HAUFE.
Ausgabe 02/2026

ACCESS
IM UNTERNEHMEN

STEUERELEMENT-WIZARDS

Erstellen sie bessere Steuerelement mit unseren Wizards oder programmieren Sie diese gleich selbst (ab Seite 2).

In diesem Heft:

DATUM SCHNELL PER CURSOR EINSTELLEN

Nie mehr Probleme mit falschen Datumsformaten durch unsere Funktion zur Datumseingabe mit den Cursortasten!

SEITE 14

ORDNER UND DATEIEN IM TREEVIEW

Arbeiten Sie im Access-Formular mit Ordnern und Dateien wie im Windows Explorer.

SEITE 22

SCHNELLE BUTTONS MIT STIL

Erstellen Sie Schaltflächen mit Icon und weiteren Funktionen ganz einfach per Assistent.

SEITE 46

Mat-Nr. 01583-5154

Effizientes Formular-Design mit Steuerelement-Wizards

Was Programmierer am wenigsten mögen, sind immer wiederkehrende Aufgaben. Einen Button hinzufügen, Beschriftung und Steuerelementname festlegen, ein Icon einbinden und die zugehörige Ereignisprozedur anlegen – das alles gehört zu dieser Kategorie. Solche Handgriffe kosten einzeln betrachtet nur wenige Minuten, summieren sich über ein Projekt aber schnell zu einem spürbaren Zeitaufwand. Es lohnt sich also, einmal in die Automatisierung zu investieren, um danach dauerhaft davon zu profitieren.

Access erlaubt es, eigene Steuerelement-Assistenten zu programmieren, die beim Hinzufügen eines Steuerelements automatisch gestartet werden. Der Assistent kann entweder einen Konfigurationsdialog öffnen oder das Steuerelement direkt mit vordefinierten Eigenschaften anlegen – ganz nach Ihren Wünschen. Das Schöne daran: Ist ein solcher Assistent einmal fertig, steht er in jeder Datenbank zur Verfügung, ohne dass Sie ihn erneut einrichten müssen.

Was dazu an Know-how nötig ist, erläutern wir ab Seite 2 ausführlich im Beitrag **Steuerelement-Wizards programmieren**. Dort erfahren Sie, wie die Add-In-Datenbank für einen solchen Assistenten aufgebaut ist, wie Sie ihn registrieren und wie Sie ihn so programmieren, dass Steuerelemente künftig nach Ihren Vorgaben entstehen.

Gleich zwei fertige Beispiele haben wir für Sie vorbereitet. Der erste Assistent bietet einen Konfigurationsdialog für Schaltflächen: Beschriftung eingeben, Steuerelementname wird automatisch abgeleitet, Icon auf Wunsch mit wenigen Klicks hinzugefügt, Ereignisprozedur direkt ins Klassenmodul des Formulars geschrieben. Gerade bei Projekten mit vielen Formularen und Schaltflächen zahlt sich dieser Assistent schnell aus. Wie das im Einzelnen funktioniert, lesen Sie in **Schnelle Schaltflächen mit Stil** ab Seite 46.

Der zweite Assistent erleichtert das Anlegen eines Hauptformulars, dem automatisch ein Unterformular hinzugefügt wird, das die Daten einer frei wählbaren Tabelle oder Abfrage in der Datenblattansicht anzeigt. Was sonst mehrere manuelle Schritte erfordert, erledigt der Assistent auf Knopfdruck – beschrieben in **Form und Subform in der Datenblattansicht per Wizard** ab Seite 61.

Außerdem zeigen wir in **Datum schnell per Tastatur einstellen** ab Seite 14, wie Sie die Datumseingabe vereinfachen und Fehleingaben von vornherein verhindern. Wer Formulare mit Datumsfeldern pflegt, kennt das Problem: Benutzer tippen Daten in den unterschiedlichsten Formaten ein, und die Fehlerbehandlung kostet mehr Zeit als die eigentliche Eingabe. Die dort vorgestellte Technik schafft hier Abhilfe.

Aufbauend darauf überführen wir die Lösung in **Datum schnell einstellen mit Klasse** ab Seite 41 in ein Klassenmodul – damit lässt sie sich schnell beliebig vielen Datumsfeldern zuweisen, ohne den Code jedes Mal neu schreiben zu müssen.

Abgerundet wird die Ausgabe durch **Mit Ordnern und Dateien im TreeView arbeiten, Teil 1** ab Seite 22, wo wir zeigen, wie sich Ordner und Dateien in einem TreeView-Steuerelement komfortabel verwalten und navigieren lassen. Der Beitrag legt die Grundlage für eine praxistaugliche Dateiverwaltung direkt aus Access heraus.

Viel Spaß beim Lesen wünscht Ihnen

Ihr André Minhorst

Steuerelement-Wizards programmieren

Access bietet im Formular- und Berichtsentwurf die Möglichkeit, Steuerelemente mithilfe von Steuerelement-Assistenten zu erstellen. Das ist für Einsteiger eine Erleichterung, führt aber auch dazu, dass sie schnell Ergebnisse erzielen, deren Zustandekommen sie anschließend nicht mehr nachvollziehen können. Sie helfen nicht, die zum Anlegen von Steuerelementen und ihren Funktionen nötigen Schritte zu verstehen. Wenn man jedoch genau weiß, wie man Steuerelemente erstellt und programmiert, gelangt man schnell zu dem Punkt, dass man die gleichen Schritte immer wieder manuell durchführen muss. Und hier kommt ein spannendes Feature von Microsoft Access ins Spiel: Wir können auch selbst Steuerelement-Wizards programmieren, mit denen wir Steuerelemente genau nach unseren Wünschen anlegen können. Dabei sparen wir wertvolle Zeit, weil wir diese so gestalten können, dass nur noch wenige Eingaben nötig sind, um die gewünschten Steuerelemente zu erhalten. In diesem Beitrag zeigen wir, welche Steuerelement-Assistenten es schon gibt und wie wir eigene grundsätzlich hinzufügen können.

Steuerelement-Wizards in Access nutzen

Wenn wir mit einer frisch installierten Access-Anwendung arbeiten, werden die Steuerelement-Assistenten, soweit verfügbar, automatisch gestartet. Dazu müssen wir lediglich ein entsprechendes Steuerelement anlegen. Die meisten professionellen Entwickler haben dieses Feature jedoch deaktiviert, da sie die Assistenten ohnehin nicht nutzen.

Wenn wir dann in der Toolbox auf eines der Steuerelemente klicken, für das ein eingebauter Assistent bereitsteht, wird dieser beim Hinzufügen des Steuerelements angezeigt. So etwa der Assistent, der beim Hinzufügen einer Schaltfläche aufgerufen wird (siehe Bild 2).

Diese ist allerdings für einen professionellen Access-Entwickler nicht hilfreich, da er zum Beispiel keinen VBA-Co-

Wenn wir jedoch eigene Steuerelement-Wizards programmieren und nutzen wollen, müssen wir diese Funktion zunächst wieder aktivieren. Das gelingt, indem wir im Formular- oder Berichtsentwurf (im Folgenden werden wir uns auf den Formularentwurf beschränken) im Ribbon die Option **Steuerelement-Assistenten verwenden** aktivieren (siehe Bild 1).

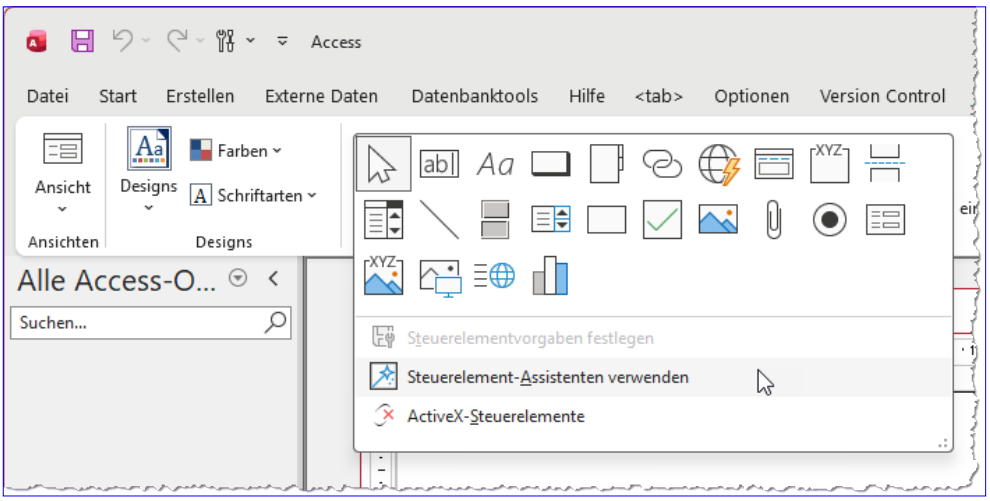


Bild 1: Aktivieren der Steuerelement-Wizards

de zum Ausführen erzeugt, sondern die Funktion in einem Makro hinterlegt.

Interessant sind zum Beispiel die Wizards für Kombinations- und Listenfelder. Starten wir diese durch Anlegen eines dieser Steuerelemente, finden wir einen Assistenten vor, der ähnliche Schritte anbietet wie der Assistent zum Erstellen von Nachschlagefeldern in Tabellen.

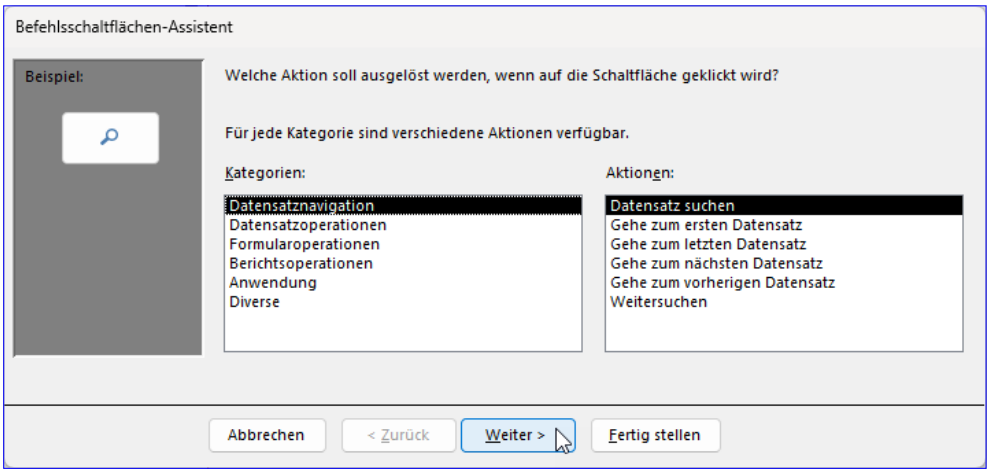


Bild 2: Der eingebaute Schaltflächen-Wizard von Access

Steuerelement-Wizards in der Registry

Ob ein Steuerelement-Wizard zur Verfügung steht, sehen wir in Access erst, wenn wir eines der Steuerelemente anlegen. Dann erscheint der Wizard, falls vorhanden, andernfalls wird das Steuerelement einfach zum Formularentwurf hinzugefügt.

Für welche Steuerelemente es eingebaute Wizards gibt, können wir in der Registry nachsehen. Diese befinden sich zum Beispiel für ein 32-Bit-Office unter dem folgenden Pfad:

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\
ClickToRun\REGISTRY\MACHINE\Software\Wow6432Node\Micro-
soft\Office\16.0\Access\Wizards\Control Wizards\
```

Hier sehen wir die verschiedenen Steuerelementtypen und darunter die dafür zur Verfügung stehenden Wizards (siehe Bild 3). Es gibt nur für wenige Steuerelemente solche Assistenten und alle haben nur ein eingebautes Exemplar.

Dadurch werden die Steuerelement-Assistenten auch direkt angezeigt, wenn wir das betreffende Steuerelement anlegen.

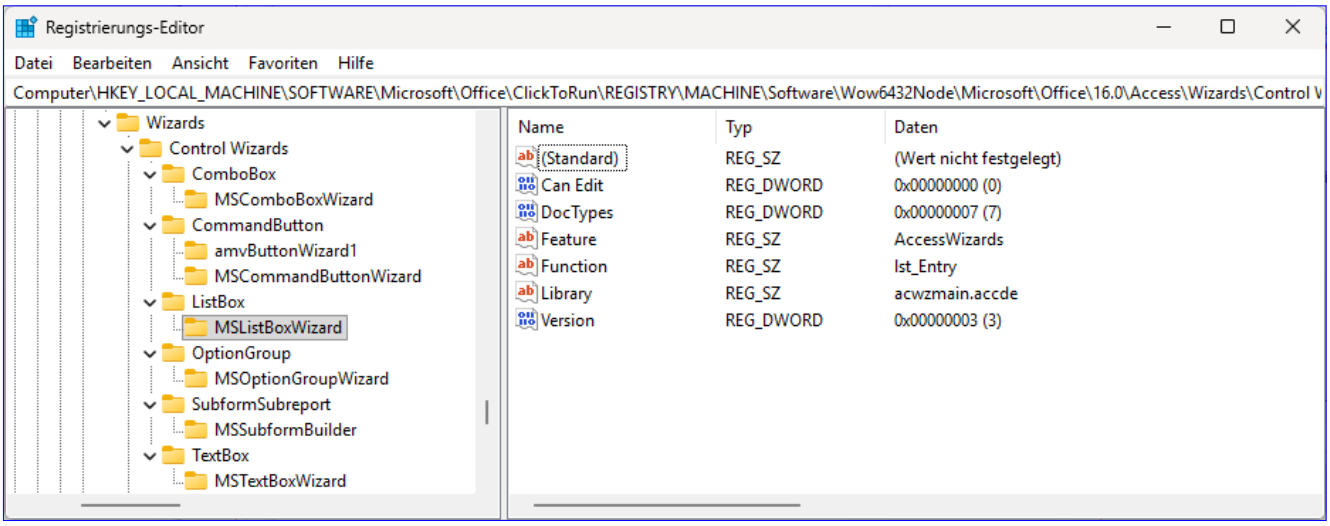


Bild 3: Steuerelement-Wizards in der Registry

Gibt es mehr als einen Wizard, weil wir beispielsweise selbst einen hinzugefügt haben, erscheint der Dialog **Generator auswählen** aus Bild 4, der alle verfügbaren Wizards anzeigt.

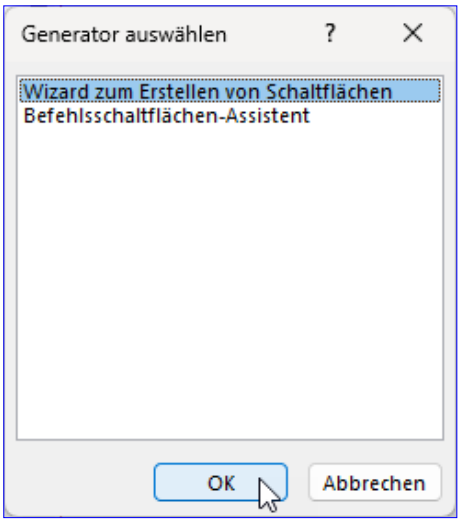


Bild 4: Mehrere Wizards für einen Steuerelementtyp

Eigenen Steuerelement-Wizard erstellen

Um einen benutzerdefinierten Steuerelement-Wizard zu erstellen, benötigen wir zunächst eine eigene Datenbank, die den Wizard enthält. Wir können über eine solche Datenbank allerdings auch mehrere Wizards gleichzeitig zur Verfügung stellen.

Diese Datenbank muss die Dateiendung **.accda** statt **.accdb** aufweisen, was wir durch Umbenennen erreichen.

Außerdem muss die Datenbank eine Tabelle namens **USysRegInfo** erhalten, welche die Informationen enthält, die beim Installieren des Wizards in die Registry übertragen werden sollen.

Diese Tabelle enthält Informationen wie die, die wir bereits weiter oben im Screenshot der Registry abgebildet haben.

Für einen Steuerelement-Wizard benötigen wir darin die folgenden Informationen:

- **Function**: Gibt die VBA-Funktion an, die beim Aufrufen des Wizards aufgerufen werden soll. Diese öffnet dann etwa ein Formular, über das wir die spezifischen Eigenschaften für das zu erstellende Steuerelement eingeben. Die Funktion muss ein spezielles Format aufweisen.
- **Library**: Enthält den Pfad zu der .accda-Datei mit den Funktionen des Wizards.
- **Description**: Hier können wir eine Beschreibung hinterlegen. Diese muss mit Bedacht gewählt werden, denn sie taucht in dem Dialog auf, der erscheint, wenn es mehrere Wizards für ein Steuerelement gibt. Man kann daher hier auch einfach den Namen des Wizards angeben und gegebenenfalls eine kurze Erläuterung.
- **Can Edit**: Gibt an, ob der Wizard bestehende Steuerelemente editieren kann (Wert **1**) oder ob er nur zum Anlegen neuer Steuerelemente zur Verfügung steht (Wert **0**).

Diese Werte hinterlegen wir in der bereits erwähnten Tabelle **USysRegInfo**. Am einfachsten übernehmen Sie diese aus der Beispieldatenbank. Das ist sogar zu empfehlen, da bereits kleinste Fehler dazu führen, dass der Wizard

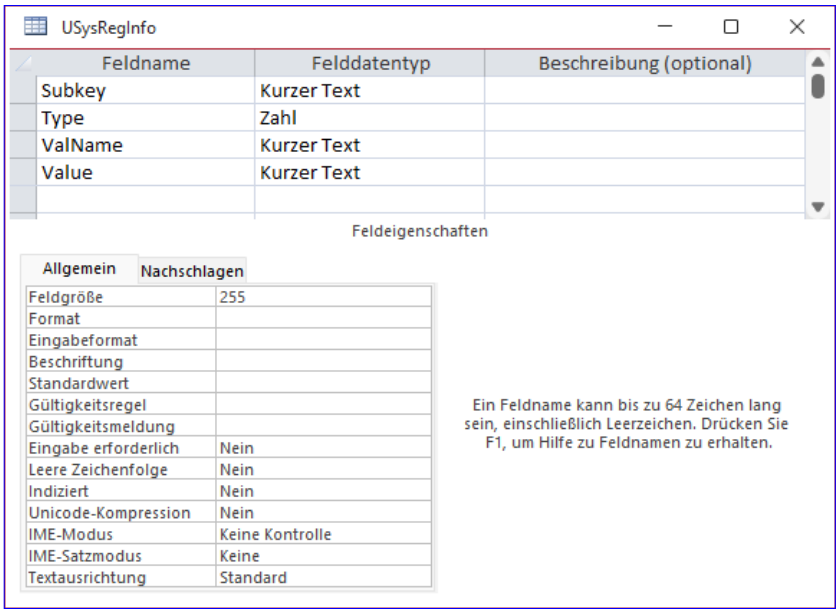
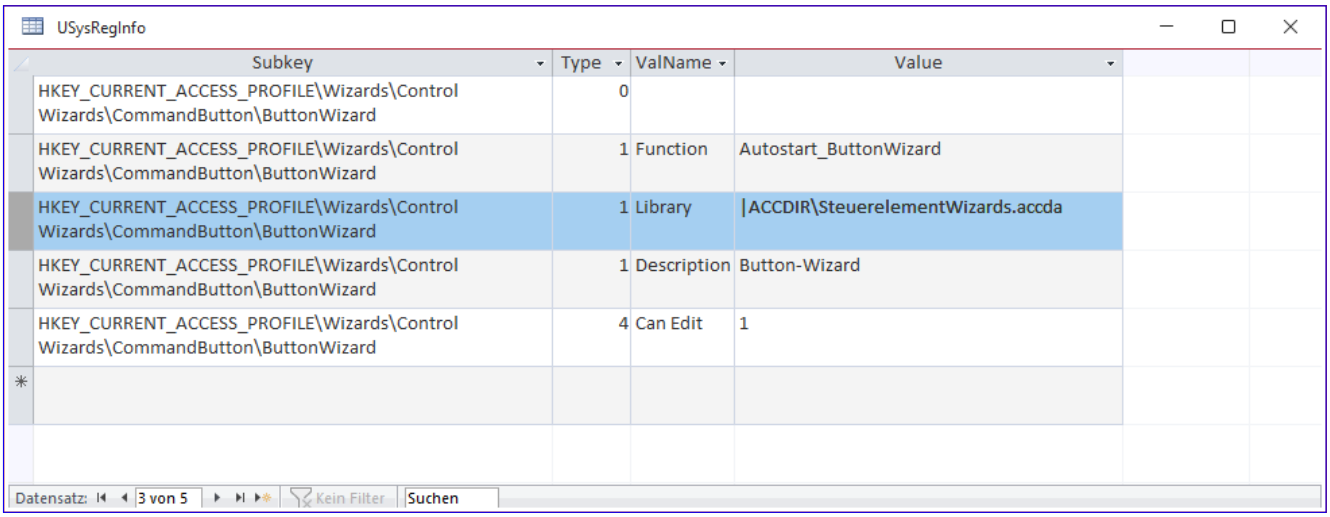


Bild 5: Die Tabelle **USysRegInfo** in der Entwurfsansicht



USysRegInfo

Subkey	Type	ValName	Value
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard	0		
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard	1	Function	Autostart_ButtonWizard
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard	1	Library	|ACCDIR\SteuerelementWizards.accda
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard	1	Description	Button-Wizard
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard	4	Can Edit	1

Datensatz: 3 von 5 Kein Filter Suchen

Bild 6: Die Tabelle **USysRegInfo** mit den Daten für einen Button-Wizard

nicht installiert werden kann. Ich habe einmal mehrere Stunden damit verbracht, einen solchen Fehler zu finden – am Ende lag es daran, dass ich statt **Can Edit** den Wert **CanEdit** (zusammengeschrieben) verwendet habe. Den Entwurf der Tabelle **USysRegInfo** sehen Sie in Bild 5.

Achtung: Wenn Sie die Tabelle anlegen oder importieren, ist diese normalerweise nicht zu sehen. Das liegt an dem Präfix **USys**. Diese Tabellen werden standardmäßig ausgeblendet und erscheinen erst, wenn wir in den Optionen des Navigationsbereichs die Eigenschaft **Systemobjekte anzeigen** aktivieren.

Für einen einfachen Button-Wizard tragen wir die Werte aus Bild 6 in diese Tabelle ein. Der erste Eintrag, dessen Felder **ValName** und **Value** leer sind, dient lediglich dazu, den Schlüssel wie in **Subkey** angegeben zur Registry hinzuzufügen. Die übrigen Einträge legen die Name-Wert-Paare unter diesem Schlüssel an.

Das Feld **Subkey** nimmt den Pfad auf, unter dem wir den Eintrag in der Registry erstellen wollen. Er lautet in diesem Fall:

```
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Control Wizards\CommandButton\ButtonWizard
```

Hier verwenden wir mit **HKEY_CURRENT_ACCESS_PROFILE** einen Platzhalter, der beim Installieren des Add-Ins automatisch durch den entsprechenden Zweig der Registry ersetzt wird. Dann folgt der Schlüssel **Wizards**, der alle Assistenten für verschiedene Elemente enthält. Der nächste Schlüssel **Control Wizards** gibt an, dass es sich bei dem Assistenten um einen Steuerelement-Assistenten handelt. Anschließend geben wir den Typ des Steuerelements an, für den wir den Assistenten aufrufen wollen, in diesem Fall **CommandButton**. Es gibt unter anderem die folgenden Typen:

Label, **TextBox**, **OptionGroup**, **ToggleButton**, **OptionButton**, **CheckBox**, **ComboBox**, **ListBox**, **CommandButton**, **Image**, **UnboundObjectFrame**, **BoundObjectFrame**, **PageBreak**, **SubformSubreport**, **Line** und **Rectangle**

Diese Bezeichnungen müssen ebenfalls exakt übernommen werden. Schließlich folgt der Name des Assistenten.

Der Wert im Feld **Subkey** muss für alle Angaben, die zu diesem Wizard in die Registry eingetragen werden sollen, exakt gleich sein.

Das Feld **Type** enthält den Typ des Registry-Eintrags. Er kann die folgenden Werte annehmen:

- **0**: Zeigt an, dass ein Schlüssel angelegt werden soll.
- **1**: Legt fest, dass ein Eintrag des Typs **REG_SZ** angelegt wird.
- **4**: Legt fest, dass ein Eintrag des Typs **REG_DWORD** angelegt wird.

Unter **ValName** geben wir den Namen des Name-Wert-Paares für den Eintrag an.

Und **Value** enthält schließlich den Wert. Hier verwenden wir für **Library** wiederum einen Platzhalter:

```
|ACCDIR
```

Dieser wird beim Installieren durch den Add-In-Ordner von Windows ersetzt, in den die Wizard-Datenbank beim Installieren kopiert wird. Dahinter müssen ein Backslash und der Name der Wizard-Datenbank folgen.

Funktion zum Starten des Wizards

Damit der Wizard gestartet werden kann, geben wir unter **Function** den Namen der VBA-Funktion an, die beim Starten ausgelöst werden soll.

Diese muss für die Verwendung in einem Steuerelement-Wizard ohne führendes Gleichheitszeichen und ohne folgendes Klammernpaar angegeben werden. In unserem Fall lautet der Eintrag **Autostart_ButtonWizard**.

Die dazu anzulegende VBA-Funktion muss ein bestimmtes Format aufweisen, da beim Aufrufen eines Steuerelement-Wizards zwei wichtige Informationen übergeben werden:

- Der Name des Steuerelements. Beim Anlegen eines neuen Steuerelements ist das der von Access vergebene Name, beim Editieren eines vorhandenen Steuerelements der gegebenenfalls bereits vom Benutzer geänderte Name.
- Der Name des Bezeichnungsfeldes des Steuerelements, soweit vorhanden. Steuerelemente mit dem Typ **TextBox**, **ListBox** oder **ComboBox** werden standardmäßig mit einem Bezeichnungsfeld angelegt, dessen Name hier übergeben wird.

Die Signatur der Funktion muss wie folgt aussehen:

```
Public Function Autostart_ButtonWizard( _
    Optional strControl As String, _
    Optional strLabel As String) As Variant

End Function
```

Zu Beispielzwecken füllen wir diese wie folgt:

```
Public Function Autostart_ButtonWizard( _
        Optional strControl As String, _
        Optional strLabel As String) As Variant
    Dim strMessage As String
    On Error GoTo Fehler
    strMessage = "Control-Wizard für '" & strControl & "'"
    If Not Len(strLabel) = 0 Then
        strMessage = strMessage & vbCrLf _
            & "mit Bezeichnungfeld '" & strLabel & "'"
    End If
    MsgBox strMessage, vbOKOnly + vbInformation, _
        "Button-Wizard"
Ende:
    Exit Function
Fehler:
    MsgBox Err.Number & " " & Err.Description
End Function
```

Wir wollen also einfach eine Meldung mit dem Namen des Steuerelements und gegebenenfalls dem Namen des Bezeichnungsfeldes ausgeben.

Eigenschaften der Wizard-Datenbank festlegen

Damit sind wir fast fertig. Es fehlen allerdings noch einige wichtige Einstellungen. Diese nehmen wir in den Eigen-

Datum schnell per Tastatur einstellen

Die Eingabe von Datumsangaben ist erstens manchmal unkomfortabel und zweitens passiert es oft, dass Benutzer das Datum im falschen Format eingeben. Dem wollen wir mit der Lösung aus diesem Beitrag vorbeugen, indem wir die Eingabe vollständig über die Tasten »Nach oben«, »Nach unten«, »Nach rechts» und »Nach links« ermöglichen. Sobald der Benutzer den Fokus auf ein Datumsfeld setzt, wird das aktuelle Datum eingesetzt, sofern das Feld noch leer ist. Dann kann er mit »Nach links« und »Nach rechts« zwischen den einzelnen Datumsbestandteilen Tag, Monat und Jahr wechseln und mit den Tasten »Nach oben« und »Nach unten« stellt er schrittweise den gewünschten Wert ein. Die Funktionalität bilden wir anschließend auch noch in einer Klasse ab, sodass die Funktion einfach mit zwei Zeilen Code zu einem Datums-Textfeld hinzugefügt werden kann.

Datumsfelder sind keine Hexerei. Man legt das Datumfeld in einer Tabelle an und legt die Tabelle als Datensatzquelle für ein Formular fest.

In diesem zieht man das Datumsfeld in den Formularentwurf und kann dieses direkt nutzen. Die Eingabe wird sogar vereinfacht durch das Anklicken des kleinen Kalender-Icons neben dem Textfeld (siehe Bild 1).

Wenn man hier allerdings einen Standardwert wie das aktuelle Datum vorgibt und der Benutzer soll dann sein Geburtsdatum eingeben, wird er schnell auf dieses Calendar-Widget verzichten. Denn dort müsste er endlos klicken, bis er zum Beispiel ein Datum erreicht, dass viele Jahre zurückliegt.

Auch die manuelle Datumseingabe birgt Tücken, denn hier muss das Format eingehalten werden. Das ist kein Hexenwerk, dennoch kommt es dabei immer wieder zu Fehleingaben.

Also stellen wir in diesem Beitrag eine Lösung vor, mit der wir das Datum ganz einfach eingeben können – nämlich allein mit den Cursor-Tasten, also **Nach oben**, **Nach unten**, **Nach links** und **Nach rechts**.

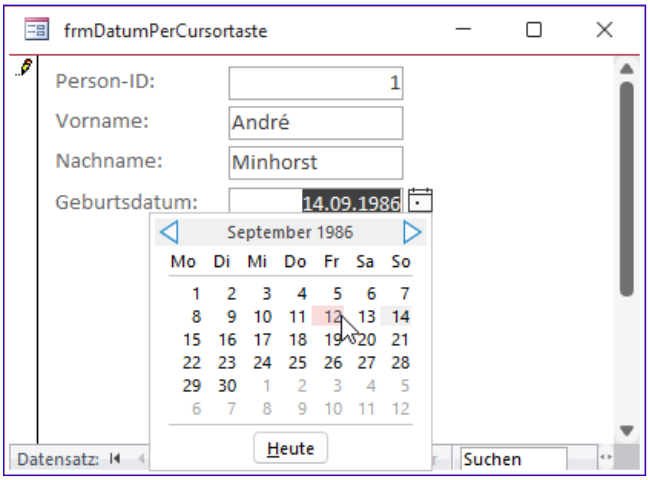


Bild 1: Herkömmliches Textfeld mit Datum

Dabei sollen gleich beim Fokuserhalt wie in Bild 2 die Tage markiert werden. Damit weiß der Benutzer, dass er nun

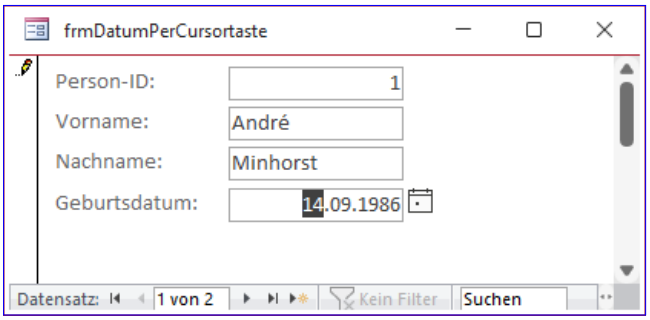


Bild 2: Markierung des Tages

die Tage für das gewünschte Datum einstellen kann. Er braucht nur noch den gewünschten Wert für den Tag mit den Tasten **Nach oben** und **Nach unten** einzustellen.

Dann betätigt er die Taste **Nach rechts**, um zum Monat zu wechseln, und legt auf die gleiche Weise wie beim Tag den Monat fest.

Schließlich springt er mit **Nach rechts** zu den Jahren und wählt auch noch das Jahr aus. Hierbei haben wir noch eine Komfortfunktion eingebaut, mit welcher der Benutzer bei gedrückter **Umschalt**-Taste 10 Jahre vor- oder zurückspringen kann.

Möchte man anschließend den Tag oder den Monat korrigieren, kann man diese mit der **Nach links**-Taste erneut ansteuern.

Programmierung der Lösung

In die Umsetzung der Lösung sind einige Ereignisse des Textfeld-Steuerelements involviert. Als Erstes benötigen wir ein Ereignis, mit dem wir direkt beim Setzen des Fokus auf das Textfeld dafür sorgen, dass die ersten beiden Zeichen für die Angabe des Tages selektiert werden.

Hier bietet sich das Ereignis **Bei Fokuserhalt** des Steuerelements an, für das wir die folgende Ereignisprozedur hinterlegen:

```
Private Sub txtGeburtsdatum_GotFocus()
    Call txtGeburtsdatumAktivieren
End Sub
```

Diese ruft eine Prozedur namens **txtGeburtsdatumAktivieren** auf. Wir lagern die Funktion des Ereignisses in eine eigene Prozedur aus, weil wir diese später noch von einer anderen Stelle aus aufrufen wollen.

Die Prozedur **txtGeburtsdatumAktivieren** nimmt einen Parameter namens **intDatePart** mit dem Datentyp **eDatePart** entgegen. Mit diesem können wir festlegen, welcher Teil des Datums – Tag, Monat oder Jahr – markiert werden soll. Die Enumeration definieren wir wie folgt:

```
Public Enum eDatePart
    datepartday = 0
    datepartmonth = 1
    datepartyear = 2
End Enum
```

Dann referenziert die Prozedur das Textfeld **txtGeburtsdatum** mit der Variablen **txt**. Dann prüft sie, ob das Textfeld leer ist, und trägt in diesem Fall das aktuelle Datum ein.

Das können Sie beliebig anpassen, wenn für den jeweiligen Anwendungsfall ein anderes Datum angezeigt werden soll. Schließlich ruft die Prozedur noch eine weitere Routine namens **SelectDatePart** auf und übergibt dieser einen Verweis auf das Textfeld und leitet den optionalen Parameter **intDatePart** weiter:

```
Private Sub txtGeburtsdatumAktivieren(Optional intDatePart
As eDatePart)
    Dim txt As TextBox
    Set txt = Me.txtGeburtsdatum
    If IsNull(txt) Then
        txt.Value = Date
    End If
    Call SelectDatePart(txt, intDatePart)
End Sub
```

Selektieren eines Datumsteils

Die Prozedur **SelectDatePart** nimmt einen Verweis auf das Textfeld entgegen, sowie einen Wert der Enumeration **eDatePart**, mit dem angegeben wird, ob der Tag, der Monat oder das Jahr selektiert werden soll.

Sie prüft, welchen Wert **lngCurrentDatePart** hat, und stellt abhängig davon die beiden Eigenschaften **SelStart** und **SelLength** des mit **txt** übergebenen Textfeldes ein. Wir gehen dabei davon aus, dass das Datum im Textfeld immer im Format **dd.mm.yyyy** formatiert ist.

Für den Tag wird **SelStart** mit dem Wert **0** auf das erste Zeichen eingestellt und die Länge der Markierung auf zwei Zeichen, für den Monat auf die Position **3** und die Länge **2** und für das Jahr auf die Position **6** und die Länge **4**:

```
Private Sub SelectDatePart(txt As TextBox, _
        lngCurrentDatePart As eDatePart)
    Select Case lngCurrentDatePart
        Case datepartday
            txt.SelStart = 0
            txt.SelLength = 2
        Case datepartmonth
            txt.SelStart = 3
            txt.SelLength = 2
        Case datepartyear
            txt.SelStart = 6
            txt.SelLength = 4
    End Select
End Sub
```

Damit ist der erste Teil bereits erledigt – beim Fokuserhalt des Textfeldes wird gegebenenfalls das aktuelle Datum eingetragen und die ersten beiden Stellen für den Tag werden markiert.

Bewegen der Markierung im Datumsfeld

Um die Eingaben des Benutzers im Datumsfeld zu erfassen, vor allem die Nutzung der Tasten **Nach oben**, **Nach unten**, **Nach links** und **Nach rechts**, verwenden wir das Ereignis **Bei Taste ab** des Textfeldes. Die entsprechende Ereignisprozedur liefert beim Aufruf zwei Parameter:

- **KeyCode**: Code der betätigten Taste
- **Shift**: Angabe, ob die **Umschalt**-, **Strg**- oder **Alt**-Taste betätigt wurde

Damit können wir sauber erfassen, wie der Benutzer über die Tastatur mit dem Textfeld arbeitet. Im ersten Schritt der Prozedur, deren ersten Teil wir in Listing 1 finden, deaktivieren wir mit **Me.Painting = False** das Neuzeichnen des Formulars, bis die Eingabe abgeschlossen ist.

Danach referenzieren wir mit der Variablen **txt** das Textfeld **txtGeburtsdatum**. Den Inhalt des Textfeldes untersuchen wir in den nächsten drei Anweisungen, um mit den Funktionen **Day**, **Month** und **Year** den Tag, den Monat und das Jahr zu ermitteln.

Danach rufen wir eine Hilfsfunktion namens **GetCurrentDatePart** auf, der wir einen Verweis auf das Textfeld übergeben. Sie soll bestimmen, ob gerade der Tag, der Monat oder das Jahr im Textfeld markiert ist. Dazu ermittelt sie die Startposition der Markierung und speichert sie in der Variablen **lngSelStart**.

Dann prüft sie, ob sich die Position zwischen **0** und **2** befindet (Tag), zwischen **3** und **5** (Monat) oder zwischen **6** und **9** (Jahr), und schreibt die entsprechende Konstante der Enumeration **eDatePart** in die Variable **lngCurrentDatePart**. Dieser Wert wird schließlich zurückgegeben:

```
Private Function GetCurrentDatePart(txt As TextBox) _
        As eDatePart
    Dim lngSelStart As Long
    Dim lngCurrentDatePart As eDatePart

    lngSelStart = txt.SelStart

    Select Case True
        Case lngSelStart >= 0 And lngSelStart <= 2
            lngCurrentDatePart = datepartday
        Case lngSelStart >= 3 And lngSelStart <= 5
            lngCurrentDatePart = datepartmonth
        Case lngSelStart >= 6 And lngSelStart <= 9
            lngCurrentDatePart = datepartyear
    End Select

    GetCurrentDatePart = lngCurrentDatePart
End Function
```

```
Private Sub txtGeburtsdatum_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim txt As TextBox
    Dim lngCurrentDatePart As eDatePart
    Dim lngDay As Long
    Dim lngMonth As Long
    Dim lngYear As Long
    Dim lngYears As Long

    Me.Painting = False
    Set txt = Me.txtGeburtsdatum
    lngDay = Day(txt.Value)
    lngMonth = Month(txt.Value)
    lngYear = Year(txt.Value)

    lngCurrentDatePart = GetCurrentDatePart(txt)

    If Shift = acShiftMask Then
        lngYears = 10
    Else
        lngYears = 1
    End If

    Select Case KeyCode
        Case vbKeyLeft, vbKeyRight
            Select Case KeyCode
                Case vbKeyLeft
                    Select Case lngCurrentDatePart
                        Case datepartday
                            lngCurrentDatePart = datepartyear
                        Case datepartmonth
                            lngCurrentDatePart = datepartday
                        Case datepartyear
                            lngCurrentDatePart = datepartmonth
                    End Select
                Case vbKeyRight
                    Select Case lngCurrentDatePart
                        Case datepartday
                            lngCurrentDatePart = datepartmonth
                        Case datepartmonth
                            lngCurrentDatePart = datepartyear
                        Case datepartyear
                            lngCurrentDatePart = datepartday
                    End Select
            End Select
            Call SelectDatePart(txt, lngCurrentDatePart)
            KeyCode = 0
            ...
```

Listing 1: Navigieren im Datum und Anpassen der einzelnen Datumsteile (Teil 1)

Mit Ordnern und Dateien im TreeView arbeiten, Teil 1

Im Artikel »Ordner und Dateien in Access-Tabellen einlesen« (www.access-im-unternehmen.de/1583) haben wir gezeigt, wie wir schnell Ordner und Dateien in Tabellen speichern, in »Dateien schnell im TreeView-Steuerelement anzeigen« (www.access-im-unternehmen.de/1584) haben wir Techniken vorgestellt, mit denen wir die Daten dieser Tabellen performant in einem TreeView-Steuerelement präsentieren. Im vorliegenden Artikel schauen wir uns nun an, wie man in diesem TreeView-Steuerelement mit den Ordnern und Dateien arbeiten kann: Wie zeigen wir einen Ordner direkt im Windows-Explorer an? Wie öffnen wir eine Datei direkt per Doppelklick? Wie können wir Kopieren, Ausschneiden und Einfügen im TreeView nutzen? Wie benennen wir Ordner und Dateien um? Und schließlich: Wie übertragen wir diese Änderungen direkt auf das Dateisystem?

Voraussetzung

Wir bauen in diesem Artikel auf dem Beispielformular **frmDateienImTreeView** auf, das wir im Artikel **Dateien schnell im TreeView-Steuerelement anzeigen** (**www.access-im-unternehmen.de/1584**) bereits vorgestellt haben (siehe Bild 1).

Ordner oder Dateien direkt im Windows Explorer anzeigen

Als Erstes wollen wir uns um die Anzeige eines Ordners oder einer Datei direkt im Windows Explorer kümmern und für diesen Zweck die benötigten Funktionen programmieren. Diese Funktionen werden wir später in verschiedenen Kontexten aufrufen – das Anzeigen eines Ordners über das Kontextmenü und das Anzeigen einer Datei im Windows Explorer über das Kontextmenü oder per Doppelklick auf das jeweilige Datei-Element im **TreeView**-Steuerelement.

Ordner direkt im Windows Explorer anzeigen

Die Prozedur zum Anzeigen eines Ordners im Windows Explorer ist denkbar einfach. Wir müssen nur die Anwendung **Explorer.exe** aufrufen und dieser den Pfad des anzuzeigenden Ordners übergeben. Außerdem können wir noch angeben, in welchem Fenstermodus der Windows Explorer geöffnet werden soll – hier wählen wir standardmäßig die normale Ansicht.

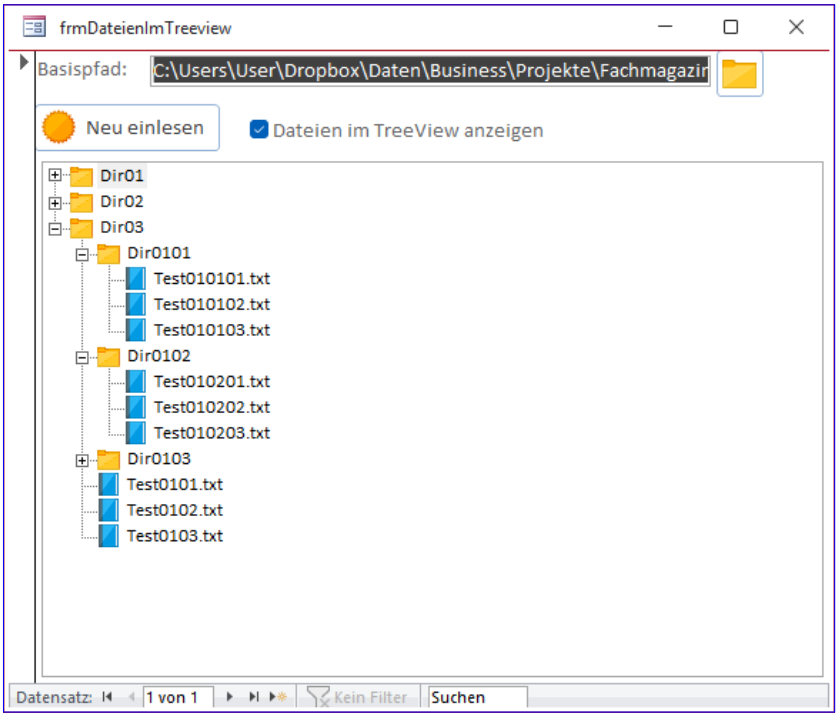


Bild 1: **TreeView**-Steuerelement mit den Einträgen, mit denen wir arbeiten wollen

Die Prozedur **OpenFolderInExplorer** nimmt den Pfad des anzuzeigenden Ordners als Parameter entgegen:

```
Public Sub OpenFolderInExplorer(strPath As String)
    Shell "explorer.exe """ & strPath & """", _
        vbNormalFocus
End Sub
```

Sie ruft die VBA-Funktion **Shell** auf und übergibt den in Anführungszeichen eingefassten Pfad als Parameter. Wenn wir etwa den Pfad des aktuellen Datenbankverzeichnisses anzeigen wollen, rufen wir die Prozedur beispielsweise wie folgt auf:

```
Call OpenFolderInExplorer(CurrentProject.Path)
```

Datei direkt im Windows Explorer anzeigen

Wir können noch einen Schritt weitergehen und direkt eine Datei im Windows Explorer markieren. Die entsprechende Prozedur nennen wir **ShowFileInExplorer**:

```
Public Sub ShowFileInExplorer(strPath As String)
    Shell "explorer.exe /select,""" & strPath & """", _
        vbNormalFocus
End Sub
```

Sie verwendet vor der Angabe des Pfades zu der zu markierenden Datei den Parameter **/select**. Diese Prozedur rufen wir beispielsweise für die aktuelle Datenbankdatei wie folgt auf:

```
Call ShowFileInExplorer(CurrentDb.name)
```

Datei mit der jeweiligen Anwendung öffnen

Die Prozedur, mit der wir eine Datei direkt in der jeweiligen Anwendung öffnen können, heißt **OpenFile** und verwendet die Methode **FollowHyperlink** des **Application**-Objekts:

```
Public Sub OpenFile(strPath As String)
    Application.FollowHyperlink strPath
End Sub
```

Um zum Beispiel eine Bilddatei aus dem aktuellen Datenbankverzeichnis anzuzeigen, nutzen wir den folgenden Aufruf:

```
Call OpenFile(CurrentProject.Path & "\pic005.png")
```

Angeklicktes Node-Element ermitteln

Für verschiedene Aktionen müssen wir zuvor das angeklickte **Node**-Element referenzieren – zum Beispiel, wenn wir eine Aktion beim Doppelklick darauf ausführen wollen.

Also deklarieren wir im Kopf des Klassenmoduls eine Variable, um das aktuelle Element zu referenzieren:

```
Dim objCurrentNode As MSComctlLib.Node
```

Wenn der Benutzer auf ein **Node**-Element klickt, soll dieses mit **objCurrentNode** referenziert werden. Wenn der Benutzer auf einen freien Bereich im **TreeView**-Steuerelement klickt, soll die Objektvariable **objCurrentNode** geleert werden.

Das erledigen wir mit der Ereignisprozedur, die beim Herunterdrücken der Maustaste ausgelöst wird. Wir legen die entsprechende Prozedur an, indem wir im Codefenster im linken Auswahlfeld den Eintrag **ctlTreeView** auswählen und im rechten Auswahlfeld **MouseDown**. Die so angelegte Prozedur füllen wir mit den folgenden Anweisungen:

```
Private Sub ctlTreeView_MouseDown( _
        ByVal Button As Integer, ByVal Shift As Integer, _
        ByVal x As Long, ByVal y As Long)
    Set objCurrentNode = ctlTreeView.Object.HitTest(x, y)
    If objCurrentNode Is Nothing Then
        Me.ctlTreeView.Object.SelectedItem = Nothing
    End If
End Sub
```

Die Prozedur erhält mit den Parametern **X** und **Y** die beim Klicken mit der Maus gültigen Koordinaten des

Mauszeigers. Diese werten wir mit der Eigenschaft **HitTest** aus.

Wenn der Benutzer ein **Node**-Element angeklickt hat, wird ein Verweis darauf zurückgeliefert. Wenn nicht, ist nicht nur **objCurrentNode** leer, sondern wir wollen dann auch den aktuell markierten Eintrag abwählen. Dazu stellen wir die Eigenschaft **SelectedItem** auf **Nothing** ein.

Aktuell markierte Datei per Doppelklick öffnen

Wenn wir die Funktion **OpenFile** per Doppelklick auf einen der Datei-Einträge im **TreeView**-Steuerelement aufrufen wollen, benötigen wir das Ereignis **DblClick**. Dieses liefert keinerlei Parameter, die Informationen über das angeklickte **Node**-Element enthalten.

Deshalb haben wir die zuvor erläuterte Ereignisprozedur angelegt. Klicken wir nun doppelt auf eines der Elemente, erhalten wir den Verweis auf das angeklickte **Node**-Element über die Objektvariable **objCurrentNode**. Wenn der Benutzer einen Doppelklick außerhalb der **Node**-Elemente durchgeführt hat, ist **objCurrentNode** leer.

Diese Variable werten wir in der folgenden Ereignisprozedur aus. Nur wenn **objCurrentNode** nicht **Nothing** ist, ermitteln wir den Key aus der **Key**-Eigenschaft und lesen danach mit **Left** das erste Zeichen aus, das entweder **f** (**Folder**) oder **i** (**File**) lautet.

Der Doppelklick soll nur etwas bewirken, wenn wir eine Datei angeklickt haben – im Falle eines Doppelklicks auf einen Ordner soll nur das Standardverhalten ausgelöst werden, in diesem Fall das Aufklappen der untergeordneten Elemente.

Wir haben in den Tabellen **tblFolders** und **tblFiles** nur die Namen des jeweiligen Ordners oder der Datei gespeichert, nicht den vollständigen Pfad zum entsprechenden Element.

Deshalb müssen wir, um die doppelt angeklickte Datei zu öffnen, erst den Pfad ermitteln. Dazu nutzen wir die Funktion **GetFilePath**, der wir die ID der Datei aus dem Key übergeben. Nachdem wir damit den Pfad ermittelt haben, können wir die Funktion **OpenFile** aufrufen und mit dieser die Datei mit dem entsprechenden Pfad öffnen:

```
Private Sub ctlTreeView_DblClick()
    Dim strPath As String
    Dim strKey As String
```

```
Public Function GetFilePath(ByVal lngFileID As Long) As String
    Dim lngFolderID As Long
    Dim strPath As String
    Dim strFolder As String
    lngFolderID = Nz(DLookup("ParentID", "tblFiles", "FileID = " & lngFileID), 0)
    strPath = Nz(DLookup("Filename", "tblFiles", "FileID = " & lngFileID), 0)
    Do While Not lngFolderID = 0
        strFolder = Nz(DLookup("Foldername", "tblFolders", "FolderID = " & lngFolderID), 0)
        strPath = strFolder & "\" & strPath
        lngFolderID = Nz(DLookup("ParentID", "tblFolders", "FolderID = " & lngFolderID), 0)
    Loop

    strPath = DLookup("Basispfad", "tblOptionen") & "\" & strPath
    GetFilePath = strPath
End Function
```

Listing 1: Die Funktion **GetFilePath** holt den Pfad zu dem Ordner oder der Datei mit der angegebenen ID.

```
    If Not objCurrentNode Is Nothing Then
        strKey = objCurrentNode.Key
        Select Case Left(strKey, 1)
            Case "i"
                strPath = GetFilePath(Mid(strKey, 2))
                Call OpenFile(strPath)
        End Select
    End If
End Sub
```

Pfad zu einer Datei ermitteln

Die Funktion **GetFilePath** erwartet die ID der Datei aus der Tabelle **tblFiles** (siehe Listing 1).

Sie liest zunächst die ID des übergeordneten Ordners aus dem Feld **ParentID** der Tabelle **tblFiles** aus.

Dann trägt sie in die Variable **strPath** zunächst nur den Dateinamen dieser Datei ein.

In der folgenden **Do While**-Schleife, die so lange durchlaufen wird, bis es keinen übergeordneten Ordner mehr gibt (**lngFolder = 0**) liest die Prozedur zunächst den Namen des übergeordneten Ordners aus der Tabelle **tblFolders** in die Variable **strFolder** ein.

Dann fügt sie diesen Namen, getrennt durch einen Backslash, vorn an den Inhalt der Variablen **strPath** an und speichert das Ergebnis wieder in **strPath**.

Außerdem ermittelt sie nun die ID des übergeordneten Elements für diesen Ordner aus der Tabelle **tblFolders** und schreibt diese in **lngFolderID**.

Auf diese Weise entsteht ein Pfad wie **Dir03\Dir0303\Test030303.txt**. Da wir die Ordner und Dateien immer von dem im Feld **Basisordner** der Tabelle **tblOptionen** aus in den Tabellen **tblFolders** und **tblFiles** speichern, müssen wir nun nur noch den dort gespeicherten Pfad hinzufügen, um den vollständigen Pfad zu erhalten.

Diesen gibt die Funktion als Ergebnis zurück.

Kontextmenü im TreeView-Steuerelement anzeigen

Wenn wir nicht nur Dateien öffnen, sondern diese und auch Ordner im Windows Explorer anzeigen wollen, benötigen wir ein Kontextmenü. Allein mit Klick-Befehlen lässt sich dies nicht so realisieren, dass es für den Benutzer noch intuitiv ist. Also fügen wir ein Kontextmenü hinzu, das je nach Position des Klicks unterschiedliche Optionen haben soll.

Beim Klick auf einen Ordner sollen folgende Befehle erscheinen. Alle Befehle sollen später mit dem Pendant für die entsprechende Aktion im »echten« Dateisystem ausgestattet werden:

- **Neuer Ordner**: Legt einen neuen Ordner-Node namens **Neuer Ordner** an und aktiviert direkt das Umbenennen des neuen Ordners.
- **Ordner löschen**: Löscht den Ordner-Node.
- **Ordner kopieren**: Kopiert den Ordner-Node.
- **Ordner ausschneiden**: Kopiert den Ordner-Node und löscht diesen, sobald er an anderer Stelle eingefügt wurde.
- **Einfügen**: Fügt einen zuvor kopierten oder ausgeschnittenen Ordner- oder Datei-Node in diesem Ordner ein.
- **Ordner im Windows Explorer öffnen**: Öffnet den Ordner im Windows-Explorer.
- **Dateipfad kopieren**: Kopiert den Dateipfad des Ordners in die Zwischenablage.
- **Ordner umbenennen**: Versetzt das Element in den Umbenennen-Modus

Beim Klick auf eine Datei sind diese Befehle vorgesehen:

- **Datei löschen**: Löscht den Node.
- **Datei kopieren**: Kopiert den Node in die Zwischenablage.
- **Datei ausschneiden**: Kopiert den Node in die Zwischenablage und löscht ihn, sobald dieser an anderer Stelle eingefügt wurde.
- **Datei im Explorer öffnen**: Zeigt die Datei im Explorer an.
- **Datei in Zielanwendung öffnen**: Öffnet die Datei in der für diese festgelegten Standardanwendung.

Und beim Klick auf den leeren Raum außerhalb der **Node**-Elemente bieten wir diese Befehle an:

- **Neuer Ordner**: Legt einen neuen Ordner namens **Neuer Ordner** an und aktiviert den Umbenennen-Modus.

Kontextmenüs anlegen

Kontextmenüs legen wir auf dem herkömmlichen Wege über die **CommandBars**-Auflistung an. Um diese anzuzeigen, verwenden wir die **MouseDown**-Eigenschaft des **TreeView**-Steuerelements. Dazu erweitern wir die bereits vorhandene Ereignisprozedur wie in Listing 2.

Das Ergebnis soll beispielsweise für Ordner-Elemente wie in Bild 2 aussehen.

Wir ermitteln hier nach wie vor das aktuell markierte **Node**-Element. Dann prüfen wir, ob der Benutzer die rechte Maustaste betätigt hat (**acRightButton**). Ist das der Fall, prüfen wir, ob aktuell ein **Node**-Element markiert ist.

```
Private Sub ctlTreeView_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As Long, ByVal y As Long)
    Dim strTyp As String

    Set objCurrentNode = ctlTreeView.Object.HitTest(x, y)

    If objCurrentNode Is Nothing Then
        Me.ctlTreeView.Object.SelectedItem = Nothing
    End If
    Select Case Button
        Case acRightButton
            If Not objCurrentNode Is Nothing Then
                strTyp = Left(objCurrentNode.Key, 1)
                Select Case strTyp
                    Case "f"
                        Call CreateCommandBarFolder()
                    Case "i"
                        Call CreateCommandBarFile()
                End Select
            Else
                Call CreateCommandBarOther
            End If
    End Select
End Sub
```

Listing 2: Prozedur zum Aufruf der verschiedenen Kontextmenüs

Falls ja, lesen wir aus der **Key**-Eigenschaft das erste Zeichen aus. Im Falle von **f** (**Folder**) rufen wir die Prozedur **CreateCommandBarFolder** auf. Lautet der Buchstabe **i** (**File**), rufen wir **CreateCommandBarFile** auf. Falls gerade kein **Node**-Element markiert ist, rufen wir die Prozedur **CreateCommandBarOther** auf.

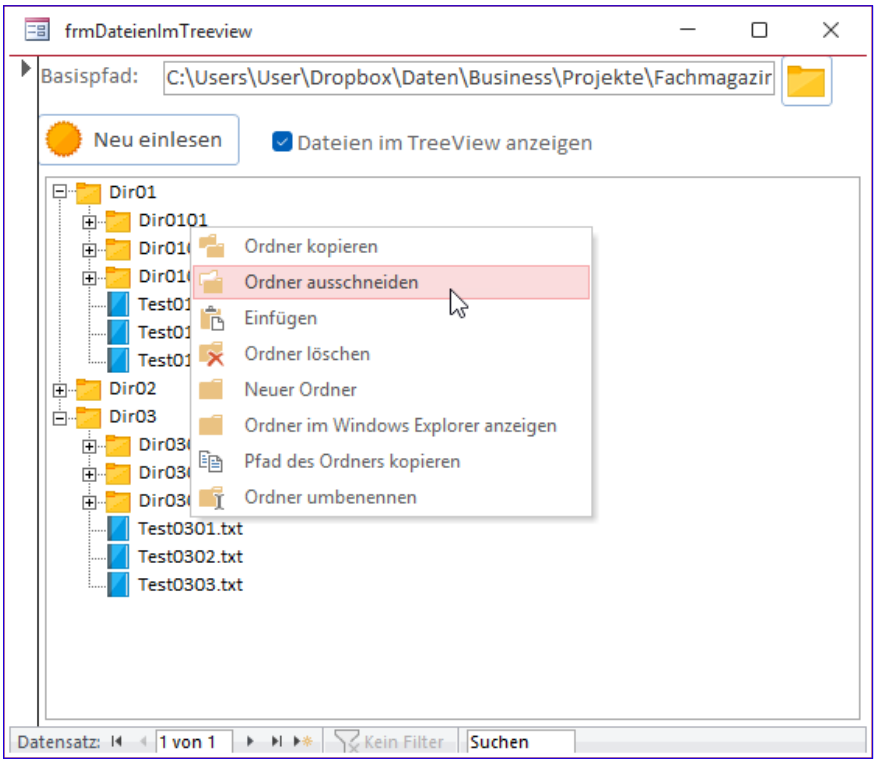


Bild 2: Anzeigen des Kontextmenüs für einen Ordner

Kontextmenü für Ordner anzeigen

Die Prozedur **CreateCommandBarFolder** soll das Kontextmenü für einen Ordner anzeigen (siehe Listing 3).

Sie nimmt den Verweis auf das **Node**-Element entgegen, für welches das Kontextmenü angezeigt werden soll. Dann liest es die ID des Elements aus, für das es angezeigt werden soll.

Falls das Kontextmenü bereits vorhanden ist, löschen wir es bei deaktivierter Fehlerbehandlung zunächst mit der **Delete**-Methode.

Danach legen wir ein neues Kontextmenü namens **Ordner** an und legen den Typ auf **msoBarPopup** fest. Außerdem stellen wir den Parameter **Temporary** auf **True** ein, damit es beim Beenden der Access-Session anschließend wieder gelöscht wird

Anschließend führen wir für die zu erstellenden Kontextmenü-Einträge jeweils ähnliche Befehle aus. Wir haben uns hier aus Platzgründen entschieden, nicht wie sonst üblich für jeden Kontextmenü-Eintrag eine eigene Variable zu deklarieren, die man normalerweise zum anschließenden Zuweisen der Eigenschaften nutzt. Stattdessen nutzen wir die **With**-Anweisung direkt mit dem neu erstellten Element und weisen damit die drei Eigenschaften zu:

- **Caption**: Erwartet die Beschriftung des Kontextmenü-Eintrags.
- **OnAction**: Hier hinterlegen wir die beim Anklicken auszuführende Funktion, die wir in einem Standardmodul als öffentliche Funktion anlegen müssen.
- **FaceID**: Erwartet einen Zahlenwert für ein eingebautes Menüsymbol.

Für die Eigenschaft **OnAction** legen wir jeweils eine Zeichenkette fest, die den Funktionsnamen enthält sowie den Key des aktuell markierten **Node**-Elements aus der Variablen **strKey**, zum Beispiel:

```
.OnAction = "=OrdnerKopieren('" & strKey & "')"
```

Schließlich zeigen wir das Kontextmenü mit der Methode **ShowPopup** an.

```
Private Sub CreateCommandBarFolder()
    Dim strKey As String
    Dim cbr As Office.CommandBar
    strKey = objCurrentNode.Key
    On Error Resume Next
    CommandBars("Ordner").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("Ordner", msoBarPopup, False, True)
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner kopieren"
        .OnAction = "=OrdnerKopieren('" & strKey & "')"
        .FaceId = 1667
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner ausschneiden"
        .OnAction = "=OrdnerAusschneiden('" & strKey & "')"
        .FaceId = 1669
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner einfügen"
        .OnAction = "=OrdnerEinfuegen('" & strKey & "')"
        .FaceId = 22
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner löschen"
        .OnAction = "=OrdnerLoeschen('" & strKey & "')"
        .FaceId = 2500
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Neuer Ordner"
        .OnAction = "=NeuerOrdner('" & strKey & "')"
        .FaceId = 1755 '4088 '4274 '2031 '
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner im Windows Explorer anzeigen"
        .OnAction = "=OrdnerExplorer('" & strKey & "')"
        .FaceId = 32
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Pfad des Ordners kopieren"
        .OnAction = "=OrdnerPfadKopieren('" & strKey & "')"
        .FaceId = 19
    End With
    With cbr.Controls.Add(msoControlButton)
        .Caption = "Ordner umbenennen"
        .OnAction = "=OrdnerUmbenennen('" & strKey & "')"
        .FaceId = 2512
    End With
    cbr.ShowPopup
End Sub
```

Listing 3: Prozedur zum Anzeigen des Kontextmenüs für Ordner

Funktionen für das Ordner-Kontextmenü

Das waren die einfachen Aufgaben. Nun folgen die einzelnen Funktionen, die beim Anklicken der Kontextmenü-Einträge ausgelöst werden sollen.

Diese legen wir im Standardmodul **mdlKontextmenues** nach dem folgenden Schema an – hier für das Kopieren eines Ordners:

```
Public Function OrdnerKopieren(strKey As String)
    MsgBox "Ordner kopieren"
End Function
```

Wie wir diese Funktionen füllen, sehen wir in den folgenden Abschnitten.

Ordner und Datei ausschneiden, kopieren und einfügen

Was genau soll beim Ausschneiden, Kopieren und Einfügen von Ordnern geschehen? Wir wollen zunächst auf der Ebene des **TreeView**-Steuerelements arbeiten. Das Nachziehen der Vorgänge im Dateisystem fügen wir später hinzu.

Beim Ausschneiden und Kopieren müssen wir uns eigentlich nur die ID des jeweiligen Elements merken, da die eigentliche Aktion erst durchgeführt wird, wenn wir den Ordner an anderer Stelle einfügen.

Wie benötigen also eine öffentlich deklarierte Variable, in der wir uns merken, welches Element ausgeschnitten oder kopiert werden soll. Dazu speichern wir am besten einfach den **Key** des Elements in dieser Variablen.

Außerdem müssen wir kenntlich machen, ob das Element mit diesem Key ausgeschnitten oder kopiert werden soll. Wir können dazu jeweils

eine Variable wie **strKeyOrdnerAusgeschnitten** und **strKeyOrdnerKopiert** anlegen.

Dann müssten wir, wenn wir einen Ordner kopieren, den Key in die Variable **strKeyOrdnerKopiert** schreiben und die Variable **strKeyOrdnerAusgeschnitten** leeren und umgekehrt – wir können schließlich nur einen Einfügevorgang durchführen, also müssen wir wissen, ob gerade ein Ordner zum Ausschneiden oder Kopieren markiert wurde.

Alternativ könnten wir einfach eine Variable namens **strKeyAusgeschnittenOderKopiert** anlegen und eine weitere **Boolean**-Variable, die angibt, ob ein Ordner ausgeschnitten oder kopiert werden soll. Wir entscheiden uns für die erste Variante:

```
Public strKeyOrdnerAusgeschnitten As String
Public strKeyOrdnerKopiert As String
```

Folglich hinterlegen wir für die beiden Kontextmenü-Funktionen die folgenden Anweisungen:

```
Public Function OrdnerKopieren(strKey As String)
    strKeyOrdnerKopiert = strKey
    strKeyOrdnerAusgeschnitten = ""
End Function

Public Function OrdnerAusschneiden(strKey As String)
    strKeyOrdnerKopiert = ""
    strKeyOrdnerAusgeschnitten = strKey
End Function
```

Einfügen eines ausgeschnittenen Ordners

Wenn wir mit der nächsten Funktion den Einfügevorgang durchführen wollen, benötigen wir Zugriff auf die **TreeView**-Objektvariable. Diese ziehen wir daher vom Klassenmodul in das Modul **mdlGlobal** und stellen die Sichtbarkeit auf **Public** ein:

```
Public objTreeview As MSComctlLib.TreeView
```

Wir beginnen mit dem Einfügen eines ausgeschnittenen Ordners, weil dies deutlich einfacher ist – auch wenn man es nicht glauben mag, denn das Ausschneiden besteht schließlich, oberflächlich betrachtet, aus zwei Schritten und das Kopieren scheinbar nur aus einem. Tatsächlich besteht das Ausschneiden aber nur im »Umhängen« des Nodes an einen anderen Parent-Node.

An dieser Stelle könnte man annehmen, dass wir uns das Anzeigen der Elemente im **TreeView**-Steuerelement durch Setzen der Eigenschaft **Sorted** auf den Wert **True** erleichtern könnten.

Das ist jedoch nicht der Fall, denn wir haben noch eine weitere Sortierungsebene: Es sollen zuerst die Verzeichnisse in aufsteigender Reihenfolge angezeigt werden und erst dann die Dateien, ebenfalls aufsteigend sortiert.

Wir müssen uns also selbst um die korrekte Reihenfolge kümmern. Beim erstmaligen Anzeigen ist das noch einfach, denn wir fügen die Elemente einfach in der entsprechenden Sortierung hinzu.

Beim späteren Ausschneiden oder Kopieren und Einfügen müssen wir uns allerdings mit einem selbst gebauten Mechanismus behelfen, um neu eingefügte Elemente an der richtigen Stelle einzufügen.

Schließlich müssen wir auch noch den Key des neuen übergeordneten Elements für den ausgeschnittenen und eingefügten Ordner-Datensatz in der Tabelle **tblFolder** definieren.

Die beim Aufrufen des Kontextmenü-Befehls **Einfügen** ausgelöste Prozedur finden wir in Listing 4. Sie nimmt den Key des Zielelements als Parameter entgegen und prüft, ob es ein kopiertes Element in **strKeyOrdnerAusgeschnitten** gibt.

Ist das der Fall, referenziert sie das entsprechende Element über den Key und prüft abermals, ob dieses Element

Datum schnell einstellen mit Klasse

Im Beitrag »Datum schnell per Tastatur einstellen« (www.access-im-unternehmen.de/1590) haben wir eine Lösung vorgestellt, mit der wir allein über die Tasten »Nach oben«, »Nach unten«, »Nach links« und »Nach rechts« das Datum in einem Textfeld einstellen können. Diese Lösung haben wir in diesem Beitrag speziell für ein Textfeld programmiert. Noch effizienter wäre die Lösung, wenn wir die Funktionalität in einer Klasse unterbringen, die man mit wenigen Codezeilen schnell zu einem Datums-Textfeld hinzufügen könnte. Genau das erledigen wir im vorliegenden Beitrag.

Die Lösung, die wir im oben genannten Beitrag vorgestellt haben, ermöglicht das einfache Navigieren in einem Datums-Textfeld. Beim Eintritt wird der Tag markiert, mit der Taste **Nach rechts** springt man zum Monat und zum Jahr und mit den Tasten **Nach oben** und **Nach unten** stellt man den aktuell markierten Teil des Datums ein. Mit **Nach links** kann man zu den vorherigen Datumsteilen zurückspringen. Außerdem können wir bei gedrückter Umschalttaste das Jahr in Zehnerschritten ändern (siehe Bild 1).

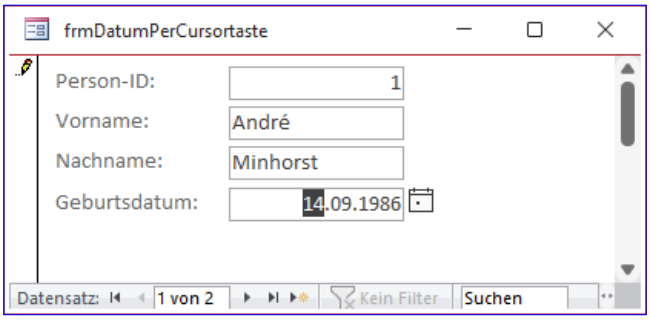


Bild 1: Navigieren im Datums-Textfeld

Allerdings umfassen die notwendigen Prozeduren rund 150 Zeilen Code, die man im aktuellen Zustand für jedes Datums-Textfeld erneut anlegen und anpassen müsste. Da es vorkommen kann, dass eine Anwendung viele Textfelder enthält, die alle mit dieser Funktion ausgestattet werden sollen, wäre das erstens eine Menge Arbeit und es würde eine Menge redundanter Code entstehen.

Daher haben wir uns entschlossen, die gesamte Funktionalität so in ein Klassenmodul zu übertragen, dass für die Verwendung nur noch wenige Zeilen Code notwendig sind.

Erstellen der Klasse

Den Code der Lösung haben wir in eine neue Klasse namens **clsDateBox** übertragen. Diese haben wir um zwei privat deklarierte Variablen erweitert:

```
Private WithEvents m_txt As TextBox
```

```
Private m_ActivateTabEnter As Boolean
```

Die erste Variable namens **m_txt** soll einen Verweis auf das Datums-Textfeld aufnehmen und ist mit dem Schlüsselwort **WithEvents** deklariert, damit wir für diese Variable Ereignisse innerhalb des Klassenmoduls definieren können.

Die zweite heißt **m_ActivateTabEnter** und liefert noch eine zusätzliche Funktion, die wir erst im Rahmen dieses Beitrags hinzugefügt haben: Wir können noch eine Eigenschaft namens **ActivateTabEnter** auf **True** einstellen.

Dies bewirkt, dass bei Betätigung der **Tab**- oder der **Enter**-Taste innerhalb des Datums-Textfeldes der Fokus nicht auf das nächste Steuerelement verschoben wird (oder bei gleichzeitigem Drücken der **Umschalt**-Taste auf das vor-

herige), sondern die Markierung im Datums-Textfeld von einem Datumselement zum nächsten bewegt wird.

Damit wir einen Verweis auf das mit der Funktion auszustattende Textfeld übergeben können, haben wir in der Klasse die folgende **Property Set**-Methode hinterlegt:

```
Public Property Set Datebox(txt As TextBox)
    Set m_txt = txt
    With m_txt
        .OnGotFocus = "[Event Procedure]"
        .OnKeyDown = "[Event Procedure]"
        .OnMouseUp = "[Event Procedure]"
    End With
End Property
```

Diese weist zunächst den mit **txt** übergebenen Verweis auf das Datums-Textfeld der Variablen **m_Txt** zu.

Danach stellen wir per VBA jeweils die Ereigniseigenschaften **Bei Fokuserhalt**, **Bei Taste ab** und **Bei Maustaste auf** auf die Eigenschaft **[Ereignisprozedur]** ein.

Wenn wir nun die entsprechenden Ereignisprozeduren im Klassenmodul festlegen, werden diese genauso ausgeführt wie Ereignisprozeduren, die wir im Klassenmodul des Formulars hinterlegt haben.

Allerdings unterscheidet sich der Weg zum Anlegen der Ereignisprozeduren für eine mit dem Schlüsselwort **WithEvents** deklarierte Variable von dem im Formular.

Hier können wir die Ereignisprozedur nicht durch Auswahl von **[Ereignisprozedur]** im Eigenschaftenblatt und anschließendes Anklicken der Schaltfläche mit den drei Punkten erstellen, sondern wir müssen diese über den VBA-Editor hinzufügen.

Dazu wählen wir im Codefenster im linken Auswahlfeld den Namen der mit **WithEvents** deklarierten Objektvariablen aus und im rechten das gewünschte Ereignis (siehe Bild 2).

Im linken Auswahlfeld erscheinen übrigens nur solche Objekte, die mit dem Schlüsselwort **WithEvents** deklariert wurden, oder, im Falle von Formular- oder Berichtsklassenmodulen, die Objekte für das Formular oder den Bericht und die enthaltenen Steuerelemente.

DateBox zu einem Formular zuweisen

Bevor wir uns den angepassten Code der Klasse anschauen, zeigen wir noch, wie man die Klasse produktiv einsetzen kann.

Dazu benötigen wir lediglich ein paar Zeilen Code in der Prozedur **Form_Load** des Klassenmoduls des Formulars und eine modulweit deklarierte Variable des Typs **clsDatebox**:

```
Dim objDatebox As clsDatebox
```

Das **Form_Load**-Ereignis ergänzen wir um die folgenden Anweisungen:

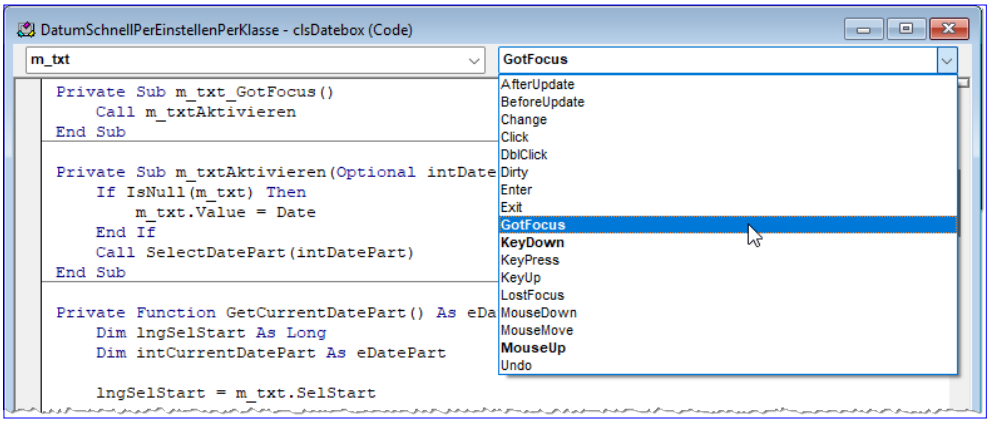


Bild 2: Hinzufügen von Ereignissen für benutzerdefinierte Objekte

Schnelle Schaltflächen mit Stil

Je schneller ich Ergebnisse beim Programmieren erhalten möchte, desto ungeduldiger werde ich, wenn mich kleinteilige, sich ständig wiederholende Aufgaben aufhalten. In diesem Fall ist vom Anlegen von Schaltflächen die Rede. Ich hätte gern so schnell wie möglich Schaltflächen, welche die von mir gewünschte Beschriftung, einen passenden Namen und je nach Situation ein Icon enthalten. Diese Icons sollen nach meinen Wünschen gestaltet sein – also beispielsweise mit transparentem Hintergrund und ohne Rahmen, sodass nur das Icon und die Beschriftung erscheinen und die Schriftfarbe und Schriftbreite diese als Schaltfläche von den Beschriftungen abhebt. Schließlich sollte auch noch direkt der VBA-Editor mit der passenden Ereignisprozedur geöffnet werden. Das alles umfasst einige Schritte, und auch wenn man eine Schaltfläche einmal formatiert hat und diese dann kopiert und als Vorlage für eine neue Schaltfläche verwendet, muss man noch einige Aspekte selbst hinzufügen. In diesem Beitrag stelle ich daher einen Assistenten vor, mit dem wir Schaltflächen wesentlich schneller und komfortabler anlegen können.

Schaltflächen – der Standardweg

Wenn ich in einem Formular eine neue Schaltfläche wie die aus Bild 1 erstellen möchte, sind einige Schritte nötig:

- Klick auf den Ribbon-Eintrag **FormularentwurflSteuerelementelSchaltfläche**
- Klick auf die Stelle im Formularentwurf, wo die Schaltfläche angelegt werden soll
- Anpassen der Beschriftung
- Wechsel zur Registerseite **Andere** im Eigenschaftenblatt und Anpassen des Steuerelementnamens
- Anlegen der Ereignisprozedur durch Wechsel zur Registerseite **Ereignisse** im Eigenschaftenblatt, Klick in die Eigenschaft **Beim Klicken**, Klick auf die Schaltfläche mit den drei Punkten
- Hinzufügen des Codes, der durch die Schaltfläche ausgelöst werden soll
- Hinzufügen eines Icons über den Ribbonbefehl **Bild einfügen**, hier gegebenenfalls noch Auswählen des gewünschten Bildes
- Einstellen der Eigenschaft **Anordnung der Bildbeschriftung** auf **Rechts**

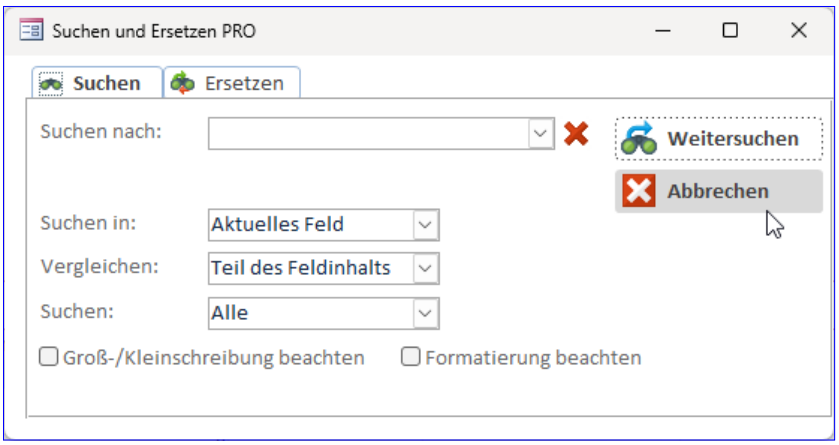


Bild 1: Individuelles Schaltflächendesign

- Hinzufügen eines Leerzeichens links von der Bildbeschriftung, damit diese ausreichend Abstand vom Icon hat

- Einstellen der Eigenschaft **Hintergrundfarbe** auf die Farbe des Formulars

- Einstellen der Eigenschaft **Rahmenart** auf **Transparent**

- Einstellung der Eigenschaften **Farbe beim Daraufzeigen** und **Farbe für gedrückten Zustand** auf die gewünschten Farben

- Einstellen der Eigenschaft **Schriftbreite** auf **Fett**

Damit ist die erste Schaltfläche angelegt.

Für die folgenden Schaltflächen wird es etwas einfacher, weil wir nun eine Schaltfläche haben, die wir bereits kopieren und als neue Schaltfläche einfügen können. Dennoch müssen wir dann folgende Einstellungen erneut vornehmen:

- Beschriftung einstellen

- Steuerelementname einstellen

- Icon auswählen

- Ereignisprozedur anlegen

Schaltflächen einfacher hinzufügen

Mit der nachfolgend beschriebenen Lösung wollen wir diese Schritte erheblich vereinfachen.

Dazu programmieren wir ein Formular, das die wichtigsten Konfigurationsmöglichkeiten für eine neue Schaltfläche enthält. In weiteren Beiträgen (siehe Ende des Beitrags) zeigen wir, wie die Lösung in einen Steuerelement-Wizard umgewandelt wird.

Damit wollen wir nur noch die sich unterscheidenden Merkmale anpassen – und auch das werden wir vereinfachen. Wir wollen hier nur noch die folgenden Aktionen durchführen müssen:

- Angabe der Beschriftung

- Auswahl des Icons

- Anschließendes Ergänzen der bereits angelegten Ereignisprozedur

Wenn wir die Beschriftung eingegeben haben, wollen wir daraus automatisch einen Steuerelementnamen ableiten. Wenn die Beschriftung etwa **Weitersuchen** lautet, soll automatisch der Steuerelementname **cmdWeitersuchen** vorgeschlagen werden.

Wenn sich die Beschriftung aus mehreren Wörtern zusammensetzt, soll der Steuerelementname aus dem gleichen Text bestehen – allerdings werden Leerzeichen entfernt, Umlaute und Sonderzeichen werden ersetzt und jedes

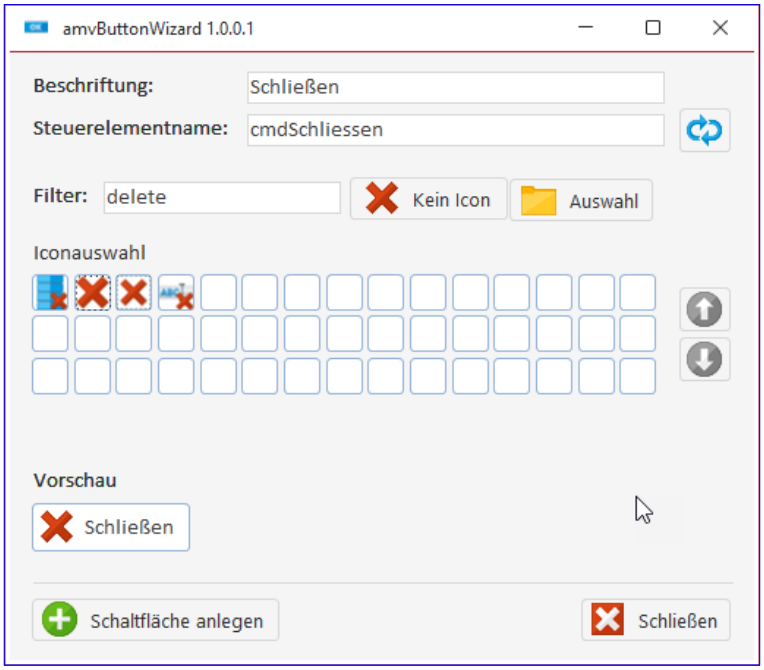


Bild 2: Das Formular **frmButtonWizard**

neue Wort wird großgeschrieben. Aus der Beschriftung **Nächste Fundstelle** wird so **cmdNaechsteFundstelle**.

Dieser Steuerelementname soll natürlich angepasst werden können, bevor das Steuerelement angelegt wird.

Funktionsweise der Lösung

Die Lösung besteht aus dem Formular aus Bild 2. In das Feld **Beschriftung** geben wir den Text ein, der auf der Schaltfläche angezeigt werden soll. Daraus wird automatisch der Name für das Steuerelement generiert, den wir anschließend nach Bedarf anpassen können.

Über die Schaltfläche **Auswahl** können wir einen Dateidialog öffnen, mit dem wir Icons auswählen, die für die zu erstellenden Schaltflächen verwendet werden sollen. Diese werden anschließend in den Bildsteuerelementen darunter angezeigt. Mit dem Feld **Filter** können wir gezielt nach den Namen der Bild-Dateien suchen. Wenn mehr Bilder vorhanden sind, als angezeigt werden können, erlauben die beiden Schaltflächen mit dem Pfeil nach oben und dem Pfeil nach unten das Blättern in den Icons.

Die Vorschau zeigt an, wie die fertige Schaltfläche später aussehen wird. Mit einem Klick auf Schaltfläche anlegen erstellen wir schließlich vollautomatisch die gewünschte Schaltfläche.

Bild 3 zeigt einige Beispiele für solche Schaltflächen.

Formular zum Anlegen von Steuerelementen

Um die gewünschten Einstellungen vorzunehmen, stellen wir ein entsprechendes Formular bereit. Dieses legen wir

```
Private Sub SteuerelementnameAktualisieren(strBeschriftung As String)
    Dim strTemp As String
    Dim strWoerter() As String
    Dim i As Integer
    If bolManuellGeaendert = True Then
        Exit Sub
    End If
    If Not Len(strBeschriftung) = 0 Then
        strWoerter = Split(strBeschriftung, " ")
        For i = LBound(strWoerter) To UBound(strWoerter)
            strTemp = strTemp & StrConv(Left(strWoerter(i), 1), vbUpperCase) & Mid(strWoerter(i), 2)
        Next i
        strTemp = Replace(strTemp, " ", "")
        strTemp = Replace(strTemp, "ä", "ae")
        strTemp = Replace(strTemp, "ö", "oe")
        strTemp = Replace(strTemp, "ü", "ue")
        strTemp = Replace(strTemp, "Ä", "Ae")
        strTemp = Replace(strTemp, "Ö", "Oe")
        strTemp = Replace(strTemp, "Ü", "Ue")
        strTemp = Replace(strTemp, "ß", "ss")
        strTemp = AlphanumerischBereinigt(strTemp)
        strTemp = "cmd" & strTemp
    Else
        strTemp = ""
    End If
    Me.txtSteuerelementname = strTemp
End Sub
```

Listing 1: Aktualisieren des Steuerelementnamens

in einer neuen, leeren Datenbankdatei unter dem Namen **frmButtonWizard** an.

Hier wollen wir zuerst die Beschriftung des Steuerelements abfragen und daraus direkt bei der Eingabe den Namen des Steuerelements ableiten. Dazu fügen wir die Steuerelemente aus Bild 4 hinzu.

Die erste Schaltfläche heißt **txtBeschriftung**, die zweite **txtSteuerelementname**. Die Schaltfläche nennen wir **cmdAktualisieren**.

Der Benutzer soll die Beschriftung der Schaltfläche in **txtBeschriftung** eingeben. Dadurch lösen wir bei der Eingabe eines jeden einzelnen Zeichens das Ereignis **Bei Änderung** aus. Für dieses hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub txtBeschriftung_Change()
    Call SteuerelementnameAktualisieren( _
        Me.txtBeschriftung.Text)
End Sub
```

Diese ruft die Prozedur **SteuerelementnameAktualisieren** auf und übergibt dieser den aktuellen Text des Textfeldes **txtBeschriftung** als Parameter.

In dieser Prozedur (siehe Listing 1) erzeugen wir mit **Split** aus den einzelnen Wörtern der Beschriftung ein Array namens **strWoerter** – dieses kann auch nur ein Element enthalten. Danach durchlaufen wir alle Elemente des Arrays und fügen diese aneinander. Dabei wird das erste Zeichen eines jeden Wortes großgeschrieben und bei den

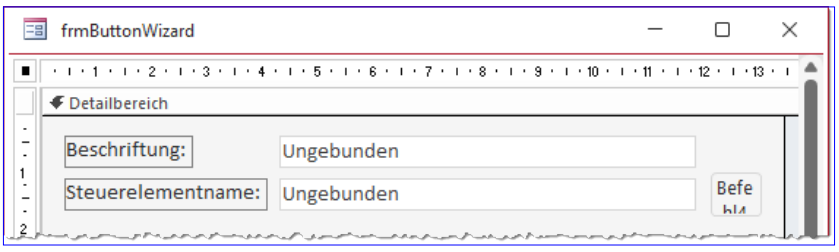


Bild 4: Erster Entwurf des Formulars

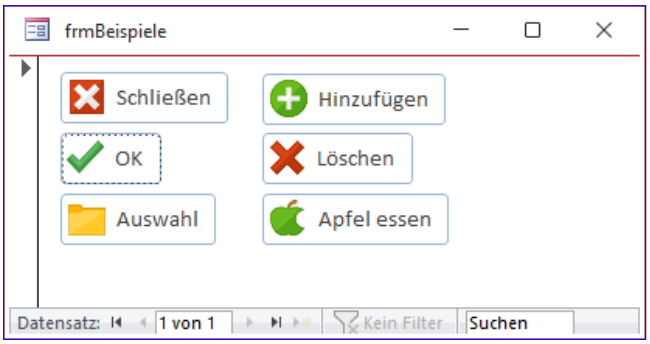


Bild 3: Einige Beispielschaltflächen, die mit der Lösung aus diesem Beitrag erstellt wurden

übrigen wird die Schreibweise beibehalten. So wird aus **Datensatz anlegen** der Text **DatensatzAnlegen**.

Danach ersetzt die Prozedur die Umlaute und das scharfe ß durch die entsprechenden Umschreibungen. Schließlich lassen wir den aktuellen Text noch durch die Funktion **AlphanumerischBereinigen** laufen, in der alle Zeichen außer a-z, A-Z und 0-9 entfernt werden. Diese Funktion sehen wir in Listing 2.

Es kann auch sein, dass der Benutzer den so generierten Steuerelementnamen manuell anpassen möchte. Das kann er tun und wir stellen dann den Wert einer Variablen namens **bolManuellGeaendert**, die wir wie folgt deklarieren, auf **True** ein:

```
Private bolManuellGeaendert As Boolean
```

Bei einer Änderung im Textfeld **txtSteuerelementname** wird die folgende Ereignisprozedur ausgelöst, welche **bolManuellGeaendert** auf **True** einstellt, den aktuellen Text in einer **TempVar**-Variablen speichert (so kann diese beispielsweise bei Laufzeitfehlern nicht gelöscht werden) und anschließend die Routine **VorschauAktualisieren** aufruft, die wir weiter unten beschreiben:

```
Private Sub txtSteuerelementname_Change()
    bolManuellGeaendert = True
```

```
Private Function AlphanumerischBereinigt(ByVal strEingabe As String) As String
    Dim i As Long
    Dim strErgebnis As String
    Dim strZeichen As String

    For i = 1 To Len(strEingabe)
        strZeichen = Mid$(strEingabe, i, 1)
        If strZeichen Like "[A-Za-z0-9]" Then
            strErgebnis = strErgebnis & strZeichen
        End If
    Next i

    AlphanumerischBereinigt = strErgebnis
End Function
```

Listing 2: Entfernen nicht-alphanumerischer Zeichen

```
    TempVars("Steuerelementname") = _
        Me.txtSteuerelementname.Text
    Call VorschauAktualisieren
End Sub
```

Schließlich fehlt noch die Schaltfläche **cmdAktualisieren**. Diese soll, falls der Benutzer den Steuerelementnamen bereits manuell angepasst hat, die automatische Aktualisierung wieder aktivieren, indem sie erstens eine initiale Aktualisierung durchführt und **bolManuellGeaendert** wieder auf **False** einstellt.

Dadurch werden folgende Änderungen in **txtBeschriftung** wieder in **txtSteuerelementname** übertragen:

```
Private Sub cmdAktualisieren_Click()
    Me.txtFilter.SetFocus
    Call SteuerelementnameAktualisieren( _
        Nz(Me.txtBeschriftung, ""))
    bolManuellGeaendert = False
End Sub
```

Icon auswählen

Nun folgt ein spannender Teil. Wie wollen wir die Auswahl eines Icons realisieren? Wir könnten einfach einen Dateiauswahl-Dialog öffnen und den Benutzer das gewünschte Icon auswählen lassen. Dieses würden wir anschließend in der Tabelle **MSysResources** der Zieldatenbank speichern und für die Schaltfläche verwenden.

Praktischer wäre es jedoch, wenn wir die einmal verwendeten Icons direkt auch in unserer Lösung speichern und für die weitere Verwendung direkt zur Auswahl anbieten – zum Beispiel, wenn wir ein Add-In aus dieser Lösung

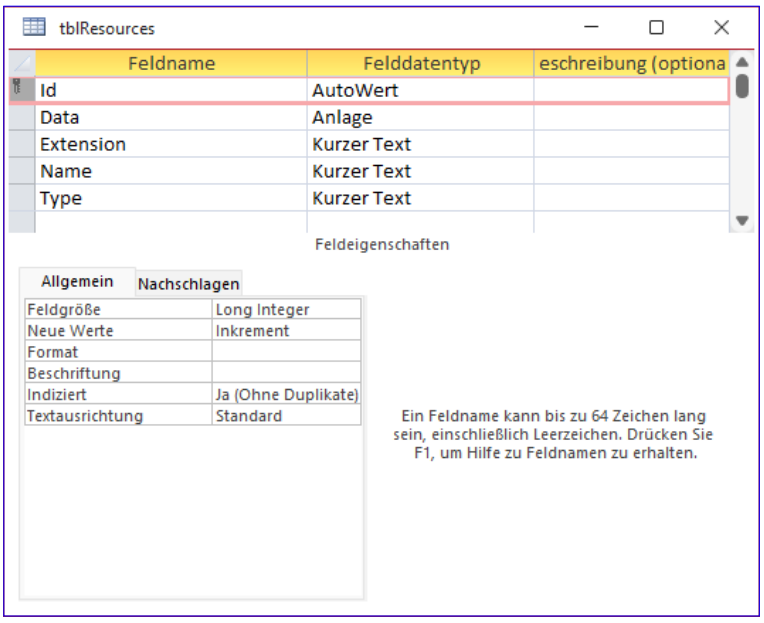


Bild 5: Tabelle zum Speichern der Icons

erstellen. Dann hätte der Benutzer einen Schritt weniger bei bereits verwendeten Icons und könnte diese direkt auswählen.

Also legen wir dazu eine neue Tabelle namens **tblResources** an, die genauso aufgebaut ist wie die Tabelle **MSysResources**. Genau genommen haben wir diese einfach kopiert und unter dem Namen **tblResources** wieder eingefügt. Den Entwurf dieser Tabelle sehen Sie in Bild 5.

Damit kommen wir zur Schaltfläche **cmdBilderAuswaehlen**, die im Entwurf aus Bild 6 markiert ist.

Diese soll einen **Dateiauswahl**-Dialog anzeigen und die Bilder anschließend in der Tabelle **tblResources** speichern. Dazu deklarieren wir eine Konstante, welche die dabei zu verwendende Tabelle enthält:

```
Private Const cStrResourcesTable As String _
    = "tblResources"
```

Die Prozedur **cmdBilderAuswaehlen** ruft die Funktion **ChooseFiles** auf, die im Modul **mdlFileDialog** enthalten ist und den Standard-Office-Dialog zum Auswählen von Dateien verwendet. Sie speichert die ausgewählten Dateipfade in der Variablen **strFileList**. Wenn diese nicht leer ist, unterteilt sie diese Liste jeweils am Pipe-Zeichen (**|**) und schreibt die einzelnen Pfade in ein Array namens **strFiles**.

Für jede Datei ruft sie einmal die Routine **BildInRessourcenSpeichern** auf und übergibt dieser den Dateipfad und den Namen der Tabelle, in der das Icon gespeichert werden soll.

Nachdem alle Icons gespeichert sind, ruft sie die Prozedur **LoadImages** auf, die dafür sorgt, dass die aktuell in der Tabelle **tblResources** gespeicherten Icons in der Matrix im Formular angezeigt werden.

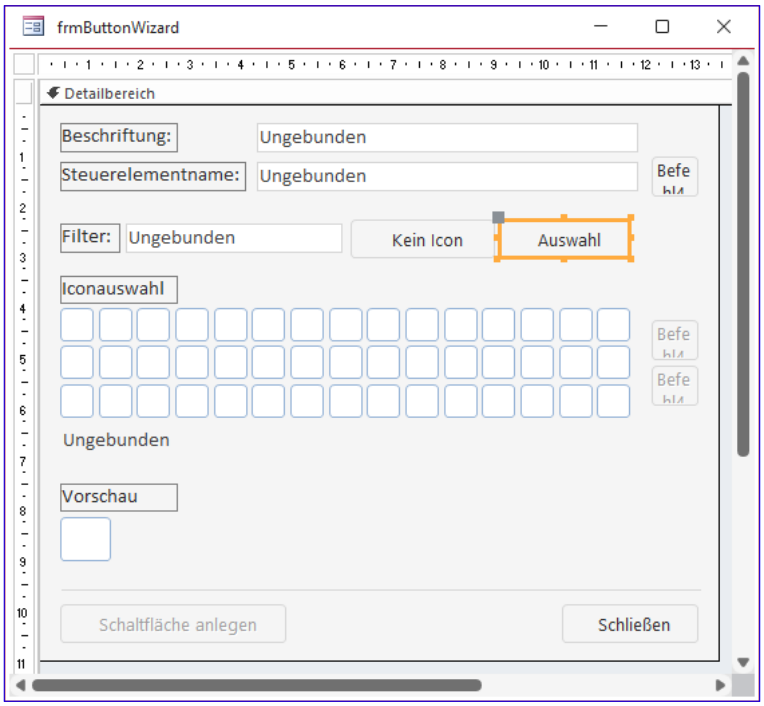


Bild 6: Vollständiger Entwurf des Formulars **frmButtonWizard**

```
Private Sub cmdBilderAuswaehlen_Click()
    Dim strFilelist As String
    Dim strFiles() As String
    Dim var As Variant

    Me.txtFilter.SetFocus
    strFilelist = ChooseFiles

    If Len(strFilelist) > 0 Then
        strFiles = Split(strFilelist, "|")
        For Each var In strFiles
            BildInRessourcenSpeichern CStr(var), _
                cStrResourcesTable
        Next var
        Call LoadImages
    End If

End Sub
```

Wie die Routine **LoadImages** funktioniert, erläutern wir im Detail im Beitrag **Icons in der Datenbank verwalten** (**www.access-im-unternehmen.de/1564**).

Und wie wir die Matrix aus Bild-Steuerelementen erzeugen, die Sie im Formularentwurf sehen, beschreiben wir im Beitrag **Schaltflächen-Matrix per VBA erzeugen** (**www.access-im-unternehmen.de/1562**).

Vorschau der Schaltfläche anzeigen

Damit haben wir nun die Textfelder, in denen die Beschriftung und der Steuerelementname gespeichert sind. Außerdem haben wir eine Funktion, mit der wir die als Icons zu verwendenden Bilder in die Tabelle **tblResources** laden können und eine Matrix aus Bild-Steuerelementen, mit denen wir die verfügbaren Icons anzeigen. Außerdem haben im oben angegebenen Artikel beschrieben, wie wir zwischen den Bildern navigieren können, wenn die Tabelle **tblResources** mehr Bilder enthält, als auf einen Blick angezeigt werden können.

Damit kommen wir zur Anzeige der Vorschau für die zu erstellende Schaltfläche. Diese soll immer aktualisiert werden, wenn wir eine Änderung an der Beschriftung oder dem gewählten Icon vornehmen.

Die dadurch ausgelöste Prozedur heißt **VorschauAktualisieren** (siehe Listing 3). Sie referenziert die als Beispielschaltfläche angelegte Schaltfläche namens **cmdVorschau**. Dann holt sie mit der Funktion **Schriftabmessungen** die Breite und die Höhe für die anzuzeigende Beschriftung und liefert diese mit den beiden Parametern **lngBreite** und **lngHoehe** zurück (diese Funktion beschreiben wir im Anschluss).

Als Höhe des Steuerelements legen wir 480 Pixel fest. Mit **Me.Painting = False** verhindern wir während des Vorgangs das Neuzeichnen des Formulars, damit es nicht zu unnötigem Flackern kommt.

Dann lesen wir aus der **TempVar**-Variablen **Picture** den zuvor zwischengespeicherten Namen des zu verwendenden Icons aus. Hat diese Zeichenkette die Länge **0**, leeren wir die Eigenschaft **PictureData** der Schaltfläche, tragen die Beschriftung ein, legen die Breite auf **480** plus der ermittelten Breite für die Beschriftung aus **lngWidth** fest und stellen die Ausrichtung auf linksbündig ein. Schließlich legen wir für die Variable **bolAktivieren** den Wert **True** fest. Dieser wird später ausgewertet, um die Schaltfläche zum Anlegen der neuen Schaltfläche zu aktivieren oder zu deaktivieren.

Wenn die **TempVar**-Variable gefüllt ist, gelangen wir in den **Else**-Zweig der Bedingung. Hier rufen wir die Funktion **GetPictureDataFromResourcesByName** auf, die wir im Modul **mdlPictureData** finden. Dieser Funktion übergeben wir den Namen des einzulesenden Icons, die Tabelle, in der sich das Icon befindet (hier **tblResources**) und eine leere **Variant**-Variable, die wir mit dem zu verwendenden Bild-Objekt füllen. Liefert die Funktion den Wert **True** als Ergebnis zurück, konnte das Bild erfolgreich eingelesen werden.

Dann weisen wir den Inhalt von **varPictureData** der Eigenschaft **PictureData** der Vorschau-Schaltfläche zu und stellen **PictureType** auf **2** ein.

Enthält die **TempVar**-Variable **Beschriftung** einen Text, wird dieser für die Eigenschaft **Caption** als Beschriftung eingetragen. Außerdem wird die Breite auf 600 Pixel plus der zuvor ermittelten Breite der Beschriftung eingestellt und **bolAktivieren** erhält den Wert **True**.

bolAktivieren hat anschließend nur den Wert **False**, wenn weder eine Bildbeschriftung vorliegt noch ein Bild ausgewählt wurde.

Danach aktivieren wir mit **Me.Painting = True** das Neuzeichnen des Formulars wieder und aktualisieren die Anzeige der Vorschau-Schaltfläche.

Danach prüfen wir den Wert von **bolAktivieren**. Hat diese Variable den Wert **True**, aktivieren wir die Schaltfläche **cmdSchaltflaecheAnlegen**, andernfalls deaktivieren wir diese. Damit stellen wir sicher, dass die Schaltfläche nicht angelegt werden kann, wenn noch Informationen fehlen.

Form und Subform in der Datenblattansicht per Wizard

Eine immer wiederkehrende Aufgabe in meinem Programmieralltag ist das Erstellen von Formularen, die ein Unterformular mit Daten in der Datenblattansicht enthalten. Das umfasst einige Schritte, die ich immer wieder manuell durchgeführt habe. Bis ich die Lösung für diesen Beitrag programmiert habe. Einen Assistenten, den ich starte, statt im Ribbon den Befehl zum Erstellen eines neuen, leeren Formulars in der Entwurfsansicht aufzurufen. Und der lediglich eine Information benötigt: Welche Tabelle oder Abfrage soll die Daten für das Unterformular bereitstellen? Das Ganze in einem Assistenten verpackt, der in jeder geöffneten Access-Datenbank per Mausklick gestartet werden kann. Nachfolgend finden Sie die Anleitung, wie ich diesen Assistenten erstellt habe und wie er funktioniert.

Die Anzeige der Daten einer Tabelle oder Abfrage in einem Unterformular wird oft benötigt. Das Unterformular zeigt die Daten in der Datenblattansicht an. Im Hauptformular kann man beliebige Steuerelemente zum Bearbeiten oder Hinzufügen von Datensätzen, zum Anzeigen eines Detailformulars zum ausgewählten Datensatz, zum Löschen von Datensätzen oder zum Filtern oder Sortieren der Datensätze anzeigen.

Die Anzeige der Daten in der Datenblattansicht ermöglicht außerdem die Nutzung der Filter- und Sortiermöglichkeiten dieser Ansicht. Zudem können wir Spalten anordnen oder ein- und ausblenden.

Das Anlegen einer solchen Kombination aus Haupt- und Unterformular erfordert einige Schritte, die wir nachfolgend auflisten:

- Anlegen des Unterformulars
- Zuweisen der Datensatzquelle für das Formular
- Hinzufügen der anzuzeigenden Felder
- Einstellen der Eigenschaft **Standardansicht** auf **Datenblatt**
- Speichern des Unterformulars unter dem gewünschten Namen
- Anlegen des Hauptformulars
- Einfügen des Unterformulars

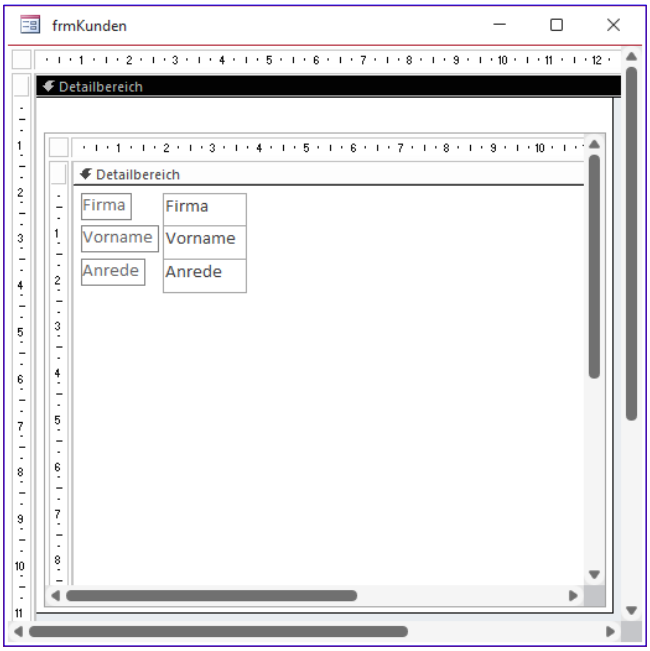


Bild 1: Formular mit Unterformular in der Datenblattansicht

- Einstellen weiterer Eigenschaften des Hauptformulars, zum Beispiel **Automatisch zentrieren** auf **Ja**, **Bildlaufleisten** auf **Keine**, **Datensatzmarkierer** und **Navigationsschaltflächen** auf **Nein**.

- Einstellen der Verankerung des Unterformular-Steuerelements, sodass sich seine Größe beim Vergrößern des Hauptformulars ebenfalls ändert

- Speichern des Hauptformulars unter dem gewünschten Namen

Diese Schritte habe ich in der Vergangenheit hunderte, wenn nicht tausende Male durchgeführt. Also beschloss ich, einen Wizard zu programmieren, der mit diese immer wiederkehrenden Schritte abnimmt und ein Formular wie das aus Bild 1 schnell erstellt.

Damit ist eine Basis geschaffen, der ich dann die weiteren Steuerelemente hinzufügen kann.

Funktionsweise des Wizards

Der Wizard wird einfach über den Eintrag **Formular-Assistent** im Ribbon aufgerufen. Dies zeigt alle verfügbaren Assistenten an (siehe Bild 2).

Hier wählen wir unseren Assistenten aus und selektieren die Datensatzquelle für das zu erstellende Formular. Diese ist bereits voreingestellt, wenn beim Aufrufen des Assistenten schon eine Tabelle oder Abfrage im Navigationsbereich ausgewählt war.

Danach erscheint unser selbst programmiertes Formular, mit dem wir weitere Details festlegen können (siehe Bild 3).

Dabei handelt es sich um die folgenden Informationen:

- **Anzuzeigende Felder**: Hier können wir die Felder festlegen, die in das Unterformular in der Datenblattansicht übernommen werden sollen.

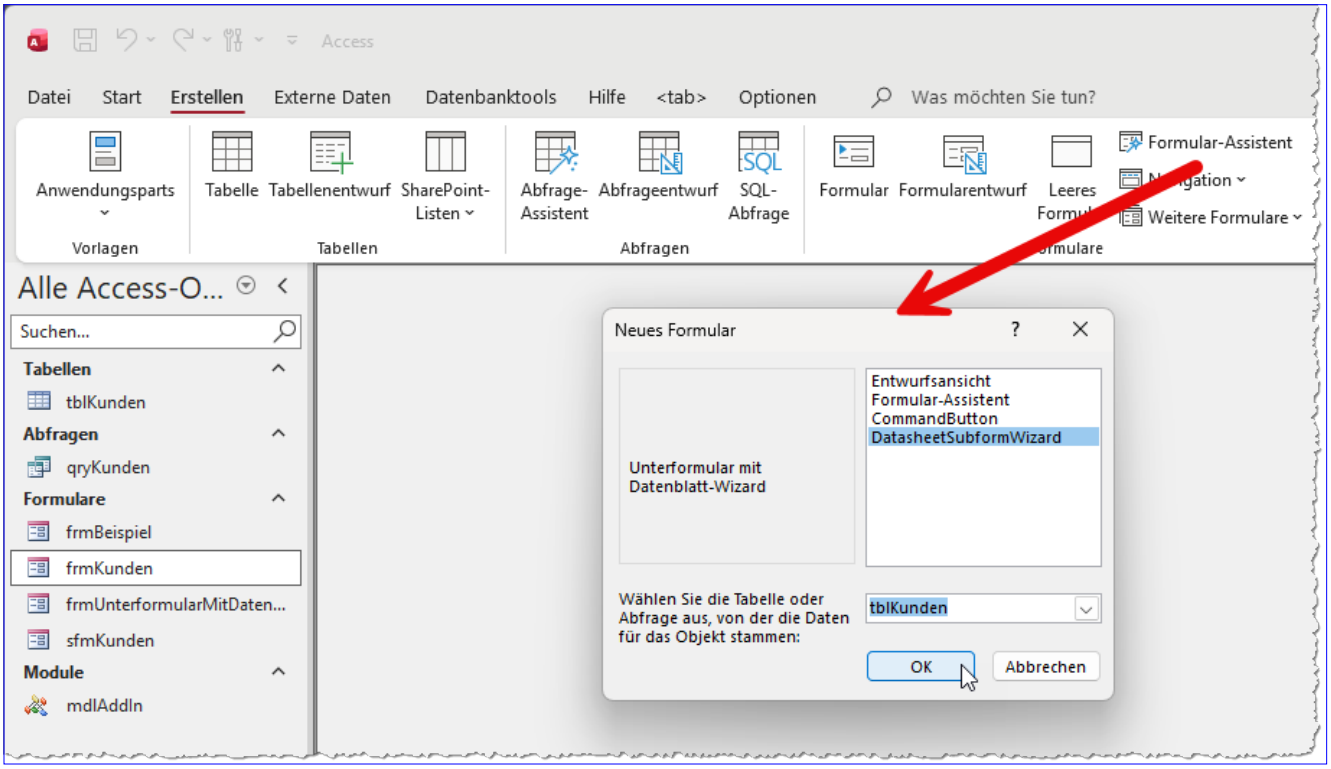


Bild 2: Unser Assistent in der Liste der Formular-Assistenten

- **Name des Hauptformulars**: Hier geben wir den Namen des zu erstellenden Hauptformulars ein, für den bereits ein Vorschlag vorgegeben wird.

- **Name des Unterformulars**: Hier stellen wir den Namen des Unterformulars ein. Auch hier finden wir einen Vorschlag.

- **Vorhandene Formulare überschreiben**: Legt fest, dass eventuell vorhandene Formulare gleichen Namens überschrieben werden sollen.

- Schaltfläche **Formular erstellen**: Startet das Erstellen des Haupt- und Unterformulars.

- Schaltfläche **Abbrechen**: Schließt das Formular, ohne neue Formulare zu erstellen.

Wenn wir nun auf **Formular erstellen** klicken, arbeitet Access kurz und präsentiert anschließend das neue Formular (siehe Bild 4).

Starten wir den Assistenten danach erneut für die gleiche Datensatzquelle, zeigt dieser direkt an, dass die automatisch vorgeschlagenen Formularnamen bereits existieren (siehe Bild 5).

In diesem Fall können wir den Namen entweder anpassen oder wir aktivieren die Option **Vorhandene Formulare überschreiben**.

Erst wenn eines von beiden erfolgt ist, wird die Schaltfläche zum Anlegen der Formulare aktiviert.

Programmierung des Assistenten

Damit die Datenbank mit unserer Lösung als Add-In installiert werden kann, benötigen wir

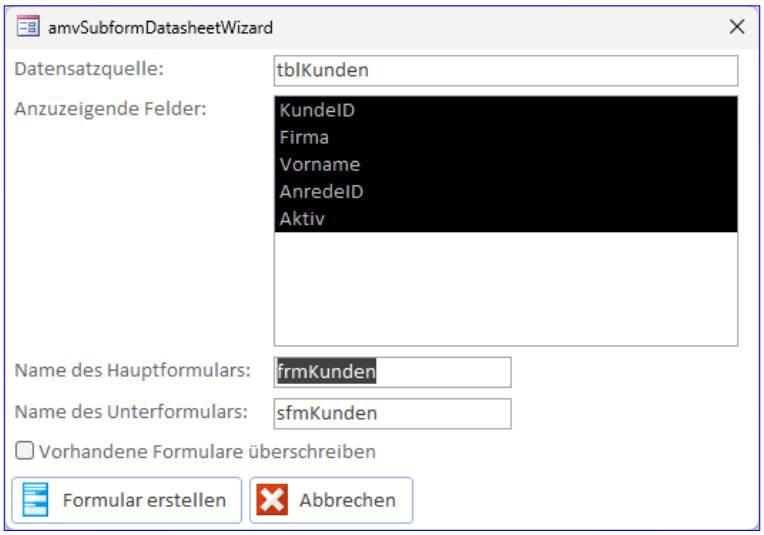


Bild 3: Der Wizard zum Erstellen von Unterformularen in der Datenblattansicht

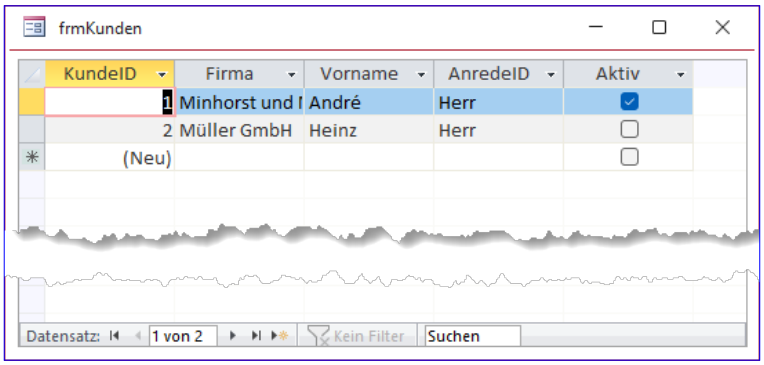


Bild 4: Das neu erstellte Formular mit Unterformular

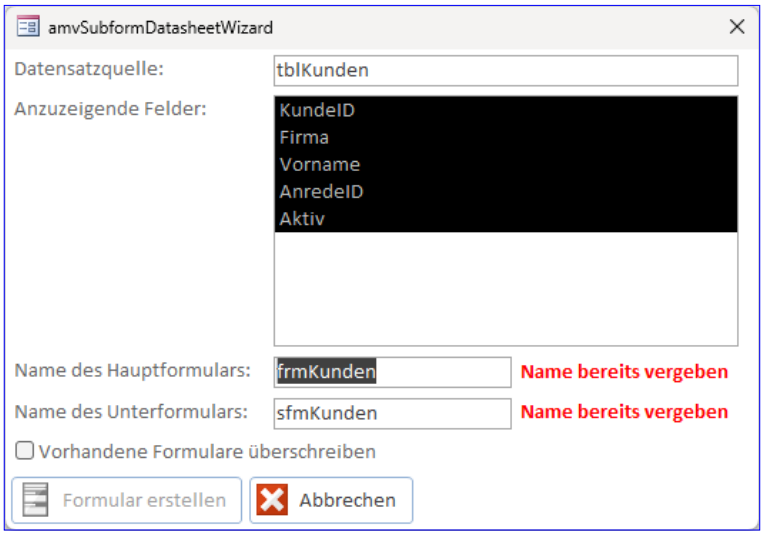


Bild 5: Hinweis auf bereits vorhandene Formulare

eine Tabelle namens **USysRegInfo**, die wir wie in Bild 6 füllen.

Diese enthält die Informationen, die beim Installieren in die Registry eingetragen werden.

Wichtig ist der Name der aufzurufenden Funktion beim Starten des Assistenten, hier **Autostart_SubformWithDatasheetWizard**.

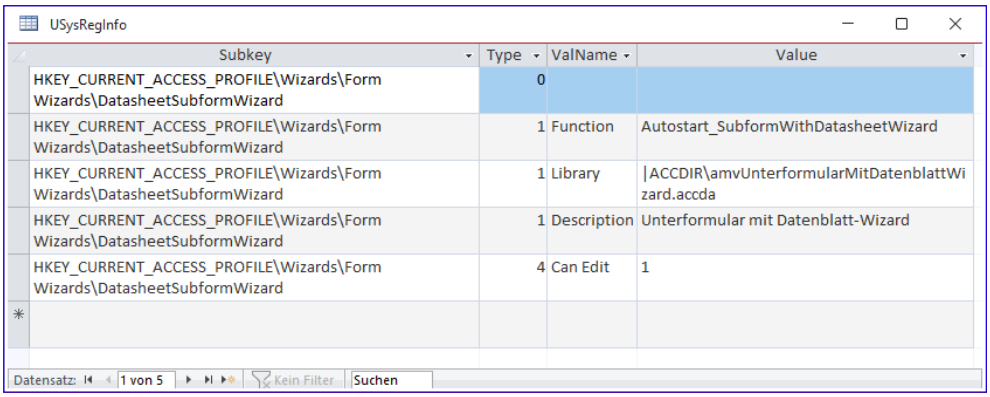


Bild 6: Die Tabelle **USysRegInfo**

Sie hat einen Parameter namens **strRecordsource**, der automatisch mit der Tabelle oder Abfrage gefüllt wird, die wir im Dialog zur Auswahl des Assistenten auswählen.

Diese Funktion sehen wir in Listing 1. Sie ruft lediglich eine weitere Prozedur namens **CreateSubformWithDatasheet** auf, der sie den Namen der Datensatzquelle übergibt.

Hauptfunktion des Wizards

Die Hauptfunktion **CreateSubformWithDatasheet** finden wir in Listing 2. Sie schreibt den Namen des Wizard-Formulars in die Variable **strWizForm** und schließt dieses zunächst, falls es noch geöffnet ist.

Dann öffnet sie das Formular als modalen Dialog und übergibt die zu verwendende Datensatzquelle mit dem Parameter **OpenArgs**. Im Formular gibt der Benutzer nun die gewünschten Daten ein und macht dieses entweder mit der Schaltfläche **Formular erstellen** unsichtbar oder schließt es mit der **Abbrechen**-Schaltfläche.

Deshalb prüfen wir im folgenden Schritt mit der Hilfsfunktion **IstFormularGeoeffnet**, ob das Formular ausgeblendet oder geschlossen wurde.

Falls das Formular noch geöffnet, aber unsichtbar ist, liest die Prozedur den Namen des zu erstellenden Formulars in **strNewForm** und den des Unterformulars in **strNewSubform** ein. Außerdem liest sie die im Listenfeld **lstFields** ausgewählten Felder mit der Funktion **GetSelectedFields** in die Variable **strFields** ein.

Diese Funktion durchläuft alle selektierten Einträge aus der Eigenschaft **ItemsSelected** und fügt den jeweiligen Eintrag zur Variablen **strFields** hinzu. Die Einträge haben jeweils ein vorangestelltes Semikolon. Nach dem Durch-

```
Public Function Autostart_SubformWithDatasheetWizard(Optional strRecordsource As String) As Variant
    On Error GoTo Fehler
    Call CreateSubformWithDatasheet(strRecordsource)
    Exit Function
Fehler:
    MsgBox Err.Number & " " & Err.Description
End Function
```

Listing 1: Funktion, die beim Start des Wizards aufgerufen wird

laufen aller Felder wird noch ein abschließendes Semikolon hinzugefügt und das Ergebnis zurückgegeben:

```
Public Function GetSelectedFields(lst As ListBox) As String
    Dim var As Variant
    Dim i As Integer
    Dim strFields As String
    For Each var In lst.ItemsSelected
        strFields = strFields & ";" & lst.ItemData(var)
        i = i + 1
    Next var
    strFields = strFields & ";"
    GetSelectedFields = strFields
End Function
```

Das Ergebnis lautet beispielsweise wie folgt:

```
;KundeID;Firma;Vorname;AnredeID;Aktiv;
```

Außerdem lesen wir den Wert des Kontrollkästchens **chkOverwriteForms** in die Variable **bolOverwriteForms** ein. Damit können wir das Formular nun endgültig schließen.

Nun erstellen wir mit der Funktion **CreateSubForm** zunächst das Unterformular und fügen diesem die ausgewählten Felder als gebundene Steuerelemente mit Beschriftungsfeld hinzu.

Diese Funktion sehen wir uns im Anschluss an.

Unterformular umbenennen

Die Funktion liefert den Namen des neu erstellten Unterformulars zurück und wir speichern es in **strTempForm**.

Dann benennen wir diese mit der Funktion **RenameForm** in den durch den Benutzer eingegebenen Namen um.

Diese Funktion nimmt den neuen und den alten Namen entgegen und außerdem den **Boolean**-Wert, der angibt, ob bestehende Formulare gelöscht werden sollen.

Wenn ein bestehendes Formular überschrieben werden soll, schließen wir dieses, falls es noch geöffnet ist, und löschen es anschließend mit **DoCmd.Delete**.

Anschließend benennen wir das neu erstellte Formular auf den angegebenen Namen um und liefern den Wert **True** zurück:

```
Public Function RenameForm(strNew As String, strOld As _
        String, bolOverwriteForms As Boolean) As Boolean
    On Error GoTo Fehler
    If bolOverwriteForms = True Then
        On Error Resume Next
        DoCmd.Close acForm, strNew
        On Error GoTo Fehler
        DoCmd.DeleteObject acForm, strNew
    End If

    DoCmd.Rename strNew, acForm, strOld
    RenameForm = True
    Exit Function
Fehler:
    MsgBox "Fehler " & Err.Number & vbCrLf & Err.Description
End Function
```

War dies nicht erfolgreich, löschen wir das angelegte Formular wieder und verlassen die Prozedur.

Hauptformular anlegen

War dies erfolgreich, erstellen wir das Hauptformular mit der Funktion **CreateForm** und übergeben dieser den Namen des zu erstellenden Formulars.

Auch diese Funktion beschreiben wir im Anschluss.

Für das neu erstellte Hauptformular rufen wir wiederum die Funktion **RenameForm** auf und übergeben den neuen und den alten Namen, damit das Formular den gewünschten Namen erhält. Wenn dies fehlschlägt, gibt die Funktion einen entsprechenden Hinweis auf. Das Formular wird gelöscht und die Prozedur beendet.

```
Private Sub CreateSubformWithDatasheet(strRecordsource As String)
    Dim db As DAO.Database
    Dim strNewForm As String
    Dim strNewSubform As String
    Dim bolOverwriteForms As Boolean
    Dim strWizForm As String
    Dim strTempForm As String
    Dim strFields As String
    strWizForm = "frmUnterformularMitDatenblatt"
    On Error Resume Next
    DoCmd.Close acForm, strWizForm
    On Error GoTo Fehler
    DoCmd.OpenForm strWizForm, WindowMode:=acDialog, OpenArgs:=strRecordsource
    If IstFormularGeoeffnet(strWizForm) Then
        strNewForm = Forms(strWizForm)!txtForm
        strNewSubform = Forms(strWizForm)!txtSubform
        strFields = GetSelectedFields(Forms(strWizForm)!lstFields)
        bolOverwriteForms = Forms(strWizForm)!chkOverwriteForms
        DoCmd.Close acForm, strWizForm
    Else
        Exit Sub
    End If
    On Error GoTo 0
    Set db = CurrentDb
    strTempForm = CreateSubform(db, strRecordsource, strFields)
    If RenameForm(strNewSubform, strTempForm, bolOverwriteForms) = True Then
        strTempForm = CreateForm(strNewSubform)
        If RenameForm(strNewForm, strTempForm, bolOverwriteForms) = False Then
            MsgBox "Formular '" & strNewForm & "' konnte nicht erstellt werden.", vbExclamation + vbOKOnly, _
                "Fehler bei Erstellung"
            DoCmd.DeleteObject acForm, strTempForm
            Exit Sub
        End If
    Else
        MsgBox "Unterformular '" & strNewSubform & "' konnte nicht erstellt werden.", vbExclamation + vbOKOnly, _
            "Fehler bei Erstellung"
        DoCmd.DeleteObject acForm, strTempForm
        Exit Sub
    End If
    DoCmd.OpenForm strNewForm
    Exit Sub
Fehler:
    Select Case Err.Number
        Case 0, 2501
        Case Else
            MsgBox "Fehler " & Err.Number & vbCrLf & Err.Description
    End Select
End Sub
```

Listing 2: Hauptfunktion des Wizards