

Accessoires

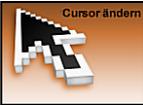
Kleine Helferlein für die Datenbank



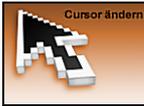
Es gibt in Access oft immer wieder die gleichen Probleme, gerade bei der Organisation größerer Datenbanken. Daher stelle ich hier verschiedene Lösungen vor, die mir (und einigen Kollegen) dabei geholfen haben.

Dabei ist mir wichtig, möglichst nur Access-Standard-Funktionalität zu benutzen und keine ActiveX-Controls oder ähnliche, die beim Kunden installiert werden müssen. Wenn es sich nicht vermeiden lässt, kommen einige wenige API-Funktionen zum Einsatz, die aber ebenfalls zu den Windows-Bordmitteln gehören.

Lorenz Hölscher

	Cursor ändern	2
	DropDown-Menüs in Formularen.....	2
	Geteilte Formulare.....	5
	Corporate Design	8
	Endlosformulare nebeneinander.....	11
	Zentrale Fortschrittsanzeige	15
	Ein- und Ausklappen	19
	Fremde Berichte einbinden	25

Der jeweilige Quellcode sowie die Beispiel-Datenbanken können auf den AEK-Download-Seiten heruntergeladen werden. Die Daten selber entstammen der Kunden-Tabelle der Nordwind-Datenbank und sind inhaltlich eher unwichtig.



Cursor ändern

Access kennt keine Eigenschaft, um den Maus-Cursor für ein Control zu ändern. Es ist aber mit ziemlich geringem Aufwand möglich, jedem aktiven Control einen eigenen Mauscursor unterzujubeln. Vor allem ist es für einige der anderen Beispiele hilfreich.

Zuerst braucht es eine passende Datei, denn Cursor haben ein eigenes Dateiformat. Das ist unproblematisch, denn abgesehen von einem breiten Angebot im Internet liegen auch auf der Festplatte (im Verzeichnis `..\\Windows\\Cursors\\`) viele geeignete Dateien.

Nun muss nur noch das Bild aus der Datei an den Mauszeiger gelangen. Dazu gibt es die API-Funktionen `LoadCursorFromFileA()` zum Laden und `SetCursor()` zum Zuweisen. Diese müssen am Anfang eines Standard-Moduls wie folgt deklariert werden:

```
Declare Function LoadCursorFromFileA Lib "user32" ( _
    ByVal lpFileName As String) As Long
Declare Function SetCursor Lib "user32" (ByVal hCursor As Long) As Long
```

Dann braucht es noch eine Prozedur `SetzeCursor()`, um den Cursor tatsächlich zu ändern. Sowohl die normalen Cursor in `*.cur`-Dateien als auch die animierten Cursor der `*.ani`-Dateien sind einsetzbar:

```
Sub SetzeCursor(strWelcher As String)
    Dim lngCursor As Long

    lngCursor = LoadCursorFromFileA(CurrentProject.Path & "\\ " & strWelcher)
    If lngCursor = 0 Then
        Exit Sub
    Else
        Call SetCursor(lngCursor)
    End If
End Sub
```

Mit dem Laden des Cursors muss sich der Code in `lngCursor` eine Long-Zahl für die Speicher-Adresse merken, an welche das Bild nun geladen wurde. Falls dieser Wert 0 war, ist das Laden aus irgendeinem Grund gescheitert, andernfalls wird mit der `SetCursor()`-Funktion dieses Bild dem Mauszeiger zugewiesen. Jetzt kann die Prozedur in einem beliebigen `MouseMove`-Ereignis der Controls aufgerufen werden:

```
Private Sub Kontaktperson_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    SetzeCursor "stopwtch.ani"
End Sub

Private Sub Kunden_Code_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    SetzeCursor "hand.cur"
End Sub
```

Sobald die Maus damit über eine der Textboxen bewegt wird, erhält sie in diesem Fall die Form der animierten oder statischen Cursor. Beim Verlassen wird automatisch wieder der normale Cursor angezeigt.



Access-typisch wird bei der Kombination Label/Textbox auch über dem Label der veränderte Cursor zu sehen sein. Um das zu vermeiden, müsste man die beiden trennen.



DropDown-Menüs in Form laren

Auf vielen Formularen sammelt sich schnell eine Vielzahl von Funktionen an, die zu ebenso vielen Buttons führen. Das folgende Beispiel mit 18(!) Buttons ist typisch dafür:



Natürlich könnten diese oft durch reguläre Menüs ersetzt werden, aber eben nicht auf Formularen im Dialog-Modus. Der Einsatz von PopUp-Menüs, die nur auf Rechtsklick geöffnet werden, ist aber vielen Benutzern nicht geläufig und widerspricht außerdem den Regeln einer intuitiven Oberfläche ("alle Bedienungselemente müssen erkennbar sein").

Die aus anderen Windows-Anwendungen bekannten DropDown-Menüs (also ein PopUp-Menü an einem Button) würden alle Wünsche erfüllen, nämlich die Sichtbarkeit durch einen Button und die Kompaktheit durch ein dort ausklappendes Menü.

Koordinaten für PopUp-Menü ermitteln

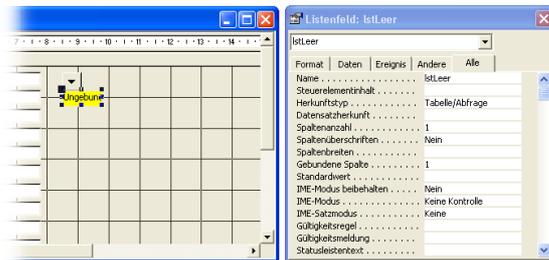
Leider passen die Koordinatensysteme von Buttons und PopUp-Menüs nicht zusammen:

- die *Left*- und *Top*-Eigenschaften des Buttons beziehen sich nur auf das Formular und rechnen in Twips (1/567 cm), aber
- die *x*- und *y*-Parameter der `ShowPopup()`-Methode beziehen sich auf den ganzen Bildschirm und rechnen in Pixeln

Immerhin stellt Windows eine API-Funktion `GetWindowRect()` bereit, welche die Bildschirm-Pixel-Koordinaten eines Objektes ermittelt. Allerdings braucht sie dazu die eindeutige Kennung des Objekts, das so genannte *windows handle* (`hWnd`).

Access stellt diese Eigenschaft für Buttons jedoch nicht zur Verfügung, sondern nur für Listboxen. Die Lösung besteht also darin, eine nicht sichtbare Listbox direkt neben den Button zu platzieren und deren Koordinaten zu nutzen.

Das Beispiel-Formular mit beliebigen Daten, einem Button `btnPopUp` und einer (hier gelb markierten) Listbox `lstLeer` sieht so aus:



Außerdem stellt der folgende Code in einem Standard-Modul `modDropDownButton` die Prozedur `ButtonKlick()` bereit, mit deren Hilfe das jeweilige PopUp-Menü im Formular aufgerufen wird:

```
Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Declare Function GetWindowRect Lib "user32" ( ByVal hWnd As Long, lpRect As RECT) As Long
Declare Function GetFocus Lib "user32" () As Long

Sub ButtonKlick(frmDieses As Form, lstDiese As ListBox)
    Dim rctKoordinaten As RECT
    Dim lngErfolg As Long
    Dim lngHandle As Long

    lstDiese.Enabled = True
    lstDiese.SetFocus
    lngHandle = GetFocus()
    lngErfolg = GetWindowRect(lngHandle, rctKoordinaten)

    frmDieses.btnPopUp.SetFocus
    lstDiese.Enabled = False
    With rctKoordinaten
        CommandBars("MeinPopUp").ShowPopup .Left, .Top
    End With
End Sub
```

Die `Type`-Deklaration ist notwendig, weil die `GetWindowRect()`-Funktion als zweiten Wert einen Rückgabeparameter einsetzt, der die vier Koordinaten in einer solchen Struktur liefert.

Damit die `ButtonKlick()`-Prozedur für verschiedene Formulare einsetzbar ist, werden sowohl das Formular als auch die `Listbox` als Parameter übergeben. Der Name des bereits vorhandenen `PopUp-Menüs` ist hingegen fest vorgegeben, weil es im Beispiel jeweils dynamisch erzeugt wird. Jede Fehlerbehandlung ist wegen des dann übersichtlicheren Codes weggelassen.

PopUp-Menü zur Laufzeit erzeugen

Grundsätzlich lässt sich natürlich für jedes Formular ein eigenes `PopUp-Menü` vorbereiten, da dessen Pflege jedoch eher lästig ist, bevorzuge ich dessen Erzeugung per Code. Dadurch kann es auch jederzeit dynamisch erzeugt und an seinen Anlass angepasst werden. Die zugehörige Prozedur `ErzeugeDiesesPopUpMenue` ist im Modul `modMenue` zu finden.

Die `CopyFace`- und `PasteFace`-Befehle dienen lediglich der optischen Gestaltung des Menüs mit Symbolen, die erste Zeile ist eine Art Überschrift und daher deaktiviert. Die beiden Versionen des `PopUp-Menüs` sehen nun wie im folgenden Bild aus, wobei die Positionierung zum `Button` vom Formular-Entwurf gesteuert wird:



Alle von einem Menü aufgerufenen Prozeduren (die in der `OnAction`-Eigenschaft genannt werden) müssen immer öffentlich und können also nicht Teil einer Klasse (wie eines Formular-Moduls!) sein. Daher stehen sie in diesem Standard-Modul, sogar dann, wenn sie sich wie bei `FirmaAnzeigen()` oder `TelefonMarkieren()` auf lokale Objekte im jeweils aktuellen Formular beziehen.

PopUp-Menü im Formular aufrufen

Innerhalb eines Formulars erfolgt der Aufruf so:

```
Private Sub btnPopUp_Click()
    ButtonKlick Me, Me.lstLeer
End Sub

Private Sub Form_Load()
    ErzeugeDiesesPopUpMenue True
    With Me.lstLeer
        .BackColor = Me.Detailbereich.BackColor
        .BorderColor = Me.Detailbereich.BackColor
        .Enabled = False
    End With
End Sub
```

In der `Form_Load()`-Prozedur wird die `Listbox` nur nachträglich farblich versteckt, damit sie während des Entwurfs noch vernünftig zu sehen ist. Ein Verstecken mittels `.Visible = False` ist nicht möglich, weil die `GetWindowsRect()`-Funktion nur auf sichtbare Objekte zugreifen kann.

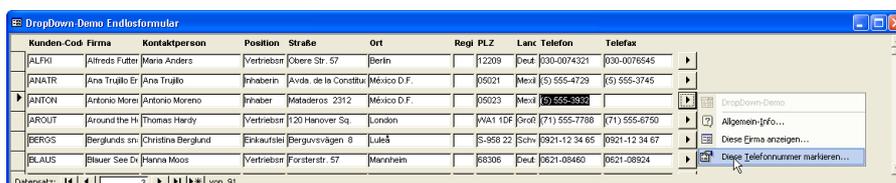
Das `BeimKlicken`-Ereignis des Buttons `btnPopUp` reicht das Formular und die `Listbox` an die zentrale Prozedur weiter und braucht ansonsten keine weitere Programmierung.

Beim Laden des Formulars wird das jeweils passende `PopUp-Menü` immer neu erzeugt. Wenn mehrere solcher Formulare gleichzeitig geöffnet sind, ist das ungeeignet, weil ein schlichter Fensterwechsel nicht das Menü aktualisieren würde. Dann muss die `ErzeugeDiesesPopUpMenue()`-Prozedur immer direkt in `btnPopUp_Click()` aufgerufen werden, was allerdings deutlich länger dauert.

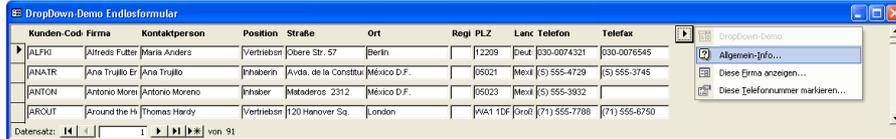
Der Aufruf ist sowohl für Einzel-Formulare geeignet...



... als auch für Endlos-Formulare ...



... und schließlich für Endlosformulare im Formulkopf:

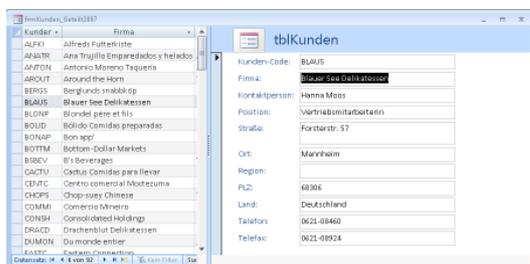


Das Dreieck auf der Schaltfläche ist übrigens ganz banal der Buchstabe 4 bzw. 6 in der Schrift *Marlett*.

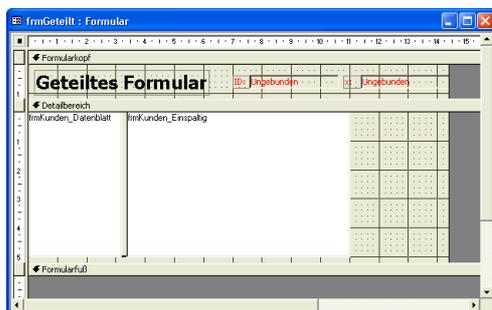


Geteilte Formulare

Eine der praktischen Neuerungen in Access 2007 sind die geteilten Formulare. Dabei handelt es sich um eine Mischung aus einem seitlichen Datenblatt und einem einspaltigen Formular wie im folgenden Bild:

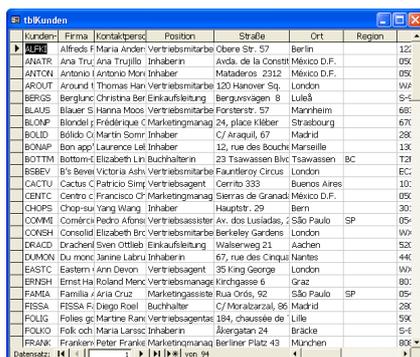


Das wäre natürlich auch praktisch in den älteren Versionen von Access und es ist gar nicht so schwierig, das dort zu verwirklichen. Es beginnt mit einem Formular ohne Datenbindung wie im folgenden Bild:



Darauf ist links ein SubForm-Control *sfmListe* für die Liste, rechts ein weiteres namens *sfmDaten* für die einspaltigen Daten und dazwischen ein unbeschrifteter schmaler Button *btnTeilung* als Trenner. Zusätzlich braucht es noch (hier im Formulkopf zu sehen) eine Textbox *edtID*. Das Konzept besteht darin, mit Auswahl in der Liste über das `Form_Current`-Ereignis die ID des Datensatzes in *edtID* zu schreiben und *sfmDaten* damit zu synchronisieren.

Die beiden Formulare, die in den SubForm-Controls angezeigt werden, enthalten die Datenquelle einmal mit der Standardansicht *Datenblatt* (links) und sichtbaren Navigationselementen sowie mit der Standardansicht *Einzel-Formular* (rechts) und ohne Navigationselemente.



Der Code in diesen eingebundenen Formularen ist minimal. Er beschränkt sich beim linken (Listen-)Formular darauf, die ID im Hauptformular in *edtID* zu schreiben:

```
Private Sub Form_Current()
    Me.Parent.Form.edtID.Value = Me.Kunden_Code.Value
End Sub
```

Im Code des rechten (Daten-)Formulars muss die Liste aktualisiert werden, sobald ein Datensatz hinzugefügt oder gelöscht wurde. Da die Erzeugung eines neuen Datensatzes nur im *Form_Current*-Ereignis abgefragt, aber erst im *Form_AfterUpdate* aktualisiert werden kann, wird die Eigenschaft über eine Modul-öffentliche Variable weitergereicht:

```
Dim m_booNeu As Boolean

Private Sub Form_AfterUpdate()
    If m_booNeu Then
        Me.Parent.Form.sfmListe.Requery
    End If
End Sub

Private Sub Form_Current()
    m_booNeu = Me.NewRecord
End Sub

Private Sub Form_AfterDelConfirm(Status As Integer)
    Me.Parent.Form.sfmListe.Requery
End Sub
```

Klasse für geteilte Formulare erstellen

Der wesentliche Code steht im Hauptformular und dient der Größenänderung. Damit das für mehrere Formulare nutzbar ist, gibt es eine Klasse *clsGeteilt*, deren Prozedurnamen auf den Einsatzzweck hinweisen und die originalen Parameter übernommen haben:

```
Dim m_frmDieses As Form
Dim m_sngX As Single

Property Set DiesFormular(frmDies As Form)
    Set m_frmDieses = frmDies
End Property
```

Die Modul-öffentliche Variable *m_frmDieses* enthält die Referenz auf das jeweilige Hauptformular, damit im Code alle seine Controls direkt angesprochen werden können. Das setzt natürlich voraus, dass die Control-Namen beibehalten wurden.

Das *MouseDown*-Ereignis wird nur benötigt, falls schon beim Ziehen mit der Maus die Darstellung aktualisiert werden soll. Da das aber ziemlich ruckelig ist, wird es hier zu Demozwecken nur mit gleichzeitig gedrückter Shift-Taste ausgeführt:

```
Sub btnTeilung_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Shift <> acShiftMask Then
        m_frmDieses.sfmListe.Visible = False
        m_frmDieses.sfmDaten.Visible = False
    End If
End Sub
```

Im *MouseMove*-Ereignis wird zuerst in der Modul-öffentlichen Variable *m_sngX* die geplante neue X-Koordinate errechnet. Wäre sie dichter als 1.500 Pixel (Konstante *c_lngAbstand*) am linken oder rechten Rand, wird stattdessen dieser Mindestabstand gewählt.

Auch hier wird die Aktualisierung der SubForm-Controls von der Shift-Taste abhängig gemacht, die ansonsten vorübergehend unsichtbar sind.

```
Sub btnTeilung_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Const c_lngAbstand = 1500

    SetzeCursor "lwe.cur" 'Achtung: SetzeCursor-Prozedur aus anderem Beispiel!
    If Button = acLeftButton Then
        m_sngX = X + m_frmDieses.btnTeilung.Left
        If m_sngX < c_lngAbstand Then m_sngX = c_lngAbstand
        If m_sngX > (m_frmDieses.InsideWidth - c_lngAbstand) Then
            m_sngX = m_frmDieses.InsideWidth - c_lngAbstand
        End If
        m_frmDieses.btnTeilung.Left = m_sngX
        m_frmDieses.edtX.Value = m_sngX 'nur zu Demo-Zwecken, kann wegfallen
        If Shift = acShiftMask Then
            m_frmDieses.sfmListe.Width = m_sngX
            m_frmDieses.sfmDaten.Left = m_sngX + m_frmDieses.btnTeilung.Width
            m_frmDieses.sfmDaten.Width = m_frmDieses.InsideWidth - m_frmDieses.sfmDaten.Left
        End If
    End If
End Sub
```

Erst im *MouseUp*-Ereignis werden die endgültigen Werte für die beiden SubForm-Controls und den Button eingestellt. Dabei wird durch die *m_sngX*-Variable die letzte gültige Verschiebung übernommen:

```

Sub btnTeilung_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    With m_frmDieses.sfmListe
        .Width = m_sngX
        .Visible = True
    End With
    m_frmDieses.btnTeilung.Left = m_sngX
    With m_frmDieses.sfmDaten
        .Left = m_sngX + m_frmDieses.btnTeilung.Width
        .Width = m_frmDieses.InsideWidth - .Left
        .Visible = True
    End With
End Sub

```

Damit in jeder Ansicht die Höhen der drei Controls und die Breite von *sfmDaten* den ganzen verbleibenden Bildschirm nutzt, muss das im *Form_Resize*-Ereignis dynamisch berechnet werden. Dabei gibt es ein paar Unwägbarkeiten, nämlich die Formarränder und evtl. vorhandene Kopf-/Fußzeilen, die ebenfalls berücksichtigt werden müssen. Die Konstante *clngDifferenz* sorgt dafür, dass diese Abweichungen abgezogen und keine unnötigen Scrollbars angezeigt werden.

```

Sub Form_Resize()
    Dim lngHoehe As Long
    Const clngDifferenz = 100

    On Error Resume Next 'falls ohne Kopf/Fuß
    lngHoehe = m_frmDieses.InsideHeight - clngDifferenz
    lngHoehe = m_frmDieses.InsideHeight - m_frmDieses.Section(acHeader).Height - _
        m_frmDieses.Section(acFooter).Height - clngDifferenz * 3
    On Error GoTo 0
    m_frmDieses.Detailbereich.Height = lngHoehe
    m_frmDieses.sfmListe.Height = lngHoehe
    m_frmDieses.btnTeilung.Height = lngHoehe
    m_frmDieses.sfmDaten.Height = lngHoehe
    With m_frmDieses.sfmDaten.Form
        On Error Resume Next 'falls ohne Kopf/Fuß
        .Section(acDetail).Height = lngHoehe - clngDifferenz * 3
        .Section(acDetail).Height = lngHoehe - .Section(acHeader).Height - _
            .Section(acFooter).Height - clngDifferenz * 6
        On Error GoTo 0
    End With
    m_frmDieses.sfmDaten.Width = m_frmDieses.InsideWidth - _
        m_frmDieses.sfmDaten.Left
End Sub

```

Klasse im Formular aufrufen

Dieser gesamte Code ist in der Klasse *clsGeteilt* gekapselt und muss im Haupt-Formular nur noch wie folgt aufgerufen werden:

```

Dim m_clsTeil As clsGeteilt

Private Sub btnTeilung_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    m_clsTeil.btnTeilung_MouseDown Button, Shift, X, Y
End Sub

Private Sub btnTeilung_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    m_clsTeil.btnTeilung_MouseMove Button, Shift, X, Y
End Sub

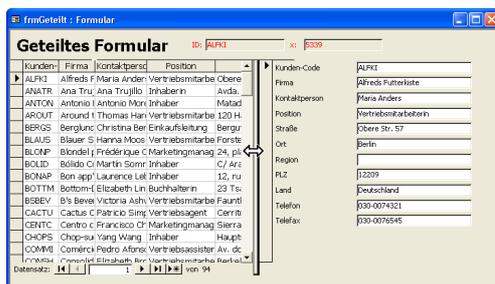
Private Sub btnTeilung_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
    m_clsTeil.btnTeilung_MouseUp Button, Shift, X, Y
End Sub

Private Sub Form_Load()
    Set m_clsTeil = New clsGeteilt
    Set m_clsTeil.DiesFormular = Me
End Sub

Private Sub Form_Resize()
    m_clsTeil.Form_Resize
End Sub

```

Da die klasseninternen Prozeduren in Namen und Parametern mit dem jeweiligen Aufrufer identisch sind, ist sofort zu sehen, welche zusammengehören. Das Haupt-Formular ist nun mitsamt seiner Teilung fertig:





Corporate Design

Kaum ist die Datenbank fertig, beschließt der Kunde, dass er sein Logo doch gerne links oben sehen würde und die Überschrift auf allen Formularen etwas kleiner werden soll. Das ist bei kleinen Datenbanken lästig, aber bei solchen mit vielen Formularen unglaublich zeitaufwändig.

Das AutoFormat ist dabei keine Lösung, denn es kann

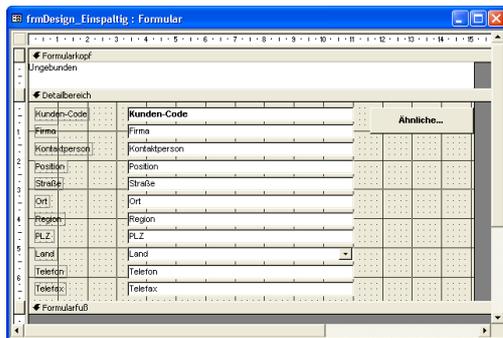
- keine Positionen verändern,
- keine neuen Elemente hinzufügen,
- nur nach Control-Typ anpassen (also beispielsweise nicht Labels unterschiedlich behandeln) und
- nicht per VBA aufgerufen werden.

Anstatt also alle Formulare "nachbasteln" zu müssen, ist es viel effektiver, direkt intelligente Formulare zu erstellen. Bei diesen stehen die allgemeinen Elemente des Kopfbereichs (also Titel, Untertitel und Logo) nicht auf jedem Formular, sondern in einem eingebundenen Unterformular.

Unterformular für Kopf-/Fußzeile in alle Formulare integrieren

Das Hauptformular lädt das Unterformular und übergibt ihm die anzuzeigenden Titeltexthe, während das Unterformular sich um das Design kümmert. Dabei darf das Kopfzeilen-Unterformular bei Bedarf auch Teile des Hauptformulars zur Laufzeit ändern, etwa die Hintergrundfarbe oder das Hintergrundbild.

Der Entwurf des "normalen" Hauptformulars sieht eigentlich aus wie immer, lediglich im Kopfbereich befindet sich ein Unterformular:



Nur durch Auswechseln des Unterformulars für die Kopfzeile lassen sich mit diesem gleichen Entwurf diese verschiedenen Laufzeit-Anzeigen erzeugen:



Dazu braucht es ein wenig zentralen Code in einem Standard-Modul, wobei die Konstanten hier nur dem einfachen Wechsel zwischen verschiedenen beispielhaften Kopfzeilen-Unterformularen dienen:

```
Public p_strTitelHaupt As String
Public p_strTitelUnter As String

'Const cstrDesignName = "Buttons"
'Const cstrDesignName = "Schluessel"
'Const cstrDesignName = "AEK12"
Const cstrDesignName = "Einfach"

Sub LadeFormularKopf(frmDieses As Form, strTitelHaupt As String, _
    strTitelUnter As String)

    Dim ctlEinzel As Control
    Dim lngDifferenz As Long 'falls noch andere Controls drauf sind

    p_strTitelHaupt = strTitelHaupt
    p_strTitelUnter = strTitelUnter
    With frmDieses
        .sfmKopf.SourceObject = "sfmKopf_" & cstrDesignName
        .Picture = .sfmKopf.Form.Picture
        .PictureSizeMode = .sfmKopf.Form.PictureSizeMode
        .Section(acHeader).BackColor = .sfmKopf.Form.Section(acDetail).BackColor
        .Section(acDetail).BackColor = .sfmKopf.Form.Section(acDetail).BackColor
        .Section(acFooter).BackColor = .sfmKopf.Form.Section(acDetail).BackColor
    End With
End Sub
```

```

lngDifferenz = (.sfmKopf.Form.Section(acDetail).Height - .sfmKopf.Height)
.Section(acHeader).Height = .Section(acHeader).Height + lngDifferenz
.sfmKopf.Height = .sfmKopf.Height + lngDifferenz
For Each ctlEinzel In .Section(acHeader).Controls
    With ctlEinzel
        If LCase(.Name) <> "sfmkopf" Then
            .Top = .Top + lngDifferenz - 7
        End If
    End With
Next
End With
End Sub

```

Das Kopfzeilen-Unterformular speichert die beiden Titeltexte in Datei-öffentliche Variablen und passt anschließend sein übergeordnetes Elternformular (also das Hauptformular) an. Dabei wird dessen Hintergrundfarbe für Kopf-, Detail- und Fußbereich an diejenige des Unterformulars angepasst und auch das Hintergrundbild übernommen.

Da gerade tabellarische Endlos-Formulare typischerweise die Überschriften im Kopfbereich des Hauptformulars enthalten, müssen diese bei einer Größenänderung des eingebetteten Unterformulars entsprechend verschoben werden. Das erledigt die `For Each/Next`-Schleife mit der Variablen `lngDifferenz`, wobei hier offenbar ein kleiner Abstand (7 Pixel) zum rechnerisch ermittelten Rand notwendig ist.

Der Code im Hauptformular beschränkt sich auf den Aufruf dieser Prozedur:

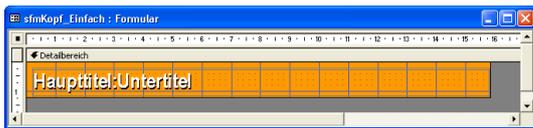
```

Private Sub Form_Load()
    LadeFormularKopf Me, "Kunden-Daten", "Daten aus der Nordwind-DB"
End Sub

```

Code im Unterformular

Das Kopfzeilen-Unterformular hingegen kann sehr einfachen Code enthalten, wenn darin wenig zu organisieren ist, wie für *sfmKopf_Einfach*:



Da hier nur die Texte aus dem Datei-öffentlichen Variablen ausgelesen und die Breite angepasst werden muss, ist der Code sehr kurz:

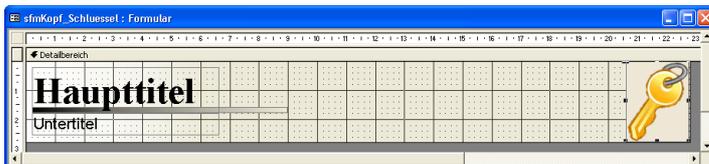
```

Private Sub Form_Load()
    Me.lblTitel.Caption = p_strTitelHaupt & " : " & p_strTitelUnter
    Me.lblTitelSchatten.Caption = Me.lblTitel.Caption
End Sub

Private Sub Form_Resize()
    Me.lblTitel.Width = Me.InsideWidth - Me.lblTitel.Left - 10
    Me.lblTitelSchatten.Width = Me.lblTitel.Width
End Sub

```

Für das Beispiel *sfmKopf_Schluesssel* ist geringfügig mehr Rechenaufwand nötig, damit das Schlüssel-Bild abhängig von der Formulargröße immer am rechten Rand positioniert und die übrigen Breiten angepasst werden:



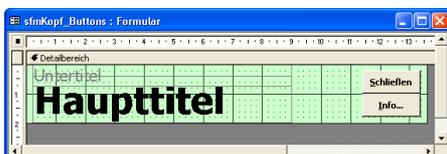
```

Private Sub Form_Load()
    Me.lblTitelHaupt.Caption = p_strTitelHaupt
    Me.lblTitelUnter.Caption = p_strTitelUnter
End Sub

Private Sub Form_Resize()
    Me.imgSchluesssel.Left = Me.InsideWidth - Me.imgSchluesssel.Width - 10
    Me.imgLinie.Width = Me.imgSchluesssel.Left - Me.imgLinie.Left - 10
    Me.lblTitelHaupt.Width = Me.imgSchluesssel.Left - Me.lblTitelHaupt.Left - 10
    Me.lblTitelUnter.Width = Me.imgSchluesssel.Left - Me.lblTitelUnter.Left - 10
End Sub

```

Auch Schaltflächen im Formularkopf sind möglich, wie *sfmKopf_Buttons* zeigt:



Dessen Code kann beispielhaft auf das Elternformular zugreifen:

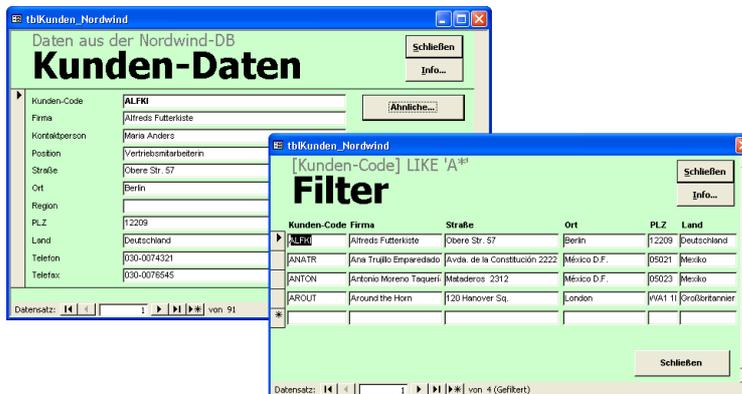
```
Private Sub btnInfo_Click()
    MsgBox "Ich bin in Formular '" & Me.Parent.Name & "' integriert.", vbInformation
End Sub

Private Sub btnSchliessen_Click()
    DoCmd.Close acForm, Me.Parent.Name
End Sub

Private Sub Form_Load()
    Me.lblTitelHaupt.Caption = p_strTitelHaupt
    Me.lblTitelUnter.Caption = p_strTitelUnter
End Sub

Private Sub Form_Resize()
    Me.btnSchliessen.Left = Me.InsideWidth - Me.btnSchliessen.Width - 10
    Me.btnInfo.Left = Me.btnSchliessen.Left
    Me.lblTitelHaupt.Width = Me.btnSchliessen.Left - Me.lblTitelHaupt.Left - 10
    Me.lblTitelUnter.Width = Me.btnSchliessen.Left - Me.lblTitelUnter.Left - 10
End Sub
```

Selbstverständlich funktioniert das auch für mehrere, gleichzeitig geöffnete Formulare ebenso wie für PopUp-Fenster (im folgenden Bild vom Button *[Ähnliche...]* ausgelöst):



Corporate Design auch für Berichte

Nur für Berichte funktioniert es nicht ganz so locker. Es ist hinzukriegen, hat aber einige Einschränkungen:

- eine Größenänderung des Unterberichts beim Laden ist nicht mehr möglich, diese muss schon zu dessen Entwurfszeit stimmen, und
- Eigenschaften wie die Hintergrundfarbe des Unterberichts sind nur außerhalb des eingebetteten Berichts auszulesen (weil sonst zu spät)

Eine Prozedur `LadeBerichtsKopf()` im Standard-Modul entspricht vom Konzept her ansonsten weitgehend der oben vorgestellten `LadeFormularKopf()`-Prozedur:

```
Sub LadeBerichtsKopf(rptDieser As Report, strTitelHaupt As String, strTitelUnter As String)
    Dim strPicture As String
    Dim intPictureSizeMode As Integer
    Dim lngHoehe As Long
    Dim lngBackColor As Long

    p_strTitelHaupt = strTitelHaupt
    p_strTitelUnter = strTitelUnter

    DoCmd.Echo False
    DoCmd.OpenReport "rptKopf_" & cstrDesignName, acViewDesign
    With Reports("rptKopf_" & cstrDesignName)
        lngHoehe = .Section(acDetail).Height
        lngBackColor = .Section(acDetail).BackColor
        strPicture = .Picture
        intPictureSizeMode = .PictureSizeMode
    End With
    DoCmd.Close acReport, "rptKopf_" & cstrDesignName
    DoCmd.Echo True

    With rptDieser.Report
        .srpKopf.SourceObject = "rptKopf_" & cstrDesignName
        .srpKopf.Height = lngHoehe
        .Picture = CurrentProject.Path & "\" & NurDatei(strPicture)
        .PictureSizeMode = intPictureSizeMode
        .Section(acDetail).BackColor = lngBackColor
    End With
End Sub
```

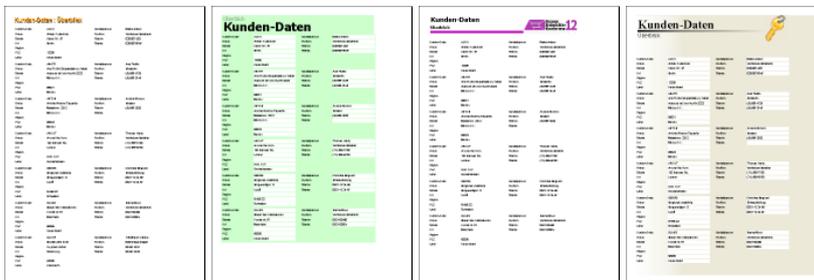
Der wesentliche Unterschied besteht darin, dass der Unterbericht zuerst vorher einzeln geöffnet werden muss, um seine Eigenschaften auslesen zu können, da er während des `Report_Open`-Ereignisses im Hauptbericht noch nicht geöffnet und also nicht zugreifbar ist.

Außerdem unterscheidet sich die `Picture`-Eigenschaft offenbar von derjenigen des Formulars, was aber nur wichtig ist, wenn der darin gespeicherte Pfad zwischenzeitlich geändert wurde. Dann sucht der Bericht nach dem dort genannten Pfad/Dateinamen, um das Bild nachzuladen. Um den dort notierten Pfad zu umgehen, wird mit der folgenden `NurDatei()`-Funktion der reine Dateiname extrahiert und diese im Verzeichnis der Datenbank gesucht:

```
Function NurDatei(strPfadMitDatei As String) As String
    Dim intPos As Integer

    intPos = InStrRev(strPfadMitDatei, "\")
    If intPos = 0 Then 'also ohne Pfad
        If Mid(strPfadMitDatei, 2, 1) = ":" Then
            'Achtung, falls Root-Verzeichnis
            NurDatei = Mid(strPfadMitDatei, 3)
        Else
            NurDatei = strPfadMitDatei
        End If
    Else
        NurDatei = Mid(strPfadMitDatei, intPos + 1)
    End If
End Function
```

Mit diesen wenigen Einschränkungen lassen sich dann jedoch ebenso verschiedene Berichtsdesigns verwirklichen, wie es für die Formulare möglich war



So lassen sich auch umfangreiche Datenbanken mit einheitlichen Kopf- und Fußzeilen ausstatten, ohne für jede Änderung aufwändig nacharbeiten zu müssen.



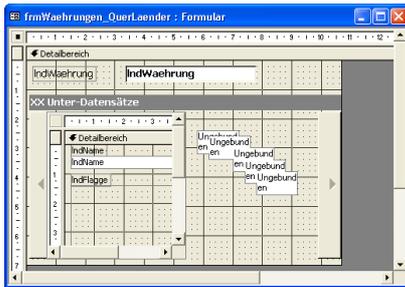
Endlosformulare nebeneinander

Endlosformulare sind so lange praktisch, wie sich die Daten in geringer Höhe und beliebiger Breite darstellen lassen. Für hochformatige Fotos von Personen oder Listboxen wird jedoch daneben sehr viel Platz verschwendet. Da wäre es besser, die Daten nebeneinander anzeigen zu können.

Da Access das nicht anbietet, lässt es sich mit Hilfe von SubForms selber programmieren. Als Vorbild dient dabei die Anzeige in *amazon.de* für Bücher, die zum ausgewählten Buch von anderen Kunden gekauft wurden:



Entsprechend erhält der Formular-Entwurf zwei Buttons `btnQuerRechts` und `btnQuerLinks` zum Einblenden der nicht sichtbaren Daten, mehrere SubForm-Elemente `sfmQuer00` (hier bis `sfmQuer05`) und ein Label `lblAnzahl`:



Das erste SubForm-Element *sfmQuer00* besitzt bereits zur Entwurfszeit die benötigten Eigenschaften Herkunftsobjekt, Breite und Höhe. Alle übrigen SubForms übernehmen dies erst zur Laufzeit, was die Pflege vereinfacht.

Der wesentliche Code wird in eine Klasse *clsQuer* ausgelagert. Diese sorgt dafür, dass die SubForms ein- und ausgeblendet, die passenden Daten ermittelt und die Buttons de-/aktiviert werden. Dazu benötigt sie einige Informationen vom aufrufenden Formular, die in Modul-öffentlichen Variablen gespeichert werden:

```

Dim m_rcsUnter As DAO.Recordset           'Datenquelle
Dim m_frmHaupt As Form                    'Hauptformular
Dim m_arrUnterIDs() As String             'alle IDs für die QuerFormulare
Dim m_intSichtbar As Integer              'wie viele QuerFormulare sichtbar sind
Dim m_intLinks As Integer                 'welcher Index am linken Rand steht
Dim m_intRechts As Integer                'welcher Index am rechten Rand steht
Dim m_intMaxEntwurf As Integer            'wie viele QuerFormulare im Entwurf vorhanden sind
Dim m_intMaxDaten As Integer              'wie viele Daten gefunden wurden
Dim m_txtFocus As TextBox                 'Textbox erhält Focus, wenn QuerFormulare deaktiviert sind
Dim m_strFeldSync As String               'das Synchronisationsfeld mit dem Hauptformular
Dim m_strFeldID As String                 'das ID-Feld im Querformular
Dim m_strRecordsetName As String          'der Recordset-Name für die QuerFormulare

Const m_cintAbstandButton = 150         'Abstand zwischen Buttons und QuerFormularen

```

Die in `Form_Load()` aufzurufende und hier ebenfalls `Form_Load()` genannte Prozedur organisiert die Starteinstellungen wie die Zuweisung an die Formular-Variable oder das Öffnen des Recordsets:

```

Sub Form_Load(frmHauptformular As Form)
    Dim sfmObject As SubForm
    Dim intI As Integer

    'alle Standardwerte vorbereiten
    Set m_frmHaupt = frmHauptformular
    m_intLinks = 0
    Set m_rcsUnter = CurrentDb.OpenRecordset(m_strRecordsetName, dbOpenDynaset)

    Set sfmObject = m_frmHaupt.sfmQuer00
    For intI = 1 To m_intMaxEntwurf
        With sfmDiesQuer(intI)
            .SourceObject = sfmObject.SourceObject
            .Top = sfmObject.Top
            .Width = sfmObject.Width
            .Height = sfmObject.Height
        End With
    Next
End Sub

```

Die übrigen Parameter werden vor allem deswegen in einzelnen Property-Prozeduren zugewiesen, damit es hier übersichtlicher ist. Technisch hätte auch `Form_Load()` eine längere Parameter-Liste erhalten können:

```

Property Let FeldSync(strFeldname As String)
    m_strFeldSync = strFeldname
End Property

Property Let FeldID(strFeldname As String)
    m_strFeldID = strFeldname
End Property

Property Let RecordsetName(strRSName As String)
    m_strRecordsetName = strRSName
End Property

Property Let AnzahlQuer(intAnzahl As Integer)
    m_intMaxEntwurf = intAnzahl - 1
End Property

Property Set TextboxFocus(txtTextbox As TextBox)
    Set m_txtFocus = txtTextbox
End Property

```

Die Prozedur `QuerDatenErmitteln()` findet zum aktuellen Datensatz alle Detaildatensätze und schreibt deren eindeutigen Schlüssel in das Array. Der Recordset dazu bleibt immer geöffnet, weil das erheblich schneller ist, als ihn für jeden Masterdatensatz erneut zu schließen und zu öffnen.

```

Sub QuerDatenErmitteln(strWertSync As String)
    Dim intI As Integer

    m_rcsUnter.FindFirst "[" & m_strFeldSync & "]" = " & strWertSync
    If m_rcsUnter.NoMatch Then
        m_frmHaupt.lblAnzahl.Caption = "Keine Datensätze"
        m_intMaxDaten = 0
        m_intRechts = -1
    Else
        ReDim m_arrUnterIDs(0)
        Do Until m_rcsUnter.NoMatch
            m_arrUnterIDs(intI) = m_rcsUnter.Fields(m_strFeldID).Value & ""
            intI = intI + 1
            ReDim Preserve m_arrUnterIDs(intI)
            m_rcsUnter.FindNext "[" & m_strFeldSync & "]" = " & strWertSync
        Loop
        m_intMaxDaten = intI
        If m_intMaxDaten = 1 Then
            m_frmHaupt.lblAnzahl.Caption = "1 Datensatz"
        Else
            m_frmHaupt.lblAnzahl.Caption = m_intMaxDaten & " Datensätze"
        End If
        m_intRechts = Min3(m_intMaxDaten - 1, m_intMaxEntwurf, m_intSichtbar)
    End If

    m_txtFocus.SetFocus
    m_frmHaupt.btnQuerLinks.Enabled = False
    ButtonCheckRechts
End Sub

```

Danach werden in `QuerFormulareAnzeigen()` die IDs den jeweiligen SubForms zugewiesen, indem deren `RecordSource`-Eigenschaft mit einem entsprechend gefilterten `SELECT` versehen wird:

```

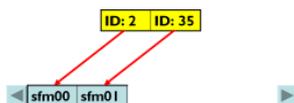
Sub QuerFormulareAnzeigen()
    Dim intI As Integer
    Dim IntQuer As Integer

    'alle unsichtbar machen
    m_txtFocus.SetFocus
    For IntQuer = 0 To m_intMaxEntwurf
        sfmDiesQuer(IntQuer).Visible = False
    Next

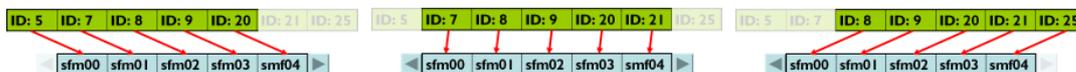
    IntQuer = 0
    For intI = m_intLinks To m_intRechts
        With sfmDiesQuer(IntQuer)
            .Visible = True
            .Form.RecordSource = "SELECT * FROM [" & m_strRecordsetName & "]" & _
                "WHERE [" & m_strFeldID & "] = '" & m_arrUnterIDs(intI) & "'"
        End With
        IntQuer = IntQuer + 1
    Next
End Sub

```

Wenn es weniger IDs als SubForms sind, werden diese jeweils zugeordnet und die übrigen SubForms ausgeblendet:



Sind jedoch mehr IDs vorhanden als SubForms, darf nur ein Teil der IDs auf die SubForms abgebildet werden. Die Modul-öffentlichen Variablen `m_intLinks` und `m_intRechts` geben dazu an, welche IDs aktuell sichtbar sind:



Die Klick-Ereignisse der Buttons sorgen im Wesentlichen dafür, dass `m_intLinks` und `m_intRechts` angepasst und die Buttons selber bei Bedarf deaktiviert werden. Da ohne Detaildatensätze alle SubForms und die Buttons deaktiviert sind, muss für diesen Fall die `m_txtFocus`-Textbox des Hauptdatensatzes zur Aktivierung vereinbart sein, weil es sonst Laufzeitfehler gibt:

```

Sub btnQuerLinks_Click()
    m_intLinks = m_intLinks - 1
    m_intRechts = m_intRechts - 1
    QuerFormulareAnzeigen
    m_frmHaupt.btnQuerRechts.Enabled = True
    m_txtFocus.SetFocus
    If m_intLinks = 0 Then
        m_frmHaupt.btnQuerLinks.Enabled = False
    End If
End Sub

```

```

Sub btnQuerRechts_Click()
    m_intLinks = m_intLinks + 1
    m_intRechts = m_intRechts + 1
    QuerFormulareAnzeigen
    m_frmHaupt.btnQuerLinks.Enabled = True
    ButtonCheckRechts
End Sub

Sub ButtonCheckRechts()      'Einzelaufruf wegen QuerDatenErmitteln()
    m_txtFocus.SetFocus
    m_frmHaupt.btnQuerRechts.Enabled = (m_intRechts < m_intMaxDaten - 1)
End Sub

```

Die beiden wichtigsten auslösenden Ereignisse sind `Form_Current()` und `Form_Resize()` zur Ermittlung der aktuellen Daten bzw. zur Ermittlung der überhaupt sichtbaren SubForms:

```

Sub Form_Current(strAuswahl As String)
    m_intLinks = 0
    QuerDatenErmitteln strAuswahl
    QuerFormulareAnzeigen
End Sub

```

Die `Form_Resize()`-Prozedur prüft, wie viele SubForms einschließlich Buttons in die vorgegebene Maximalbreite `InsideWidth` hineinpassen würden und speichert deren Anzahl in `m_intSichtbar`:

```

Sub Form_Resize(strAuswahl As String)
    Dim intI As Integer
    Dim lngGesamtBreit As Long
    Dim lngQuerBreit As Long

    lngGesamtBreit = m_frmHaupt.btnQuerLinks.Width + m_cintAbstandButton      'Anfangswert
    lngQuerBreit = m_frmHaupt.sfmQuer00.Width                                'alle gleich breit

    For intI = 0 To m_intMaxEntwurf
        With sfmDiesQuer(intI)
            If lngGesamtBreit <= (m_frmHaupt.InsideWidth - m_frmHaupt.btnQuerRechts.Width - _
                lngQuerBreit - m_cintAbstandButton) Then
                .Left = lngGesamtBreit
                .Visible = True
                lngGesamtBreit = lngGesamtBreit + lngQuerBreit
            Else
                Exit For
            End If
        End With
    Next

    m_intSichtbar = intI - 1
    m_frmHaupt.btnQuerRechts.Left = lngGesamtBreit + m_cintAbstandButton
    With m_frmHaupt
        .lblAnzahl.Width = .btnQuerRechts.Left + .btnQuerRechts.Width - .lblAnzahl.Left
    End With
    ButtonCheckRechts
    QuerDatenErmitteln strAuswahl
    QuerFormulareAnzeigen
End Sub

```

Damit das jeweilige SubForm-Objekt einfacher zu schreiben ist, gibt es die `sfmDiesQuer()`-Funktion:

```

Function sfmDiesQuer(intNr As Integer) As SubForm
    Set sfmDiesQuer = m_frmHaupt.Controls("sfmQuer" & Format(intNr, "00"))
End Function

```

In `QuerDatenErmitteln()` musste der kleinste Wert aus `m_intMaxDaten` (wie viele Detaildatensätze), `m_intMaxEntwurf` (wie viele QuerForms stehen im Entwurf zur Verfügung) und `m_intSichtbar` (wie viele davon passen auf das Formular) gefunden werden. Dazu steht diese spezielle `Min3()`-Funktion bereit.

```

Function Min3(Eins As Variant, Zwei As Variant, Drei As Variant) As Variant
    If Eins < Zwei Then
        Min3 = Eins
    Else
        Min3 = Zwei
    End If

    If Min3 > Drei Then
        Min3 = Drei
    End If
End Function

```

Je nach gefundenen Unterdatensätzen werden nun passende Endlosformulare nebeneinander angezeigt und mit Daten gefüllt. Sind zu viele Daten vorhanden, lassen sie sich mit den Buttons rechts und links verschieben. Im aufrufenden Formular werden die Ereignisse dann eigentlich nur an die Klasse weitergereicht:

```

Dim m_clsQuer As clsQuer

Private Sub btnQuerLinks_Click()
    m_clsQuer.btnQuerLinks_Click
End Sub

```

```

Private Sub btnQuerRechts_Click()
    m_clsQuer.btnQuerRechts_Click
End Sub

Private Sub Form_Current()
    m_clsQuer.Form_Current "" & Me.lndWaehrung.Value & ""
End Sub

Private Sub Form_Load()
    Set m_clsQuer = New clsQuer

    With m_clsQuer
        .RecordsetName = "tblLaender"
        .FeldSync = "lndWaehrung"
        .FeldID = "lndName"
        Set .TextboxFocus = Me.lndWaehrung
        .AnzahlQuer = 6
        .Form_Load Me
    End With
End Sub

Private Sub Form_Resize()
    m_clsQuer.Form_Resize "" & Me.lndWaehrung.Value & ""
End Sub

```

Damit ist die Anzeige der nebeneinander liegenden Unterformulare funktionsfähig:



Zentrale Fortschrittsanzeige

Gerade bei großen Datenmengen oder langwierigen Aktionen in Access ist es sinnvoll, Benutzer darüber zu informieren, wie weit die Arbeit vorangeschritten ist, am besten mit einem Fortschrittsbalken.

Access bringt bereits alles mit, um einen Fortschrittsbalken in der Statuszeile anzuzeigen, denn der blaue Fortschrittsbalken in der Statuszeile lässt sich mit der vielseitigen SysCmd()-Funktion erzeugen. Ihre tatsächliche Aufgabe erhält sie erst mit der jeweiligen Konstante des ersten Parameters.

Fortschrittsanzeige in der Statuszeile

Zum Testen braucht es in einer leeren Datenbank ein Formular *frmFortschrittStatuszeile* mit einem Button *btnStatuszeile* und einem (unsichtbaren) Label *lblGucken* wie im folgenden Bild:



Die *btnStatuszeile_Click*-Prozedur erhält den folgenden Code. Die Prozedur ist nur eine inhaltsleere Schleife und zeigt den Fortschrittsbalken in der Statuszeile an:

```

Private Sub btnStatuszeile_Click()
    Dim lngZaehler As Long
    Const clngMaximum = 5000000

    Me.lblGucken.Visible = True 'Hinweis sichtbar machen
    Me.Repaint
    SysCmd acSysCmdInitMeter, "Bitte warten...", clngMaximum

```

```

For lngZaehler = 1 To clngMaximum
    SysCmd acSysCmdUpdateMeter, lngZaehler
Next

SysCmd acSysCmdRemoveMeter
Me.lblGucken.Visible = False 'Hinweis wieder verstecken
End Sub

```

Die Statuszeile beginnt hier mit der Initialisierung, wofür der erste Parameter auf `acSysCmdInitMeter` gestellt werden muss. Der zweite Parameter nennt den Text, der anschließend unverändert immer vor dem Fortschrittsbalken angezeigt wird.

Im dritten Parameter (hier als Konstante `clngMaximum`) steht der Maximalwert für die 100%-Anzeige. Mit dem zweiten Aufruf von `SysCmd` und dem Parameter `acSysCmdUpdateMeter` wird der Fortschrittsbalken ständig aktualisiert. Der blaue Balken verlängert sich automatisch im Verhältnis von `lngZaehler` zu `clngMaximum`:



Das sieht zwar keiner, weil niemand die Augen zur Statuszeile ganz unten bewegt, wenn oben eine Schaltfläche gedrückt wurde (deswegen erscheint die "Jetzt schnell unten gucken!"-Anzeige), aber ansonsten ist das nett und unproblematisch.

Richtig gut ist es jedoch nicht, denn es lässt sich weder in die Bildschirmmitte rücken noch mit fortlaufend aktualisierten Texten versehen wie beispielsweise dem Fortschritt in Prozent-Werten oder dem Namen der gerade bearbeiteten Datei.

Wer versucht, den Text auch während der Schleife mittels `acSysCmdInitMeter` zu setzen, sieht den blauen Balken immer nur am Anfang stehen, es dauert mehr als zehn Mal so lang und flackert ganz furchtbar. Auch der Einsatz von `SysCmd acSysCmdSetStatus`, "dynamischer Text" in der Schleife führt bloß zu einem Laufzeitfehler, weil anschließend der Fortschrittsbalken nämlich ganz gelöscht (und durch diesen Text ersetzt) ist.

(Achtung: in Access2007 sollte man auf gar keinen Fall `DoCmd.Echo` einsetzen, um in einer solchen Schleife dynamische Texte anzuzeigen, denn das ist indiskutabel langsam: für bescheidene 10.000 Durchläufe schon volle 17 sec.!)

Der Vorteil dieser `SysCmd()`-Funktion besteht darin, dass sie schnell ist, überall eingesetzt werden kann und keine Vorbereitung braucht. Leider lässt sie sich kaum beeinflussen und viele Benutzer gucken überhaupt nicht in die Statuszeile.

Fortschrittsanzeige im Formular

Die alternative Technik für einen Fortschrittsbalken verlagert diesen in ein Formular. Das ist fast automatisch im Blickfeld des Benutzers und bietet vor allem völlige Gestaltungsfreiheit. Die speziellen Elemente für ein solches Formular sehen aus wie im folgenden Formular-Entwurf:



Zum Button `btnFormular` kommen noch ein blaues Rechteck `rctInnen` und ein durchsichtiges Rechteck `rctAussen` mit vertieftem Rand sowie ein durchsichtiges Label `lblProzent`. In der Schleife wird `rctInnen` dann von `Width = 0` bis zu `rctAussen.Width` vergrößert und `inlblProzent.Caption` der aktuelle Wert angezeigt:

```

Private Sub btnFormular_Click()
    Dim lngZaehler As Long
    Const clngMaximum = 5000

    Me.rctAussen.Visible = True
    Me.rctInnen.Visible = True
    Me.lblProzent.Visible = True

    For lngZaehler = 1 To clngMaximum
        Me.rctInnen.Width = Me.rctAussen.Width * lngZaehler / clngMaximum
        Me.rctInnen.BackColor = CLng(32000 * lngZaehler / clngMaximum)
        With Me.lblProzent
            .Caption = Format(lngZaehler / clngMaximum, "0.0 %")
            .ForeColor = IIf(lngZaehler / clngMaximum > 0.5, vbWhite, vbBlack)
        End With
        Me.Repaint
    Next

    Me.rctAussen.Visible = False
    Me.rctInnen.Visible = False
    Me.lblProzent.Visible = False

```

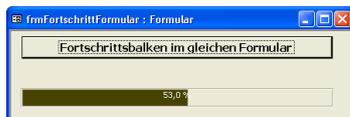
```
End Sub
```

Die tatsächliche Breite von *rcInnen* selber zu berechnen, ist offensichtlich keine große Herausforderung. Viel wichtiger ist der *Me.Repaint*-Befehl, denn ohne diesen sähe der Benutzer keinen echten Fortschritt. Der Bildschirm würde sonst erst am Ende der Prozedur automatisch wieder aktualisiert und der Balken spränge dann scheinbar direkt auf seinen 100%-Wert.

Damit *rcInnen* zur Entwurfszeit nicht *Width = 0* haben muss und dann kaum markierbar wäre, wird er in beliebiger Breite erstellt. Erst die *Form_Load*-Prozedur reduziert die Breite und macht ihn anfangs für Benutzer unsichtbar:

```
Private Sub Form_Load()
    Me.rcInnen.Width = 0
    Me.lblProzent.Caption = ""
End Sub
```

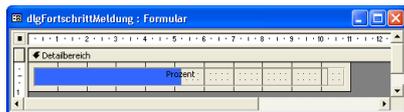
Im obigen Beispiel wird beim Fortschritt sogar laufend die Farbe des Balkens geändert, indem sich dessen *BackColor* dynamisch anpasst. In der Schleife selber sorgt das Überschreiten der halben Breite dafür, dass die Farbe der Beschriftung von Schwarz auf Weiß wechselt. Da die Farben von *rcInnen* recht dunkel sind, wäre der Text sonst nicht lesbar:



Fortschritts-Formular zentral nutzen

Anstatt nun wie hier auf jedem Formular einen Fortschrittsbalken anzulegen, kann man das auch zentral von einem speziellen Formular erledigen lassen, welches dann kurz eingeblendet wird.

Das (einfach aus dem vorigen Beispiel kopierte) neue Formular *dlgFortschrittMeldung* sieht im Entwurf praktisch ebenso aus wie eben, ist nur ohne Button. Wichtig ist vor allem, dass die Formular-Eigenschaft *PopUp* auf *Ja* steht, damit es nicht im Vollbild erscheint, sondern klein vor anderen Formularen:



Der Code ändert sich nun so, dass kein Klick-Ereignis und keine Schleife mehr im Formular enthalten ist, sondern lediglich mit der Prozedur *Balkenbreite()* über einen Parameter die Größe des inneren Rechtecks angegeben werden kann:

```
Private Sub Form_Load()
    Me.rcInnen.Width = 0
    Me.lblProzent.Width = Me.rcAussen.Width
End Sub

Sub Balkenbreite(dblAnteil As Double)
    Me.rcInnen.Width = Me.rcAussen.Width * dblAnteil
    Me.rcInnen.BackColor = CLng(32000 * dblAnteil)
    With Me.lblProzent
        .Caption = Format(dblAnteil, "0.0 %")
        .ForeColor = IIf(dblAnteil > 0.5, vbWhite, vbBlack)
    End With
    Me.Repaint
End Sub
```

Die Prozedur *Balkenbreite()* darf jetzt nicht mehr als *Private* gekennzeichnet sein, weil sie sonst von außen nicht sichtbar wäre. Von einem normalen Modul *modFortschritt* aus wird nun beispielhaft dieses Formular von außen aufgerufen:

```
Sub ZeigeFortschritt(strTitel As String)
    Dim dlgMeldung As Form_dlgFortschrittMeldung
    Dim lngZaehler As Long
    Const clngMaximum = 5000

    Set dlgMeldung = New Form_dlgFortschrittMeldung
    dlgMeldung.Visible = True
    dlgMeldung.Caption = strTitel

    For lngZaehler = 1 To clngMaximum
        dlgMeldung.Balkenbreite (lngZaehler / clngMaximum)
    Next

    DoCmd.Close acForm, dlgMeldung.Name
End Sub
```

Da der normale *DoCmd.OpenForm*-Aufruf wenig Kontrolle über das aufgerufene Formular ermöglicht, müssen wir uns daran erinnern, dass es sich bei Access-Formularen um Klassen-Objekte handelt. Sie können also auch "ehrllich" wie richtige Klassen instanziiert und zugewiesen werden.

Daher deklariert der Code zunächst eine Objekt-Variable der Klasse `Form_dlgFortschrittMeldung`, die von Access automatisch erst dann angelegt wird, sobald in einem Formular/Bericht Code enthalten ist. Es ist nicht bloß die allgemeine `Form`-Klasse, weil IntelliSense auch die Prozedur aus genau diesem Formular kennen soll.

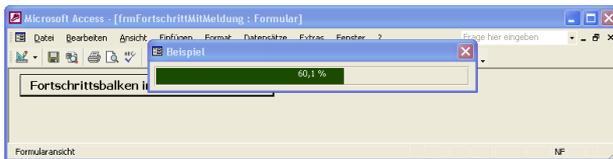
Dann wird mit dem Schlüsselwort `New` eine neue Instanz der Klasse angelegt und der Variablen `dlgMeldung` zugewiesen. Das Formular ist nun zwar geöffnet, aber noch nicht sichtbar, das muss in der nächsten Zeile noch nachgeholt werden.

Danach lässt sich in der Schleife einfach die Prozedur `Balkenbreite()` mit dem jeweiligen Anteil aufrufen und so der Fortschritt auch mit einem zentralen Formular anzeigen. Beim Schließen muss der Code für den `DoCmd.Close`-Befehl den Typ und Namen des Objekts nennen, weil ja auch dieser Befehl (anders als sonst meistens) von außen aufgerufen wurde.

Zum Testen genügt hier ein Formular mit einem Button und dessen Code wie folgt:

```
Private Sub btnMeldung_Click()
    modFortschritt.ZeigeFortschritt "Beispiel"
End Sub
```

Wenn die *Automatisch zentrieren*-Eigenschaft für `dlgFortschrittMeldung` auf `Ja` steht, erscheint die zentrale Fortschrittsmeldung nunmehr in der Bildschirmitte und zeigt den als Argument übergebenen Titel an:



Zentrales Formular in einer Schleife aufrufen

Das ist insgesamt schon ein Schritt in die richtige Richtung, da es jetzt nur noch des Aufrufs einer zentralen Prozedur `ZeigeFortschritt()` bedarf. Allerdings soll die Schleife selbstverständlich beim Aufrufer bleiben und nicht wie hier zu Demozwecken in der Prozedur enthalten sein. Der Start, das Weiterzählen sowie das Ende des Fortschrittsformulars werden also im allgemeinen Modul `modFortschritt` auf drei Prozeduren verteilt:

```
Dim m_dlgMeldung As Form_dlgFortschrittMeldung
Dim m_lngZaehler As Long
Dim m_lngMaximum As Long

Sub ZeigeFortschrittStart(strTitel As String, lngMaximum As Long)
    Set m_dlgMeldung = New Form_dlgFortschrittMeldung
    m_dlgMeldung.Visible = True
    m_dlgMeldung.Caption = strTitel
    m_lngMaximum = lngMaximum
    m_lngZaehler = 1
End Sub

Sub ZeigeFortschrittWeiter(Optional strText As String = "")
    m_lngZaehler = m_lngZaehler + 1
    m_dlgMeldung.Balkenbreite (m_lngZaehler / m_lngMaximum)
    If strText <> "" Then
        m_dlgMeldung.Caption = strText
    End If
End Sub

Sub ZeigeFortschrittEnde()
    DoCmd.Close acForm, m_dlgMeldung.Name
End Sub
```

Damit das zentrale Fortschrittsformular so überhaupt nutzbar ist, muss dessen Variable selbstverständlich (mindestens) Modul-öffentlich sein. Nun können die drei Teile von einer beliebigen Schleife aus aufgerufen werden.

Ein neues Formular `frmFortschrittMitMeldung_DateiImport` simuliert hierbei den Import von Daten aus einer Textdatei. Das dient hier allerdings nur der Demonstration einer Schleife im aufrufenden Code, die möglichst lange dauert, denn wirkliche Daten werden nicht verarbeitet. Dazu wird eine Textdatei mit den ersten vier Kapiteln aus Cäsars *De bello gallico* zeilenweise ausgelesen.

```
Private Sub btnMeldung_Click()
    Dim strZeile As String
    Dim lngZeilen As Long
    Dim lngKanal As Long

    lngKanal = FreeFile()

    Open CurrentProject.Path & "\DeBelloGallico.txt" For Input As #lngKanal
    Do While Not EOF(lngKanal)
        Input #lngKanal, strZeile
        lngZeilen = lngZeilen + 1
    Loop
    Close #lngKanal

    ZeigeFortschrittStart "Datei-Import", lngZeilen
    Open CurrentProject.Path & "\DeBelloGallico.txt" For Input As #lngKanal
```

```

Do While Not EOF(lngKanal)
    ZeigeFortschrittWeiter "Import: " & Left(strZeile, 20) & "..."
    Input #lngKanal, strZeile
Loop
Close #lngKanal

ZeigeFortschrittEnde
End Sub

```

Da die Anzahl der Zeilen vorher nicht bekannt ist, wird diese in einer ersten Schleife ermittelt und in `lngZeilen` gespeichert. Natürlich verdoppelt das in Wirklichkeit die benötigte Zeit, weil die Datei zweifach eingelesen wird, aber das ist hier unerheblich, weil es nur eine Funktionsdemo ist.

Mit `ZeigeFortschrittStart()` wird dann der zentrale Fortschrittsdialog geöffnet und die Startparameter übergeben. Innerhalb der Schleife zählt `ZeigeFortschrittWeiter()` automatisch weiter und am Ende schließt `ZeigeFortschrittEnde()` ihn wieder:



Schleife mit Esc-Taste abbrechen

Das Ganze lässt sich noch verbessern, indem die Schleife mittels Esc-Taste jederzeit unterbrochen werden kann. Das braucht wieder die Hilfe einer API-Funktion, nämlich `GetAsyncKeyState`. Im Modul `modFortschritt` verarbeitet das die eigene `EscAbbruch()`-Funktion:

```

Public Const VK_ESCAPE = &H1B

Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer

Function EscAbbruch() As Boolean
    Dim intState As Integer

    intState = GetAsyncKeyState(VK_ESCAPE)
    EscAbbruch = (intState And 2 ^ 16)
End Function

```

In der Schleife kann dann jederzeit abgefragt werden, ob zwischenzeitlich die Esc-Taste gedrückt worden war:

```

'...wie bisher
Do While Not EOF(lngKanal)
    ZeigeFortschrittWeiter "Import: " & Left(strZeile, 20) & "..."
    Input #lngKanal, strZeile
    If EscAbbruch() Then
        If MsgBox("Hoppla, trotzdem weitermachen?!", _
            vbQuestion + vbYesNo + vbDefaultButton2) = vbNo Then
            Exit Do
        End If
    End If
Loop
Close #lngKanal
'...weiter wie bisher

```

Nun verhält sich die Fortschrittsanzeige endlich so, wie ein Benutzer das von einem Programm erwartet:



Ein- und Ausklappen

Für eine Controlling-Datenbank sollten Kosten sowohl auf allgemeiner Abteilungsebene, auf darin enthaltener Projektebene sowie schließlich auf der Ebene einzelner Projekt-Teile eingegeben werden können. Selbstverständlich mussten die Kosten später kumuliert auf allen Ebenen sichtbar sein.

Das ist zwar in der grundsätzlichen Datenbank-Technik nicht besonders anspruchsvoll, weil es mit reflexiv verknüpften Tabellen lösbar ist, aber es gibt in Access keine brauchbare TreeView-artige Darstellung. Da der Kunde - wie viele andere inzwischen auch - keine Installation von Controls oder DLLs zulässt, musste eine übersichtliche Formularoberfläche alleine mit Access-Bordmitteln geschaffen werden.

Dabei gibt es verschiedene Herausforderungen:

- eine Darstellung wie im TreeView mit eingerückten Texten ist in Endlosformularen nicht vorgesehen
- jede Zeile muss die zusammengefassten Werte der untergeordneten Daten anzeigen können
- wegen der Übersicht ist vor allem das Ein- und Ausklappen von Zeilen mit untergeordneten Werten hilfreich

Wegen der einfacheren inhaltlichen Kontrolle ist das hier benutzte Beispiel mit Ländern, Bundesländern/Kantonen und Städten bzw. Stadtteilen realisiert. In der Tabelle `tblDaten` steht im Feld `datdatIDRef` die Referenz auf das übergeordnete Objekt und dessen `datID`-Wert drin, so dass sich eine verschachtelte Liste ergibt. Die anzuzeigenden Daten sind in `datBetrag` (Währung) und in `datGenehmigt` (Ja/Nein) enthalten:

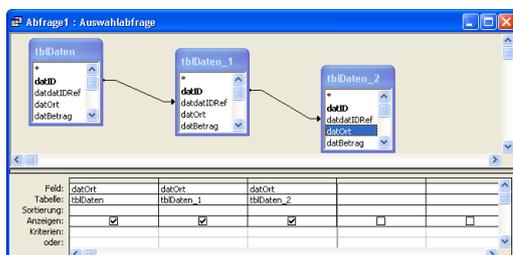
datID	datdatIDRef	datOrt	datBetrag	datGenehmigt	dat_Ebene	dat_Sort	dat_Expandier	datIstSichtbar	datZeigtPlus
1	0	Deutschland	31.353,00 €	<input type="checkbox"/>	1	01	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
2	1	Nordrhein-Westfalen	21.354,00 €	<input type="checkbox"/>	2	0102	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	2	Aachen	12.345,00 €	<input checked="" type="checkbox"/>	3	010203	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	2	Düsseldorf	2.435,00 €	<input type="checkbox"/>	3	010204	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	2	Köln	6.574,00 €	<input type="checkbox"/>	3	010205	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	1	Bayern	1.000,00 €	<input checked="" type="checkbox"/>	2	0106	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	0	Österreich	22.522,00 €	<input type="checkbox"/>	1	07	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
8	0	Schweiz	11.506,00 €	<input checked="" type="checkbox"/>	1	08	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
9	8	Genf	2.738,00 €	<input checked="" type="checkbox"/>	2	0809	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
10	8	Kanton Bern	8.768,00 €	<input checked="" type="checkbox"/>	2	0910	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
11	3	Burtscheid	99,00 €	<input checked="" type="checkbox"/>	4	01020311	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
12	6	Nürnberg	5,00 €	<input checked="" type="checkbox"/>	3	010612	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
13	6	München	110,00 €	<input type="checkbox"/>	3	010613	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
14	12	Langwasser	100,00 €	<input checked="" type="checkbox"/>	4	01061214	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
15	12	Gostenhof	200,00 €	<input type="checkbox"/>	4	01061215	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
16	12	Mögeldorf	300,00 €	<input checked="" type="checkbox"/>	4	01061216	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
25	10	Stadt Bern	2.245,00 €	<input checked="" type="checkbox"/>	3	081025	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
26	10	Thun	6.432,00 €	<input checked="" type="checkbox"/>	3	081026	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
27	10	Interlaken	8.461,00 €	<input checked="" type="checkbox"/>	3	081027	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
28	7	Burgenland	654,00 €	<input type="checkbox"/>	2	0728	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
29	28	Rust	3.213,00 €	<input type="checkbox"/>	3	072829	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
30	28	Eisenstadt	789,00 €	<input type="checkbox"/>	3	072830	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
31	7	Tirol	3.489,00 €	<input type="checkbox"/>	2	0731	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
32	31	Kitzbühel	6.654,00 €	<input type="checkbox"/>	3	073132	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
33	31	Kufstein	6.413,00 €	<input type="checkbox"/>	3	073133	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
34	0	Monaco	1.231.231,00 €	<input type="checkbox"/>	1	34	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
* (0Wert)	0		0,00 €	<input type="checkbox"/>	0		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Die Felder `datIstSichtbar` und `datZeigtPlus` dienen dem Zwischenspeichern der "TreeView"-Anzeige im zukünftigen Formular.

Eltern-Kind-Beziehungen ermitteln

Bevor die Daten in ihren jeweiligen Abhängigkeiten von übergeordneten Daten angezeigt werden können, müssen diese Eltern-Kind-Beziehungen jeweils ermittelt werden.

Aus Performance-Gründen (und da die Daten oft gelesen, aber selten neu geschrieben werden) gibt es keine Live-Berechnung der Daten-Abhängigkeiten. Stattdessen werden die in der Abfrage gefundenen Ebenen, Sortierung und Expandierbarkeit auf Knopfdruck aktualisiert und in zusätzlichen Feldern `dat_Ebene`, `dat_Sort`, `dat_Expandier` der Tabelle gespeichert. Diese sind im obigen ScreenShot bereits hell markiert, weil sie berechnete Werte enthalten.



Eine Analyse von reflexiv verknüpften Daten kann in einer Abfrage nach dem obigen Muster erfolgen, bei der die ursprüngliche Tabelle mehrfach enthalten ist und die Verbindung zwischen Primärschlüssel und Fremdschlüssel hergestellt wird. Daraus ergeben sich folgende Datenzeilen:

tblDaten.datOrt	tblDaten_1.datOrt	tblDaten_2.datOrt
Deutschland	Bayern	München
Deutschland	Bayern	Nürnberg
Deutschland	Nordrhein-Westfalen	Köln
Deutschland	Nordrhein-Westfalen	Düsseldorf
Deutschland	Nordrhein-Westfalen	Aachen
Nordrhein-Westfalen	Düsseldorf	
Nordrhein-Westfalen	Aachen	Burscheid
Nordrhein-Westfalen	Köln	
Aachen	Burscheid	
Düsseldorf		
Köln		
Bayern	Nürnberg	Gostenhof
Bayern	Nürnberg	Langwasser
Bayern	München	
Bayern	Nürnberg	Möggeldorf
Österreich	Tirol	Kurfstein
Österreich	Tirol	Kitzbühel
Österreich	Burgenland	Eisenstadt
Österreich	Burgenland	Rust
Schweiz	Kanton Bern	Thun
Schweiz	Kanton Bern	Statt Fiem

Das Hauptproblem ist dabei, dass vorher durch den Abfrage-Entwurf zwingend festgelegt wird, bis zu welcher Verschachtelungstiefe die Abhängigkeiten berücksichtigt werden. Daher kann hier in der zweiten Zeile der Stadtteil von Nürnberg nicht auch noch als Kind-Beziehung angezeigt werden.

Zudem tauchen alle Beziehungen auf mehreren Ebenen auf, hier *Deutschland-Bayern-Nürnberg* in Zeile 2 und erneut *Bayern-Nürnberg...* in den Zeilen 12, 13 und 15 wegen der Stadtteile. Da das nicht praktikabel ist, werden die Informationen zur Ebene und damit zusammenhängend der Sortierungs-Reihenfolge per VBA-Funktion ermittelt (und in der Tabelle gespeichert).

In einem Standard-Modul *modGenerierungen* werden entsprechende Funktionen bereitgestellt, die für jeden Datensatz herausfinden, ob er untergeordnete Daten ("Kinder") besitzt:

```
Dim m_rcsDaten As DAO.Recordset

Function HatKinder(lngID As Long) As Boolean
    If m_rcsDaten Is Nothing Then
        Set m_rcsDaten = CurrentDb.OpenRecordset("tblDaten", dbOpenDynaset)
    End If

    m_rcsDaten.FindFirst "[datdatIDRef]=" & lngID
    HatKinder = Not m_rcsDaten.NoMatch
End Function
```

Der Zugriff auf eine Modul-öffentliche Recordset-Variable *m_rcsDaten* sorgt für eine erhebliche Beschleunigung, da das Öffnen und Schließen eines Recordsets mit Abstand der langsamste Teil des Datenzugriffs ist. Es reicht hier, einfach zu suchen, ob ein anderer Datensatz diesen als *datdatIDRef*-Wert angegeben hat, denn dann hat er "Kinder" und ist also später ausklappbar.

Zusätzlich muss seine Stellung innerhalb der Hierarchie ermittelt werden. Dabei hat sich am praktikabelsten erwiesen, alle Daten mit ihrer zweistelligen ID als Zeichenkette hintereinander zu speichern. Deren Sortierung entspricht dem inhaltlichen Zusammenhang und deren Länge der Ebene, auf welcher sich der jeweilige Datensatz befindet. Im gleichen Modul muss dafür nur die Sortierungszeichenkette ermittelt werden:

```
Function MeineSortierung(lngID As Long) As String
    Dim strSortierung As String
    Dim lngIDFinden As Long

    If m_rcsDaten Is Nothing Then
        Set m_rcsDaten = CurrentDb.OpenRecordset("tblDaten", dbOpenDynaset)
    End If

    lngIDFinden = lngID
    Do
        m_rcsDaten.FindFirst "[datID]=" & lngIDFinden
        If m_rcsDaten.NoMatch Then
            Exit Do
        Else
            lngIDFinden = Nz(m_rcsDaten("datdatIDRef").Value, 0)
            If lngIDFinden = 0 Then
                Exit Do
            Else
                strSortierung = Format(lngIDFinden, "00") & strSortierung
            End If
        End If
    Loop

    MeineSortierung = strSortierung & Format(lngID, "00")
End Function
```

Die Funktion *MeineSortierung()* geht dabei von der aktuellen *datID* aus "aufwärts" anhand der jeweiligen *datdatIDRef* zum jeweiligen Eltern-Datensatz und fügt deren IDs als Zeichenkette aneinander. Deren Länge (wegen der "00"-Formatierung allerdings durch 2 geteilt) gibt die jeweilige Ebene an.

Beide Funktionen werden in einer Abfrage aufgerufen:

```
SELECT tblDaten.datID, tblDaten.datOrt,
    meineSortierung([datID]) AS Sort,
    Len([Sort])/2 AS Ebene,
    HatKinder([datID]) AS Expandierbar
```

```
FROM tblDaten;
```

Damit ergeben sich diese Daten:

datID	datOrt	Sort	Ebene	Expandierbar
1	Deutschland	01	1	-1
2	Nordrhein-Westfalen	0102	2	-1
3	Aschen	010203	3	-1
4	Düsseldorf	010204	3	0
5	Köln	010205	3	0
6	Bayern	0106	2	-1
7	Österreich	07	1	-1
8	Schweiz	08	1	-1
9	Genf	0809	2	0
10	Kanton Bern	0810	2	-1
11	Burtscheid	01020311	4	0
12	Nürnberg	010612	3	-1
13	München	010613	3	0
14	Langwasser	01061214	4	0
15	Gostenhof	01061215	4	0
16	Mögeldorf	01061216	4	0
25	Stadt Bern	081025	3	0
26	Thun	081026	3	0
27	Interlaken	081027	3	0
28	Burgenland	0728	2	-1
29	Rust	072829	3	0
30	Eisenstadt	072830	3	0
31	Tirel	0731	2	-1
32	Kitzbühel	073132	3	0
33	Kufstein	073133	3	0
34	Monaco	34	1	0

Diese werden dann in der Tabelle in den `dat_...-Feldern` gespeichert, damit die Datenanzeige später schneller geht. Damit ist der erste Teil erledigt, nämlich die Analyse der hierarchischen Daten.

Zwischenwerte berechnen

Mit diesen Informationen kann der zweite Schritt erfolgen: die Ermittlung der Zwischenwerte, also der Summe für die Währung in `datBetrag` und der Zusammenfassung der Ja/Nein-Werte in `datGenehmigt`. Auch dies erfolgt wieder über VBA-Funktionen:

```
' Achtung: public, damit ein Formular das zurücksetzen kann!
Public p_rcsKomplett As DAO.Recordset

Function MeineUntersumme(strSort As String, strFeldname As String) As Double
    If p_rcsKomplett Is Nothing Then
        Set p_rcsKomplett = CurrentDb.OpenRecordset("tblDaten", dbOpenDynaset)
    End If

    p_rcsKomplett.FindFirst "[dat_Sort] LIKE '" & strSort & "'"
    Do Until p_rcsKomplett.NoMatch
        MeineUntersumme = MeineUntersumme + Nz(p_rcsKomplett.Fields(strFeldname).Value, 0)
        p_rcsKomplett.FindNext "[dat_Sort] LIKE '" & strSort & "'"
    Loop
End Function
```

Auch hier beschleunigt eine öffentliche `Recordset`-Variable den Zugriff auf die Daten, allerdings diesmal Datei-öffentlich, damit sie auch vom Formular aus auf `Nothing` gesetzt werden kann. Das ist etwa dann nötig, wenn sich Daten ändern oder Datensätze hinzugefügt/gelöscht wurden.

Etwas schwieriger ist die Zusammenfassung der Ja/Nein-Werte in `datGenehmigt`, weswegen es auch ausdrücklich in das Beispiel aufgenommen wurde. Die Eltern-Ebene darf ja nur dann `Wahr` oder `Falsch` zusammenfassen, wenn auch alle Kind-Daten einheitlich `Wahr` oder `Falsch` sind. Sobald einer abweicht, kann die Zusammenfassung nur noch `Unklar` (hier als `Null` festgelegt, weil zum `TripleState` von `CheckBoxen` passend) sein.

```
Function MeinUnterjanein(strSort As String, strFeldname As String) As Variant
    If p_rcsKomplett Is Nothing Then
        Set p_rcsKomplett = CurrentDb.OpenRecordset("tblDaten", dbOpenDynaset)
    End If

    p_rcsKomplett.FindFirst "[dat_Sort] LIKE '" & strSort & "'"
    Do Until p_rcsKomplett.NoMatch
        If IsEmpty(MeinUnterjanein) Then
            MeinUnterjanein = CBool(p_rcsKomplett.Fields(strFeldname).Value)
        Else
            If MeinUnterjanein <> p_rcsKomplett.Fields(strFeldname).Value Then
                MeinUnterjanein = Null
                Exit Do 'kann sich nicht mehr ändern
            End If
        End If
        p_rcsKomplett.FindNext "[dat_Sort] LIKE '" & strSort & "'"
    Loop
End Function
```

Diese Funktionen werden in der Abfrage `qryDatenKomplett` aufgerufen:

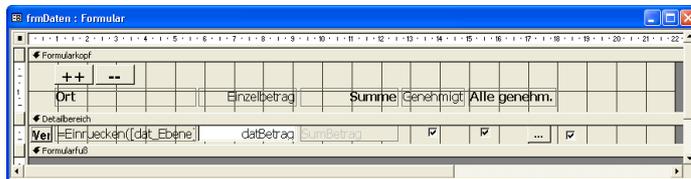
```
SELECT tblDaten.*,
       MeineUntersumme([dat_Sort],"datBetrag") AS SumBetrag,
       MeinUnterjanein([dat_Sort],"datGenehmigt") AS BooGenehmigt
FROM tblDaten
ORDER BY tblDaten.dat_Sort;
```

In den letzten beiden Spalten ist jeweils das Ergebnis dieser und aller untergeordneten Daten zu sehen, ein leeres Feld in `BooGenehmigt` entspricht dem Null-Wert:

idOrt	idEbene	Ort	datBetrag	datGenehmigt	dat_Ebene	dat_Sort	dat_Expandierbar	datZeigtPlus	SumBetrag	BooGenehmigt
0	0	Deutschland	31.353,00 €	<input type="checkbox"/>	01	01	<input type="checkbox"/>	<input type="checkbox"/>	75875	
2	1	Nordrhein-Westfalen	21.354,00 €	<input type="checkbox"/>	2.01.02	02	<input type="checkbox"/>	<input type="checkbox"/>	42807	
3	2	Aachen	12.345,00 €	<input checked="" type="checkbox"/>	3.01.0203	03	<input type="checkbox"/>	<input type="checkbox"/>	12444	-1
11	3	Burtscheid	99,00 €	<input checked="" type="checkbox"/>	4.01.020311	11	<input type="checkbox"/>	<input type="checkbox"/>	99	-1
4	2	Düsseldorf	2.435,00 €	<input type="checkbox"/>	3.01.0204	04	<input type="checkbox"/>	<input type="checkbox"/>	2435	0
5	2	Köln	6.574,00 €	<input type="checkbox"/>	3.01.0205	05	<input type="checkbox"/>	<input type="checkbox"/>	6574	0
6	1	Bayern	1.000,00 €	<input checked="" type="checkbox"/>	2.01.06	06	<input type="checkbox"/>	<input type="checkbox"/>	1715	
12	6	Münchberg	6,00 €	<input checked="" type="checkbox"/>	3.01.0612	12	<input type="checkbox"/>	<input type="checkbox"/>	606	
14	12	Langwasser	100,00 €	<input checked="" type="checkbox"/>	4.01.061214	14	<input type="checkbox"/>	<input type="checkbox"/>	100	-1
15	12	Gostenhof	200,00 €	<input type="checkbox"/>	4.01.061215	15	<input type="checkbox"/>	<input type="checkbox"/>	200	0
16	12	Milgater	300,00 €	<input checked="" type="checkbox"/>	4.01.061216	16	<input type="checkbox"/>	<input type="checkbox"/>	300	-1
13	6	München	110,00 €	<input type="checkbox"/>	3.01.0613	13	<input type="checkbox"/>	<input type="checkbox"/>	110	0
7	0	Österreich	22.522,00 €	<input type="checkbox"/>	1.07	07	<input type="checkbox"/>	<input type="checkbox"/>	43734	0
29	7	Burgenland	654,00 €	<input type="checkbox"/>	2.0729	29	<input type="checkbox"/>	<input type="checkbox"/>	654	0
28	29	rust	3.215,00 €	<input checked="" type="checkbox"/>	3.072928	28	<input type="checkbox"/>	<input type="checkbox"/>	3215	0
30	29	Eisenstadt	799,00 €	<input checked="" type="checkbox"/>	3.072930	30	<input type="checkbox"/>	<input type="checkbox"/>	799	0
31	7	Tirol	3.489,00 €	<input type="checkbox"/>	2.0731	31	<input type="checkbox"/>	<input type="checkbox"/>	16506	0
32	31	Kitzbühel	6.654,00 €	<input checked="" type="checkbox"/>	3.073132	32	<input type="checkbox"/>	<input type="checkbox"/>	6654	0
33	31	Kufstein	6.412,00 €	<input checked="" type="checkbox"/>	3.073133	33	<input type="checkbox"/>	<input type="checkbox"/>	6412	0
8	0	Schweiz	11.506,00 €	<input checked="" type="checkbox"/>	1.08	08	<input type="checkbox"/>	<input type="checkbox"/>	40150	-1
9	8	Grenf	2.739,00 €	<input checked="" type="checkbox"/>	2.0809	09	<input type="checkbox"/>	<input type="checkbox"/>	2739	-1
10	8	Kanton Bern	8.766,00 €	<input checked="" type="checkbox"/>	2.0810	10	<input type="checkbox"/>	<input type="checkbox"/>	25956	-1
25	10	Stattl Bern	2.245,00 €	<input checked="" type="checkbox"/>	3.081025	25	<input type="checkbox"/>	<input type="checkbox"/>	2245	-1
26	10	Thun	6.432,00 €	<input checked="" type="checkbox"/>	3.081026	26	<input type="checkbox"/>	<input type="checkbox"/>	6432	-1
27	10	Interlaken	9.481,00 €	<input checked="" type="checkbox"/>	3.081027	27	<input type="checkbox"/>	<input type="checkbox"/>	8481	-1
34	0	Monaco	1.231.231,00 €	<input type="checkbox"/>	1.34	34	<input type="checkbox"/>	<input type="checkbox"/>	1231231	0
0	0		0,00 €	<input type="checkbox"/>	0	0	<input type="checkbox"/>	<input type="checkbox"/>		

Daten visualisieren

Der letzte Schritt besteht darin, die Werte in einer TreeView-ähnlichen Darstellung anzuzeigen und auch Kind-Werte ein- und ausblenden zu können. Das alles geschieht mit einem normalen Endlosformular auf Basis von `qryDatenKomplett`:



Im Ergebnis sieht das Formular dann so aus:

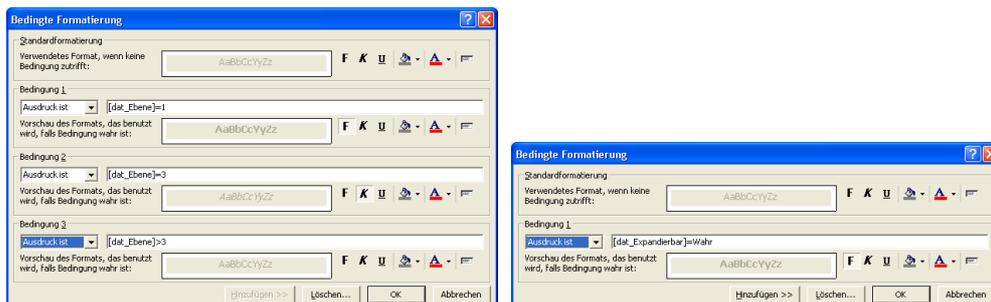


Da Access per VBA keinen Zugriff auf die verschiedenen Zeilen eines Endlosformulars zulässt, sind verschiedene Tricks nötig. Die "+"-"/"-"-Buttons links beispielsweise sind gar keine Schaltflächen, da sie dynamisch ihre Beschriftung ändern müssen. Es handelt sich vielmehr um eine Textbox (mit der Formel `=Wenn([dat_Expandierbar];Wenn([datZeigtPlus];"+";"-");"")` und erhöhtem Rand) sowie einem Button mit `Transparent:Ja`.

Das Einrücken der Orte geht nur mit einer VBA-Funktion `Einruecken()`, daher müssen diese Felder als berechnete Daten deaktiviert werden (das Editieren der Zeile erfolgt mit dem [...] -Button rechts):

```
Function Einruecken(varEbene As Variant, varInhalt As Variant) As Variant
    If IsNull(varInhalt) Then
        Einruecken = Null
    Else
        Einruecken = String((varEbene - 1) * 2, " ") & varInhalt
    End If
End Function
```

Als Formel steht darin dann `=Einruecken([dat_Ebene];[datOrt])`, so dass je nach Ebene einfach mehr oder weniger Leerzeichen davor stehen. Die unterschiedliche Schrift wird über ein bedingtes Format geregelt und ist daher nur für maximal 4 Ebenen veränderlich:



Entsprechendes gilt für die Währungs-Summen in `sumBetrag`, die für Zeilen mit untergeordneten Daten unterschiedlich formatiert werden (rechtes Bild).

Etwas anders funktioniert die Darstellung der Ja/Nein-Zusammenfassungen, weil Checkboxes keine Bedingte Formatierung kennen. Sie lassen sich jedoch mit Dreifacher `Status:Ja` so umschalten, dass ein Null-Wert eine eigene Darstellung erhält. Daher gab die `MeinUnterjanein()`-Funktion diesen Wert für den unklaren Fall zurück.

Checkboxen verbessern

Allerdings sind diese drei Darstellungen für Checkboxes nach meiner Erfahrungen von den meisten Benutzern nicht sicher zu unterscheiden und (als grundsätzliches Problem der Checkboxes unter Windows!) nicht in der Größe veränderlich.

Daher kann alternativ auch wieder eine Textbox eingesetzt werden, welche die drei anzuzeigenden Werte mit der Formel `=Wenn(IstNull([BooGenehmigt]);"ü";Wenn([BooGenehmigt];"p";""))` berechnet. In der Schriftart *Wingdings* sieht es dann so aus wie im folgenden Bild und macht nicht nur die Unterschiede deutlicher, sondern lässt sich auch besser grau färben:

Ort	Einzelbetrag	Summe	Genehmigt	Alle genehm.
Deutschland	31.353,00 €	75.875,00 €	<input type="checkbox"/>	ü✓ ...
Nordrhein-Westfalen	21.354,00 €	42.807,00 €	<input type="checkbox"/>	ü✓ ...
Aachen	12.345,00 €	12.444,00 €	<input checked="" type="checkbox"/>	ü✓ ...
Burscheid	99,00 €	99,00 €	<input checked="" type="checkbox"/>	ü✓ ...
Düsseldorf	2.435,00 €	2.435,00 €	<input type="checkbox"/>	ü✓ ...
Köln	6.574,00 €	6.574,00 €	<input type="checkbox"/>	ü✓ ...
Bayern	1.000,00 €	1.715,00 €	<input checked="" type="checkbox"/>	ü✓ ...
Österreich	22.522,00 €	43.734,00 €	<input type="checkbox"/>	ü✓ ...
Schweiz	11.506,00 €	40.150,00 €	<input checked="" type="checkbox"/>	ü✓ ...
Monaco	1.231.231,00 €	1.231.231,00 €	<input type="checkbox"/>	ü✓ ...

Ein- und Ausklappen

Das Ein- und Ausklappen der jeweils untergeordneten Elemente ist über das Setzen des `datIstSichtbar`-Feldes zu lösen. Entweder werden alle, deren `dat_Sort`-Zeichenkette wie diejenige des angeklickten Datensatzes anfängt, ausgeblendet (einklappen) oder alle, deren `dat_Sort`-Zeichenkette zwei Zeichen länger ist (also die genaue Kind-Ebene davon), werden eingeblendet (ausklappen):

```
Private Sub btnEinAus_Click()
    Dim strSQL As String
    Dim bkmHier As Variant
    Dim booPlus As Boolean

    booPlus = Me.datZeigtPlus.Value
    Me.datZeigtPlus.Value = Not Me.datZeigtPlus.Value

    bkmHier = Me.Bookmark
    If booPlus Then
        'ausklappen, also nur exakt die Unterebene sichtbar
        strSQL = "UPDATE tblDaten SET tblDaten.datIstSichtbar = -1, " & _
            "tblDaten.datZeigtPlus = -1 " & _
            " WHERE (dat_Sort LIKE '" & Me.dat_Sort.Value & "??');"
    Else
        'einklappen, also alle Unterebenen unsichtbar
        strSQL = "UPDATE tblDaten SET tblDaten.datIstSichtbar = 0 " & _
            " WHERE (dat_Sort LIKE '" & Me.dat_Sort.Value & "??');"
    End If

    CurrentDb.Execute strSQL

    Set p_rcsKomplett = Nothing 'erzwingt Neuladen bei Abfrage!
    Me.Requery
    Me.Bookmark = bkmHier
End Sub
```

Diese Prozedur steht im Formular selber und wird durch Klick auf den (transparenten) Button `btnEinAus` ausgelöst, der unsichtbar auf der Textbox am linken Rand liegt. Diese täuscht zwar durch erhöhten Rand einen Button vor, soll aber das Klick-Ereignis nicht erhalten, weil sie dann aktiv und also per Cursor markierbar wäre. Das sähe sehr irritierend aus.

Zusätzlich gibt es noch Prozeduren für die beiden Schaltflächen oben links, mit denen sich alle Zeilen einblenden oder (bis auf die erste Ebene) ausblenden lassen.

```
Private Sub btnAlleMinus_Click()
    CurrentDb.Execute "UPDATE tblDaten SET tblDaten.datIstSichtbar = 0 " & _
        "WHERE tblDaten.datdatIDRef<>0"
    CurrentDb.Execute "UPDATE tblDaten SET tblDaten.datIstSichtbar = -1, " & _
        "tblDaten.datZeigtPlus = -1 WHERE tblDaten.datdatIDRef=0"
    Me.Requery
End Sub

Private Sub btnAllePlus_Click()
```

```

CurrentDb.Execute "UPDATE tblDaten SET tblDaten.datIstSichtbar = -1, " & _
"tblDaten.datZeigtPlus = 0"
Me.Requery
End Sub

```

Je nach Datenmenge kann man nach Aktualisierung eines Datensatzes noch das Formular Neuberechnen lassen, so dass alle Zusammenfassungswerte wieder korrekt anzeigen:

```

Private Sub Form_AfterUpdate()
    Me.Recalc
End Sub

```

Da es sich im Original-Projekt natürlich um eine Mehrbenutzer-Umgebung handelt, müssen dort die Inhalte für `datIstSichtbar` und `datZeigtPlus` noch in lokalen Tabellen-Kopien verwaltet werden, damit sich die Benutzer nicht gegenseitig stören. Das erhöht den Organisationsaufwand leider noch ein wenig und ist hier weggelassen worden.

Bei diesem Beispiel war mein Kollege Peter Breuer an Ideen und Ausarbeitung beteiligt, dem ich hier für seine Mitarbeit danke.

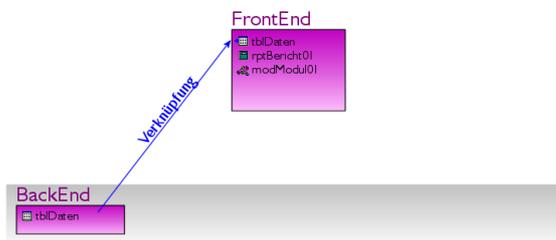


Fremde Berichte einbinden

Datenbanken für mehrere Benutzer enthalten meistens Berichte, wie sie für alle Benutzer vorgegeben werden. Früher oder später gibt es aber Abteilungen oder sogar einzelne Benutzer, die eigene Bericht benötigen.

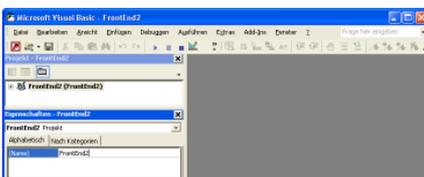
Jetzt gibt es die Auswahl zwischen Pest oder Cholera, denn entweder erhält dieser Benutzer Entwurfsrechte am allgemeinen FrontEnd und importiert dort seine persönlichen Berichte oder er macht sich eine Kopie davon mit seinen eigenen Berichten und muss dann alle Änderungen immer nachvollziehen. Beides ist unbrauchbar.

Die einfachste Lösung besteht darin, die persönlichen Berichte vom übrigen FrontEnd zu trennen. Leider scheint Access das nicht zu unterstützen. Mit ein wenig Trickserei geht es aber doch. Eine normale Datenbank mit Front-End und Back-End sieht typischerweise so aus wie im folgenden Bild:



Das BackEnd enthält die Daten in Tabellen, welche per Verknüpfung im FrontEnd genutzt werden. Ebenfalls im FrontEnd befinden sich alle anderen Elemente, vor allem die Berichte, die für alle gelten.

Ein beliebiger Benutzer mit eigenen Berichten erstellt nun zunächst ein FrontEnd2 mit den üblichen Verknüpfungen zu allen Daten, die für seine Berichte gebraucht werden. Damit es später im Verweis zu erkennen ist, sollte es einen eindeutigen internen Namen erhalten, hier `FrontEnd2`:



Außerdem muss der Aufruf der eigenen Berichte im FrontEnd2 stehen, daher enthält dessen Modul `modBerichte` die folgenden Prozeduren:

```

Sub ZeigeBericht(strName As String)
    DoCmd.OpenReport strName, acViewPreview
End Sub

Function BerichtsListe() As String
    Dim intZaehler As Integer

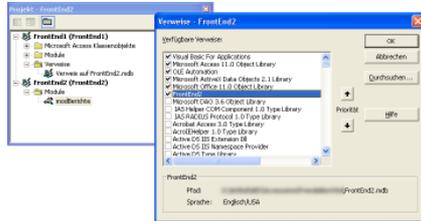
    With CodeProject.AllReports
        For intZaehler = 0 To .Count - 1
            BerichtsListe = BerichtsListe & .Item(intZaehler).Name & ";"
        Next
    End With
End Function

```

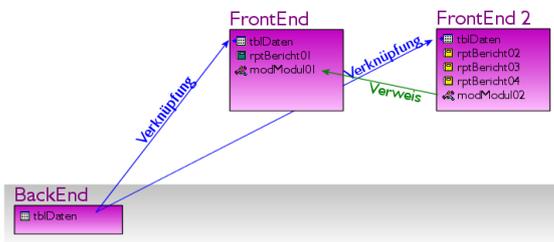
End With
End Function

Die Prozedur `ZeigeBericht()` ruft anhand des übergebenen Parameters später einen Bericht in der Seitenvorschau auf. Mit `BerichtsListe()` kann bei Bedarf eine Liste aller vorhandenen Berichte zurückgegeben werden. Dabei ist vor allem zu beachten, dass sich das `AllReports`-Objekt auf `CodeProject` (jenes, in dem der Code steht) und nicht auf `CurrentProject` (jenes, welches ein Benutzer geöffnet hat) bezieht.

Jetzt muss vom allgemeinen `FrontEnd1` aus ein Verweis auf dieses `FrontEnd2` eingerichtet werden, mit *Extras / Verweise* im VB-Editor:



Da Access-Datenbanken hier nicht automatisch aufgelistet werden, muss diese mit der *Durchsuchen*-Schaltfläche explizit angegeben werden. Der Verweis ermöglicht es ab jetzt einer Prozedur im `FrontEnd1`, auf eine öffentliche Prozedur von `FrontEnd2` zuzugreifen:



Dadurch lässt sich im `FrontEnd`-Modul folgender Code schreiben (wobei ja `FrontEnd2` der interne Name von `FrontEnd2` ist):

```
Sub ZeigeFremdenBericht(strName As String)
    FrontEnd2.ZeigeBericht strName
End Sub

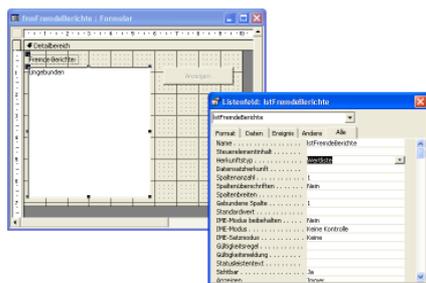
Function AlleFremdenBerichte() As String
    AlleFremdenBerichte = FrontEnd2.BerichtsListe()
End Function

Sub TestAlleFremdenBerichte()
    MsgBox "Liste: " & vbCrLf & Replace(BerichtsListe(), ";", vbCrLf)
End Sub
```

Die `TestAlleFremdenBerichte()`-Prozedur zeigt schon die grundsätzliche Funktionsfähigkeit mit der Anzeige der `FrontEnd2`-Berichte von `FrontEnd1` aus (damit die Namen zeilenweise untereinander stehen, tauscht die `Replace()`-Funktion dabei jedes Semikolon gegen einen Zeilenumbruch aus):



Das Semikolon als Trennzeichen war aber durchaus absichtlich gewählt, weil es für Listboxen als Zeilentrenner geeignet ist. In `FrontEnd1` wird nämlich zur Anzeige der fremden Berichte am besten ein Formular wie das folgende erstellt:



Die Listbox `IstFremdeBerichte` steht dabei für den *Herkunftstyp* schon auf `Wertliste`, damit die anzuzeigenden Namen als *Datensatzherkunft* (`RowSource`) angegeben werden können. Beim Laden wird die Zeichenkette mit den Namen der fremden Berichte dann nur noch an diese `RowSource`-Eigenschaft übergeben:

```

Private Sub btnAnzeigen_Click()
    FrontEnd2.ZeigeBericht Me.lstFremdeBerichte.Value
End Sub

Private Sub Form_Load()
    Me.lstFremdeBerichte.RowSource = FrontEnd2.BerichtsListe()
End Sub

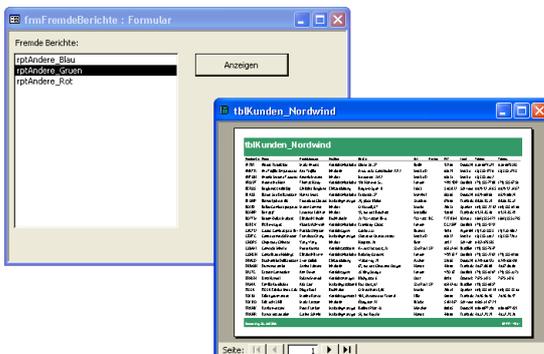
Private Sub lstFremdeBerichte_Click()
    Me.btnAnzeigen.Enabled = True
End Sub

```

Der Button *btnAnzeigen* ist im Entwurf deaktiviert und wird erst mit Auswahl eines Berichts wieder aktiv. Der Name dieses Berichts kann dann ohne weitere Prüfung in seinem *BeimKlicken*-Ereignis benutzt werden, weil er ja automatisch ermittelt wurde.



Die Berichte aus FrontEnd2 sind zur Unterscheidung bunt gefärbt und daher deutlich zu erkennen.



Damit gibt es eine flexible Möglichkeit, individuelle externe Berichte in ein allgemeines FrontEnd einzubinden.



Fazit

Ich hoffe, die hier vorgestellten *Accessoires* bieten ausreichend Ideen und Konzepte, wie eine Access-Datenbank mit wenig Aufwand besser organisiert werden kann. Der Code darf gerne nach eigenem Gutdünken verbessert werden, vor allem, weil aus Gründen der Übersicht keine Fehlerbehandlung enthalten ist. Die Ideen oder Prozeduren als eigene auszugeben oder gar zu verkaufen, fände ich aber doof.

Lorenz Hölischer