RecordSource = "tbl200Bilder"  
  
End Function
```

Im Deklarationsbereich des Formulklassenmoduls ist die (dynamische) Variable „BinaryData()“ als Datentyp *Byte* deklariert.

Was passiert nun in der Importprozedur?

Nach Zuweisung der Objektvariablen für das Recordset wird zunächst über die Funktion „FreeFile“ die nächste freie Dateinummer der Variablen „Nr“ zugewiesen. Anschließend wird die übergebene Datei im Binärmodus³ geöffnet.

³ Im Binärmodus werden Dateien byteweise gelesen und geschrieben. Zum Lesen dient GET und zum Schreiben PUT.

Die Variable „BinaryData“ wird unter Verwendung der „LOF“-Funktion auf die Größe der geöffneten Datei neu dimensioniert. Den eigentlichen Einlesevorgang erledigt im Binärmodus die „GET“-Anweisung, der eine Variable, in dem Fall „BinaryData()“, genannt werden muß, in der die Daten zwischengespeichert werden.

Ist dies erledigt, kommt „AppendChunk“ ins Spiel.

Dazu wird in das Recordset-Feld „oleBILD“ mit „AppendChunk BinaryData“ der zwischengespeicherte Variableninhalt geschrieben.

Damit ist das Bild in der Datenbank binär gespeichert.

Ein Bild anzeigen

Damit das binär gespeicherte Bild im Bildsteuerelement angezeigt werden kann, muß es wieder zurückverwandelt werden (in der Prozedur „btnGetPic_Click“ gelb unterlegt).

Diese Aufgabe erledigt „GetChunk“.

```
Function BinExport(strDatei As String)
Dim DSG As Recordset
Dim Nr As Long, BinaryData() As Byte

Set DSG = CurrentDb.OpenRecordset("tbl200Bilder", dbOpenDynaset)
DSG.Filter = "[cntID] = " & Me.cntID
Set DSG = DSG.OpenRecordset()

Nr = FreeFile

Open strDatei For Binary As #Nr
  ReDim BinaryData(DSG!oleBILD.FieldSize)

  BinaryData() = DSG!oleBILD.GetChunk(0, DSG!oleBILD.FieldSize)
  Put #Nr, , BinaryData()
Close #Nr

Erase BinaryData

DSG.Close: Set DSG = Nothing

End Function
```

In der Funktion „BinExport“ (erneuten Dank an Günther Ritter) wird das Recordset zuerst auf den aktuell angezeigten Autowert gefiltert. „BinExport“ erwartet als Übergabeparameter einen Dateinamen als String.

Nach Ermittlung der nächsten freien Dateinummer wird die übergebene Datei erneut im Binärmodus geöffnet. Die Neudimensionierung der Variablen „BinaryData“ erfolgt in diesem Fall unter Verwendung der Eigenschaft „FieldSize“ des OLE-Feldes.

„BinaryData“ wird mit „GetChunk“ der Inhalt des OLE-Feldes von Byteposition 0 bis zur Feldgröße übergeben, bevor der Variablenwert mit „PUT“ in die geöffnete Datei geschrieben wird. Damit ist das ursprüngliche Bild wieder auf der Festplatte als Bild vorhanden.

Die eigentliche Anzeigeprozedur unterscheidet sich nicht von der, mit der ein beliebiges externes Bild angezeigt wird.

Sollen nun die einzelnen Bilder jeweils zum Datensatz angezeigt werden, ist im FORM_CURRENT-Ereignis das Bild zuerst aus den Binärdaten zurückzuverwandeln und auf Festplatte abzulegen.

```
Private Sub Form_Current()  
  
If Me.txtDATEI <> "" Then  
    If Not IsNull(Me.cntID) Then  
        Call BinExport(DividePath(CurrentDb.Name) & "temp" & Right(Me.txtDATEI, 4))  
        Me.picBild.Picture = DividePath(CurrentDb.Name) & "temp" & Right(Me.txtDATEI, 4)  
    End If  
End If  
  
End Sub
```

Im Beispiel werden die Bilder immer als „temp“ gespeichert und beim Schließen des Formulars von der Festplatte gelöscht.

Auf diese Weise wird erreicht, daß bei geöffneter Datenbank maximal so viele temporäre Bilddateien auf der Festplatte vorhanden sind, wie es gespeicherte Dateitypen gibt.

Was sind die Vor- und Nachteile von Binär- und Externer Speicherung?

Binärspeicherung bewirkt, daß das Bild **immer** verfügbar ist - auch wenn es nicht mehr auf Festplatte existiert. Wurde das Bild auf der Platte gelöscht, kann ich es aus der Datenbank jederzeit wiederherstellen. Ich habe also quasi eine Sicherungskopie. Dafür muß ich in Kauf nehmen, daß die Datenbank zwar nicht extrem aufgebläht wird, aber doch um die Größe des Bildes „wächst“. (*Beispiel: Datenbank komprimieren – Größe zeigen - Einfügen von Skizze_Männchen.tif – Größe zeigen – Datensatz löschen – Datenbank komprimieren – Größe zeigen*)

Letztlich muß ich damit rechnen, daß meine Datenbank deutlich schneller die Maximalgröße erreicht, als wenn ich lediglich einen Verweis auf die Bilddatei in Form des Bildpfades hinterlege. (*Beispiel: Datenbank komprimieren – Größe zeigen - Einfügen von Skizze_Männchen.tif – Größe zeigen – Datensatz löschen – Datenbank komprimieren – Größe zeigen*)

Mein Fazit: Binärspeicherung ist gute Alternative, wenn man eine Datenbank mit festen Bildbeständen weitergeben will. Für eine variable Bildverwaltung mit vielen Bildern unterschiedlicher Größen bevorzuge ich den Verweis auf das Bild.

Scannen mit Access

Bestandteile des Betriebssystems nutzen

Zu einer vernünftigen Bildverwaltung mit Access gehört zweifelsohne auch die Möglichkeit, direkt aus Access heraus einen Scanner anzusteuern und das Scanergebnis in die Datenbank einfließen zu lassen.

Am schönsten wäre es, dazu direkt die TWAIN.DLL verwenden zu können. Nur leider ist mir dies bisher noch nicht gelungen und deshalb kann ich dazu auch keine Lösung bieten.

Für VB wird als eine solche Möglichkeit z.B. ezTwain mit der eztw32.dll verkauft, wobei mich die offene Version nicht überzeugen konnte.

Aber warum zu diesem Zweck nicht auf die Möglichkeiten zurückgreifen, über die jeder Windows-Rechner seit Windows 95b automatisch verfügt – nämlich die „Imaging-Steuerelemente“?

Um es gleich vorwegzunehmen: Imaging ist eine sehr gute und kostenlose Möglichkeit. Aber leider, leider sind die Imaging-Steuerelemente eigentlich mit jeder Betriebssystemversion verändert worden. Und damit untereinander nicht beliebig austauschbar.

Zur Geschichte von Imaging

Alle Informationen dazu selbstverständlich nur, soweit sie mir bekannt sind.

In Windows 95 und 95a war Microsoft FAX für das Versenden und Empfangen von Faxdokumenten zuständig. Faxdokumente trugen die Dateierdung „awd“.

Bereits zu Windows 95a-Zeiten stellte Microsoft „Wang Imaging“ zum kostenlosen Download bereit. Nach Installation von „Wang Imaging“ wurden Faxversand und –empfang nun nicht mehr von Microsoft FAX, sondern von Imaging abgewickelt. Imaging konnte Dateien des Typs „awd“ und zusätzlich der Typen „tif“ und „bmp“ schreiben. Lesen konnte es weitere Dateitypen.

Ab Windows 95b wurde „Imaging“ Standard für die Faxbehandlung.

Windows 95b und NT 4.0 enthalten „Wang Imaging“.

Nach Übernahme der Wang Laboratories durch Kodak heißt Imaging seit Windows 98 nunmehr „Kodak Imaging“ (ob das Produkt dadurch besser geworden ist, soll nicht diskutiert werden).

Nach der, ich nenne sie einmal, Zumutung mit der zwischenzeitlichen Symantec Fax Starter Edition ist ab Windows 2000 wieder Microsoft Fax aktiv. Zuständig für die Abwicklung ist im Hintergrund aber weiterhin Imaging. Das alte Fax-Format „awd“ wurde durch „tif“ abgelöst. Unter Windows 2000 stellt Imaging zum Schreiben die Formate „tif“ und „bmp“, sowie „jpg“ zur Verfügung. Durch die Möglichkeit Bilder im JPG-Format zu speichern, ist Imaging nun erst richtig vernünftig anwendbar.

Die Imaging-Steuerelemente

Mit jeder installierten Imaging-Version werden auch die Imaging-Steuerelemente mitinstalliert. Darüber lassen sich Scanner ansteuern, Bilder verwalten und sogar Bilder bearbeiten.

Zur Kompatibilität

Wie erwähnt sind die Steuerelemente der einzelnen Imaging-Ausgaben im allgemeinen untereinander nicht austauschbar.

Lediglich die „Wang-Imaging-Controls“ von Windows 95 und NT 4.0 können auf beiden Betriebssystemen problemlos (mir sind zumindest noch keine Probleme aufgefallen) ausgetauscht werden. „Wang-Imaging-Controls“ werden unter Windows 98 teilweise toleriert.

Das hat zur Folge, daß Entwicklungen die die Imaging-Controls verwenden, auf den gleichen Betriebssystemen vorgenommen werden müssen, auf denen die Anwendung später auch laufen soll. Die Imaging-Controls können „nicht“ in das Setup der ODE oder MOD aufgenommen werden! Auch bei Berücksichtigung dessen kann trotzdem nicht garantiert werden, daß die Steuerelemente auch auf dem Anwendungs-System einwandfrei erkannt werden.

Eine teure Angelegenheit, um diese Probleme zu umgehen, ist der Einsatz von „Imaging Professional“. Für jeden Rechner, auf dem „Imaging Professional“ verwendet werden soll, ist eine separate Lizenz zu erwerben und „Imaging Professional“ zu installieren.

Weitere Informationen

Besitzer von Windows 95/98/NT4 können alle Eigenschaften, Methoden und Ereignisse für die Imaging-Controls den Hilfedateien

1. wangocxd.hlp oder
2. imgocxd.hlp

entnehmen. In Windows 2000 sind diese Hilfedateien wohl nicht mehr enthalten.

Beispieldatenbank

Formulare

frmScanControl

Bei diesem Formular handelt es sich um eine „abgespeckte“ Version des gleichen Formulars aus „ezScan“. Abgespeckt in dem Sinne, daß verschiedene Funktionalitäten nicht enthalten sind, sondern nur diejenigen, die zum Verständnis der Imaging-Steuerelemente notwendig sind.

Das Beispielformular selber wird über die Symbolleiste gesteuert. Neben einigen Prozeduren im Formular-Klassenmodul finden sich die anderen Prozeduren im Modul „mdlScanControl“.

Entwurfsansicht

Bevor auf die Steuerelemente zugegriffen werden kann, müssen Sie im Formular plziert werden.

- ◆ Für den Scanvorgang als solches wird ein „Steuerung für Kodak-Bildscan“-Control (ScanControl) benötigt.
- ◆ Zum Laden und Speichern von Bildern dient ein „Steuerung für Kodak-Bildverwaltung“-Control (AdminControl).
- ◆ Zum Einfügen von Seiten in ein geladenes Bild wird ein weiteres „Steuerung für Kodak-Bildverwaltung“-Steuerelement verwendet (allerdings im Beispiel ohne Funktion).
- ◆ Die Bildanzeige und -bearbeitung letztlich erfolgt in einem „Steuerung für Kodak-Bildbearbeitung“-Control (EditControl).

Nur das EditControl ist in der Formularansicht sichtbar, während ScanControl und AdminControl unsichtbar bleiben. Deshalb wird die Größe des EditControls auf den erwarteten Anzeigebereich angepaßt.

Die Controls lassen sich mit Voreinstellungen belegen, die zur Laufzeit überschreibbar sind.

0 – Display only	Benötigt als Zielangabe ein EditControl in dem das Scanergebnis angezeigt werden kann. Wird, sofern ein EditControl auf dem Formular vorhanden ist, automatisch eingestellt. Bei mehreren EditControls kann aus der Dropdown-Box ausgewählt werden, wo die Anzeige erfolgen soll.
1 – Display and File	Benötigt zusätzlich zum Ziel-EditControl noch die Angaben <ul style="list-style-type: none"> ◆ unter welchem Namen die Datei abgespeichert werden soll und ◆ ob das Scanergebnis an die Datei angehängt oder eine vorhandene Seite überschrieben werden soll
2 – File only	Benötigt kein Ziel-EditControl, sondern nur die Angaben zur Datei.
3 – Display and Use File Template	Benötigt Angaben zum Ziel-Edit-Control und zur Dateivorlage. Dateivorlage bewirkt, daß der Name der Zieldatei bei jedem Scan um 1 erhöht wird. (Funktioniert nicht mit JPEG.)
4 – Use File Template Only	Benötigt die Angabe zur Dateivorlage.
5 – Fax Only	Faxt das eingescannte Bild. Dafür wird eine Dateivorlage benötigt.

Voreingestellt werden kann außerdem der Bildtyp (TIF, BMP oder JPG), sowie ob das Scanner-Setup angezeigt werden soll.

Voreinstellungen des EditControls

Die Voreinstellungsmöglichkeiten sind zu umfangreich, um hier im einzelnen darauf eingehen zu können.

Unter anderem läßt sich voreinstellen,

mit welchem Zoom-Faktor das Bild angezeigt werden soll,

ob das EditControl Scrollbars enthalten,

ob per Tastatur-Shortcut gescrollt werden kann und

welche Bildpalette angewendet werden soll.

Zusätzlich läßt sich hierüber auch das „Steuerung für Kodak-Bildanmerkung“-Control voreinstellen. Bei diesem Control handelt es sich um ein Subcontrol des EditControls.

Voreinstellungen des AdminControls

Das AdminControl wird sowohl zum Laden und Speichern von Bildern (Dokumenten), als auch für den Druckvorgang und die kontextbezogene Hilfe benötigt.

Der Dateidialog entspricht weitgehend dem des CommonDialog-Controls.

Im Beispiel können alle Voreinstellungen übernommen werden, da die Steuerung über Programmcode erfolgt.

Einmal scannen bitte

Mit der Schaltfläche in der Symbolleiste wird die Prozedur „scQuickScan“ aufgerufen.

```
Public Function scQuickScan(numFT%)
On Error Resume Next
'Dateitypen: 1 - TIFF (Standard) / 3 - BMP / 6 - JPEG
Dim frmSC As Form
Set frmSC = Forms!frmScanControl

'Die Gespeichert-Variable zurücksetzen
fSaved = False

With frmSC!ScanCtl
.FileType = numFT
.ShowSetupBeforeScan = Nz(DLookup("fSHOWSETUPBEFORESCAN", "tbl910Optionen"), True)
.StartScan
.DestImageControl = "ImgEdit1"
End With

End Function
```

Als Übergabeparameter erwartet die Prozedur einen der drei Dateitypen, die Imaging schreiben kann. Dieser Dateityp wird der Eigenschaft „FileType“ des ScanControls übergeben.

Über die folgende Eigenschaft „ShowSetupBeforeScan“ (deren Wert im Beispiel aus einer Tabelle ausgelesen wird), wird das ScanControl angewiesen, entweder das Scanner-Setup anzuzeigen oder nicht. Als Scanner-Setup in dem Sinne wird die mit dem Scanner für den Scanvorgang gelieferte Software deklariert. Sofern es sich nicht um Dokumente handelt, die immer über die ganze Auflagefläche gescannt werden, ist es immer sinnvoll „ShowSetupBeforeScan“ auf True zu setzen. Insbesondere bei Bildern lassen sich dann die entsprechenden Scanabmessungen und Feineinstellungen noch vor dem eigentlichen Scan vornehmen.

Gestartet wird der Scan mit der Methode „StartScan“.

Die Übergabe von „DestImageControl“ ist im Beispiel überflüssig, da alle nicht via Code eingestellten Eigenschaften und Methoden aus den Voreinstellungen herausgezogen werden.

Ich möchte aber zweimal scannen...

Einfachste Variante: Den Button ein zweites Mal drücken.

Eine andere Möglichkeit bietet die Verwendung der Methode „ShowScanNew“. Allerdings lassen sich damit nur Seiten in eine Multipage-Datei (TIF) einfügen. JPG und BMP können nicht mehrere Seiten enthalten. Deshalb ist diese Methode für solche Dateien ungeeignet.

```

Public Function scShowScanNew()
On Error Resume Next
Dim frmSC As Form
Set frmSC = Forms!frmScanControl

'Erstmal einen Bildnamen vergeben
frmSC!ImgAdm1.ShowFileDialog 1
'Jetzt den Dateinamen dem Bild zuweisen und ScanNew-Dialog anzeigen
With frmSC!ScanCtl
.Image = frmSC!ImgAdm1.Image
.showscannew
End With

End Function

```

Um dem ScanNew-Dialog einen Dateinamen zuzuweisen, unter dem das Bild gespeichert werden soll, wird im Beispiel das AdminControl verwendet und der Eigenschaft „Image“ des ScanControls die gleichnamige Eigenschaft des AdminControls zugewiesen.

Dieses Vorgehen ist nicht unbedingt erforderlich. Im Bedarfsfall kann der Bildname auch im Dialog angegeben werden.

Mit jedem Drücken der „Scannen“-Schaltfläche wird eine Seite an die Datei angefügt. Über die Schaltfläche „Neu scannen“ kann der Scanvorgang für die zuletzt gescannte Seite wiederholt werden.

Leider ist gerade der Scanner irgendwie abgestürzt...

Mit der Methode „ResetScanner“ des ScanControls lassen sich Hard- und Software zurücksetzen.

Damit sollen die (sowieso recht beschränkten) Möglichkeiten des ScanControls an dieser Stelle als abgeschlossen angesehen werden.

Das Bild soll weiterverarbeitet werden

Wie beim ScanControl beschrieben, kann das Scanergebnis direkt in eine Datei geschrieben werden. Das mag unter manchen Umständen sinnvoll sein, aber vermutlich nur in sehr seltenen Fällen. Die Regel dürfte vielmehr sein, daß das Scanergebnis vor Einpflege in die Datenbank zumindest noch einmal gesichtet werden soll.

Dafür ist das EditControl zuständig.

Mit der Funktion „scQuickScan“ wurde das Bild in das EditControl gescannt und wird nun dort angezeigt.

Zoomen

In das EditControl geladene Bilder können stufenlos von 2 bis 6.500 Prozent gezoomt werden. Im Falle des Beispielformulars stehen zum Zoomen verschiedene Möglichkeiten zur Verfügung:

```

Public Function ecZoom(numWert%)
'Für das Kontextmenü werden feste Werte verwendet
'On Error Resume Next
Dim frmSC As Form
Set frmSC = Forms!frmScanControl
With frmSC!ImgEd1
.Zoom = numWert
.AutoRefresh = True

```

```
End With  
End Function
```

In dieser Funktion wird der Zoomfaktor auf den Wert eingestellt, der in dem Formularfeld „In %“ ausgewählt oder eingegeben wird.

```
Public Function ecZoomPlus()  
'Verdoppelung des aktuellen Zoomwertes  
On Error Resume Next  
Dim frmSC As Form  
Set frmSC = Forms!frmScanControl  
  
'Ist kein Bild angezeigt, dann Prozedur verlassen  
If frmSC!ImgEd1.ImageDisplayed = False Then Exit Function  
  
Dim numZwiVar%  
  
numZwiVar = frmSC!ImgEd1.Zoom  
'Wenn 0 als Ergebnis, dann verdoppeln  
If numZwiVar = 0 Then numZwiVar = 100  
  
With frmSC!ImgEd1  
    .Zoom = Int(numZwiVar * 2)  
    .Refresh  
End With  
  
End Function
```

Die Funktion hinter der „Plus-Lupe“ in der Symbolleiste schaut zunächst, ob überhaupt ein Bild angezeigt wird („ImageDisplayed“). Ist dies der Fall, wird der aktuelle Wert der Zoom-Eigenschaft des EditControls ausgelesen (ImgEd1.Zoom), verdoppelt und der Zoom-Eigenschaft erneut zugewiesen. Damit die Änderung auch sichtbar wird, muß als letztes die Methode „Refresh“ angewendet werden.

```
Public Function ecZoomMinus()  
'Halbierung des aktuellen Zoomwertes  
On Error Resume Next  
Dim frmSC As Form  
Set frmSC = Forms!frmScanControl  
  
'Ist kein Bild angezeigt, dann Prozedur verlassen  
If frmSC!ImgEd1.ImageDisplayed = False Then Exit Function  
  
Dim numZwiVar%  
  
numZwiVar = frmSC!ImgEd1.Zoom  
  
With frmSC!ImgEd1  
    .Zoom = Int(numZwiVar / 2)  
    .Refresh  
End With  
  
End Function
```

Bei der „Minus-Lupe“ wird der in der Zoom-Eigenschaft gespeicherte Wert einfach halbiert.

Die letzte für das Beispielformular verfügbare Variante ist der Zoom auf Auswahlgröße. Standardmäßig kann mit dem Mauszeiger auf dem EditControl ein Rechteck aufgezogen werden (man kann noch mehr Sachen für die Maus einstellen, aber das ist hier nicht Gegenstand).

```
Public Function ecZoomToSelection()  
On Error Resume Next  
Dim frmSC As Form  
Set frmSC = Forms!frmScanControl  
  
'Ist kein Bild angezeigt, dann Prozedur verlassen  
If frmSC!ImgEd1.ImageDisplayed = False Then Exit Function  
  
    frmSC!ImgEd1.ZoomToSelection  
  
End Function
```

Die Methode „ZoomToSelection“ zoomt das Bild so in das EditControl, daß es entweder in Höhe oder Breite in das Control eingepaßt wird. Darauf, welche Einpassung vorgenommen wird, kann von außen kein Einfluß genommen werden.

EIGENSCHAFT ODER METHODE

Bei den ersten drei Möglichkeiten (Fester Wert, Plus-Lupe, Minus-Lupe) wird die *Eigenschaft* „Zoom“ verändert. Deshalb muß anschließend ein „Refresh“ erfolgen. Bei der Methode „ZoomToSelection“ ist ein solcher „Refresh“ nicht erforderlich.

Was bedeutet das und wie wirkt es sich aus?

„ZoomToSelection“ verändert nur die Anzeige, während „Zoom“ das Bild selber verändert!

Beim Speichern eines Bildes aus dem EditControl heraus, muß der User entscheiden, ob er das Bild mit dem Zoomfaktor speichern will, mit dem es gescannt wurde, oder mit dem aktuell zugewiesenen Zoomwert. Sprich, ist der aktuelle Zoomfaktor 200% und das Bild wird damit gespeichert, dann ist das Bild auch tatsächlich/körperlich 200% groß.

„ZoomToSelection“ kann bedenkenlos und beliebig oft ausgeführt werden. Es wirkt sich nicht auf die Größe aus, mit der das Bild gespeichert wird.

Das Bild speichern

Damit wird der wohl komplexeste Vorgang im Beispiel ausgelöst.

```
Public Function ecSaveImage(strType$, fThumb As Boolean)  
On Error GoTo Err_SaveImage  
Dim fd As New FileDialog  
Dim frmSC As Form, frmSP As Form  
Set frmSC = Forms!frmScanControl  
  
'Wird im Image Edit Control kein Bild angezeigt, dann Prozedur verlassen  
If frmSC!ImgEd1.ImageDisplayed = False Then  
    MsgBox "Für das Ausführen dieser Prozedur muß ein Bild angezeigt sein.", vbInformation  
    Exit Function  
End If  
  
'Das Dialogformular zur Auswahl öffnen
```

```
DoCmd.OpenForm "fdlgSaveAs" . . . . acDialog
```

```
'Wurde Dialogformular nicht geschlossen, dann....
```

```
If IsOpenForm("fdlgSaveAs") = True Then
```

```
Dim strFN$, numFileType%, numPageType%, fZoom As Boolean
```

```
Dim numCompType%, numCompInfo%
```

```
numFileType = Nz(Forms!fdlgSaveAs!cboFileType, 6)
```

```
numPageType = Nz(Forms!fdlgSaveAs!cboPageType, 6)
```

```
numCompType = Nz(Forms!fdlgSaveAs!cboCompression, 6)
```

```
numCompInfo = Nz(Forms!fdlgSaveAs!cboCompressionInfo, 4096)
```

```
fZoom = Forms!fdlgSaveAs!opgZoom
```

```
With fd
```

```
.DialogTitle = "Bild / Dokument speichern"
```

```
.Filter1Text = "TIFF (*.tif)"
```

```
.Filter1Suffix = "*.tif"
```

```
.Filter2Text = "Bitmap (*.bmp)"
```

```
.Filter2Suffix = "*.bmp"
```

```
.Filter3Text = "JPEG (*.jpg)"
```

```
.Filter3Suffix = "*.jpg"
```

```
.DefaultFileName = "NewPic"
```

```
.InitDir = Nz(DLookup("txtSTANDARDPFAD", "tbl910Optionen"), CurDir)
```

```
.ShowSave
```

```
End With
```

```
If fd.FileName = vbNullString Then
```

```
MsgBox "Operation abgebrochen", vbInformation
```

```
Exit Function
```

```
End If
```

```
strFN = fd.FileName
```

```
If MsgBox("Anmerkungen in Bild einbrennen?@(Für TIFF und AWD nicht notwendig.)@" _  
, vbYesNo + vbQuestion) = vbYes Then
```

```
frmSC!ImgEd1.BurnInAnnotations 2, 2
```

```
End If
```

```
With frmSC!ImgEd1
```

```
.ImageControl = "ImgEdit1"
```

```
.Display
```

```
Select Case strType
```

```
Case "SavePage"
```

```
.SavePage strFN, numFileType, numPageType, numCompType, numCompInfo, fZoom
```

```
Case "SaveAs"
```

```
Select Case fThumb
```

```
Case True
```

```
'Die Zoom-Vorgabe umsetzen
```

```
Dim sngZoomF As Single
```

```
Select Case frmSC!opgZoom
```

```
Case 1 'Feste Breite
```

```
sngZoomF = Int((frmSC!txtZoomFixedWidth * 100) / frmSC!ImgEd1.ImageWidth)
```

```
Case 2 'Feste Höhe
```

```
sngZoomF = Int((frmSC!txtZoomFixedHeight * 100) / frmSC!ImgEd1.ImageHeight)
```

```
Case 3 'Faktor in %
```

```
sngZoomF = frmSC!comZoom
```

```
End Select
```

```

Thumbnail zoomen
With frmSC!ImgEd1
    .ImageControl = "ImgEdit1"
    .ZoomMode = 1 'Inch-to-Inch
    .Zoom = sngZoomF
End With

'Datei und Thumb speichern
With frmSC!ImgEd1
    .ImageControl = "ImgEdit1"
    'Originalgröße
    .SaveAs strFN, numFileType, numPageType, numCompType, numCompInfo, False
    'Thumbnail
    .SaveAs "TN" & strFN, numFileType, numPageType, numCompType, numCompInfo,
    True
End With
Case False
    .SaveAs strFN, numFileType, numPageType, numCompType, numCompInfo, fZoom
End Select
End Select
End With

If MsgBox("Soll das gescannte Bild in die Datenbank übernommen werden?", _
    vbYesNo + vbQuestion) = vbYes Then
    Dim strFP$
    strFN = DivideFileName(fd.FileName)
    strFP = DividePath(fd.FileName)

    DoCmd.OpenForm "frmSavedPictures", acNormal, , , acFormAdd
    Set frmSP = Forms!frmSavedPictures
    frmSP!txtBILD = strFN
    frmSP!txtPFAD = strFP
    frmSP!picBild.Picture = strFP & "\" & strFN
End If

DoCmd.Close acForm, "fdIlgSaveAs"
End If

Exit Function

Err_SaveImage:
Select Case Err
    Case 1001, 1002, 2114 'Bild nicht angezeigt/Bildtyp wird nicht unterstützt
        Exit Function
    Case 32755 'Abbrechen
        Exit Function
    Case Else
        MsgBox Err.Number & " " & Err.Description
        Resume Next
End Select

End Function

Public Function ecShowAnnotationToolPalette()
On Error Resume Next
Dim frmSC As Form

```

```
Set frmSC = Forms!frmScanControl
  With frmSC!ImgEd1
    .ImageControl = "ImgEdit1"
    .ShowAnnotationToolPalette
```

```
End With
```

```
End Function
```

Ich habe die Prozedur komplett so gelassen, wie sie auch in ezScan enthalten ist.

Das Formular „fdlgSaveAs“ ist in der Beispieldatenbank enthalten (die Tabellen als Grundlage für die Kombinationsfelder des Formulars aber nicht).

Im Dialogformular wird festgelegt, mit welchem Dateityp (FileType), Seitentyp (PageType) und mit welcher Kompression (CompressionType, CompressionInfo) das Bild gespeichert werden soll. Seitentyp und Dateityp unterscheiden sich dahingehend, daß ich als Dateityp zwischen TIF, BMP und JPG auswählen kann; für den Seitentyp kann festgelegt werden, ob es sich u.a. um ein Bild mit 24-Bit-Farben, 8-Bit-Farben oder auch Graustufen handelt.

Weiterhin kann festgelegt werden, ob nun das Bild als solches oder auch noch ein Thumbnail des Bildes gespeichert werden soll.

Anschließend wird der FileDialog zum Speichern aufgerufen.

Danach kommt die Nachfrage, ob Anmerkungen auf das Bild mit eingebrannt werden sollen oder nicht.

Was bedeutet das?

Anmerkungen können mit dem Subcontrol des EditControls auf einem Bild angebracht werden. Das Format TIFF speichert Annotation Data separat zum Bild, so daß Anmerkungen in einem TIF-Bild auch bei einem späteren Aufruf bearbeitet werden können.

Andere Formate verfügen nicht über diese Möglichkeit. Damit Anmerkungen untrennbar mit dem Bild verbunden werden, müssen sie deshalb in das Bild „gebrannt“ werden. Werden sie nicht „eingebrannt“, sind sie beim Speichervorgang verloren.

Der Methode „BurnInAnnotations“ können verschiedene Parameter übergeben werden. In der Beispielprozedur wird das Control angewiesen, alle markierten Anmerkungen in der dargestellten Form einzubrennen.

Nun wird über eine SELECT-CASE-Anweisung ausgewählt, was eigentlich gespeichert werden soll. Ob es sich um eine einzelne Seite handelt, die an eine Multipage-Datei angefügt wird, oder ob die Datei als solches gespeichert werden soll.

Ist letzteres ausgewählt, also das Speichern der Datei als solches, wird erneut über ein SELECT-CASE verzweigt, ob nur das einzelne Bild oder auch gleichzeitig (hintereinander) ein Thumbnail mit gespeichert werden soll.

Es wird festgestellt, welche Zoomvariante im Formular gedrückt ist („Feste Breite“, „Feste Höhe“, „In %“) und der Zoom auf das Bild angewendet.

Mit der Methode „SaveAs“ des EditControls erfolgt schlußendlich der Speichervorgang.

Im Beispiel wird die „SaveAs“-Methode zweimal hintereinander verwendet. Dabei mache ich mir die Möglichkeit zunutze, der Methode Parameter zu übergeben. Unter anderem läßt sich bei „SaveAs“ auch vorgeben, ob ein Bild mit dem aktuellen Zoom oder dem Original-Zoom gespeichert werden kann.

Damit ist das Bild auf Festplatte gespeichert. Aber nun soll es noch in die Datenbank übernommen werden.

Dazu wird das Formular „frm100Bildpfade“ im „Add“-Modus geöffnet und die soeben gespeicherten Werte nach dort übergeben.