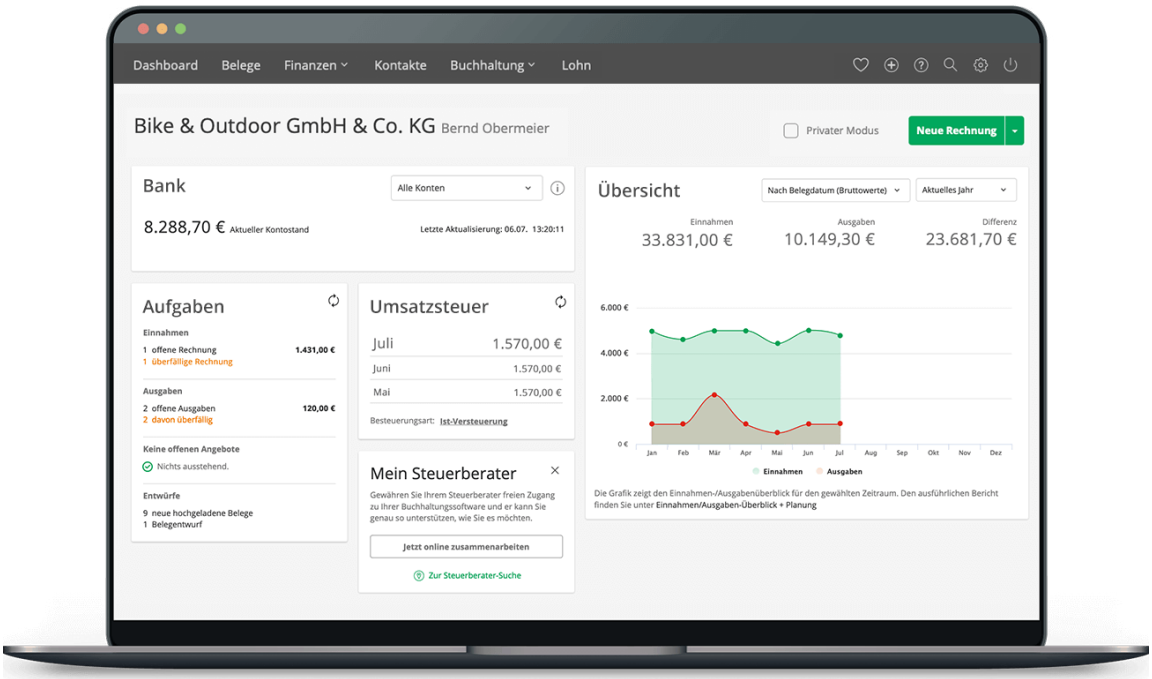
HAUFE.
Ausgabe 02/2023

ACCESS IM UNTERNEHMEN

LEXOFFICE PER VBA PROGRAMMIEREN

Sie möchten Ihre Kundenverwaltung unter Access und die Buchhaltung mit lexoffice kombinieren? Ab Seite 39 liefern wir die Grundlagen dazu!



In diesem Heft:

ALTERNATIVE MSGBOX

Fügen Sie Ihren Meldungen mehr Struktur hinzu – mit einer fett gesetzten Überschrift.

SEITE 20

STEUERELEMENTE AUSRICHTEN

Genervt davon, Steuerelemente immer manuell auszurichten? Lernen Sie, wie das per Mausklick gelingt!

SEITE 22

FELDBESCHRIFTUNGEN OPTIMAL NUTZEN

Erfahren Sie, wie Sie die Verwendung der Beschriftung von Tabellenfeldern optimieren können!

SEITE 6

lexoffice und Access

lexoffice ist eines der am meisten genutzten Tools für die Buchhaltung. Es erlaubt die Verwaltung von Kontakten, Kontoumsätzen und Rechnungen sowie den Abgleich von gestellten Rechnungen und Rechnungseingängen. Außerdem lassen sich damit die Daten für den Steuerberater vorbereiten. Was hat das mit Microsoft Access zu tun? Viele Unternehmen nutzen nutzen lexoffice und führen vorbereitende oder sich überschneidende Aufgaben mit Access aus. Seit kurzem bietet lexoffice allerdings eine offene Rest API an – also eine Schnittstelle, mit der man von außen auf die mit lexoffice verwalteten Daten zugreifen kann. Grund genug für uns, diese Möglichkeit genauer anzuschauen!

lexoffice bietet zum Beispiel die Möglichkeit, Kundendaten oder Rechnungsdaten aus einer Access-Anwendung heraus direkt als Kontakte oder Rechnungen in lexoffice anzulegen. Wenn Sie also eine Bestellverwaltung unter Access nutzen und bisher die Rechnungen als Bericht erstellt, per E-Mail an den Kunden geschickt und dann den Eingang der Zahlung selbst überwacht haben, können Sie eine Menge dieser Schritte an lexoffice auslagern – und erhalten gleichzeitig noch zusätzliche Funktionen wie die Vorbereitung der Daten für den Steuerberater. An dieser Stelle bereits der Hinweis, dass lexoffice kostenpflichtig ist! Mehr zu diesem Thema erfahren Sie im Beitrag **Zugriff auf lexoffice per REST-API und VBA** ab Seite 39.

Microsoft hat die Anzeige von Sicherheitswarnungen in Access geändert, sodass man eine Anwendung unbekannter Herkunft nun nicht mehr einfach per Mausklick freischalten kann. Denen neuen Weg zur Freischaltung zeigen wir unter dem Titel **Sicherheitswarnungen in Access** ab Seite 2.

Die Werte, die wir der Eigenschaft **Beschriftung** von Tabellenfeldern hinzufügen, werden nicht nur als Spaltenüberschriften in der Datenblattansicht angezeigt, sondern auch beim Erstellen von Steuerelementen auf Basis dieser Felder in Formularen und Berichten verwendet. Meist passt man immer die gleichen Beschriftungen an, um Feldnamen wie **Strasse**, **EMail** oder **AnredeID** als **Straße**, **E-Mail** oder **Anrede** anzuzeigen. Wie wir das automatisieren können und warum und wie wir diese Beschriftungen zwischenzeitlich entfernen, beschreiben wir im Beitrag **Tabellenfeldbeschriftungen im Griff** ab Seite 6.

Die Standard-Meldungsfenster von VBA können nur unformatierte Texte anzeigen. Wenn Sie auch einmal eine fett gesetzte Überschrift innerhalb einer Meldung präsentieren wollen, erfahren Sie ab Seite 20 im Beitrag **Alternative MsgBox mit Überschrift** mehr.

Das Arrangieren von Steuerelementen ist eine Fleißarbeit. Schon die eingebauten Funktionen bieten einige Erleichterungen, die wir im Beitrag **Steuerelemente ausrichten und anpassen** ab Seite 22 präsentieren.

Einen Schritt weiter gehen wir im Beitrag **Steuerelemente ausrichten per VBA** ab Seite 28. Hier stellen wir Techniken vor, mit denen Sie beispielsweise untereinander angeordnete Steuerelemente per Aufruf einer VBA-Routine schnell anordnen und anpassen können.

Im Beitrag **Formatassistent für Textfelder, Teil 2** fügen wir ab Seite 57 Funktionen zum Formatassistenten aus der vorherigen Ausgabe hinzu.

Und unter **Access-Add-In per Knopfdruck erstellen** stellen wir ab Seite 66 ein Add-In vor, mit dem Sie schnell bestehende Datenbanken in Add-Ins umwandeln können.

Jetzt aber viel Spaß beim Lesen und Ausprobieren!

Ihr André Minhorst

Sicherheitswarnungen in Access

Nach einigen Jahren der Konstanz hat sich Microsoft mal wieder etwas Neues ausgedacht, um für mehr Sicherheit für die Anwender zu sorgen. Bisher erschien bei Datenbanken aus unbekannten Quellen beim Öffnen ein gelber Balken, der direkt per Mausklick auf eine Schaltfläche die Möglichkeit bot, die Datenbank als sicher einzustufen und sofort auf diese zuzugreifen. Das ist bei sonst gleichen Bedingungen nun nicht mehr möglich und es ist ein zusätzlicher Schritt erforderlich. Welcher das ist und wie Sie die Bedingungen verändern können, um direkt auch auf unbekannte Datenbanken zuzugreifen, zeigen wir in diesem Beitrag.

Wenn Sie bis vor nicht allzu langer Zeit eine Datenbank beispielsweise von einem Kunden per E-Mail erhalten haben und diese auf dem eigenen Rechner öffnen wollten, haben Sie normalerweise die Meldung aus Bild 1 erhalten. Mit einem Mausklick auf die Schaltfläche **Inhalt aktivieren** war das Problem üblicherweise erledigt – die Anwendung war danach sofort einsatzbereit.

Mit einem Klick auf diese Schaltfläche wurde die Datenbankdatei der Liste der vertrauenswürdigen Datenbanken hinzugefügt, die in der Registry gespeichert wird.

Seit Januar 2023 stellen wir allerdings fest, dass dieses Verhalten der Vergangenheit angehört: Wenn wir nun eine Datenbankdatei beispielsweise per E-Mail erhalten und diese nach dem Speichern in einem normalen Verzeichnis öffnen, erhalten wir die Meldung aus Bild 2.

Diese enthält im Vergleich zu der vorherigen Meldung vor allem keine Schaltfläche mehr, mit der man den enthaltenen VBA-Code aktivieren kann.

Datenbankzugriff erlauben

Zum Aufheben dieser Beschränkung gibt es zwei Möglichkeiten:

- Speichern der Datenbank in einem vertrauenswürdigen Verzeichnis
- Anpassen der Dateieigenschaften der Datenbank

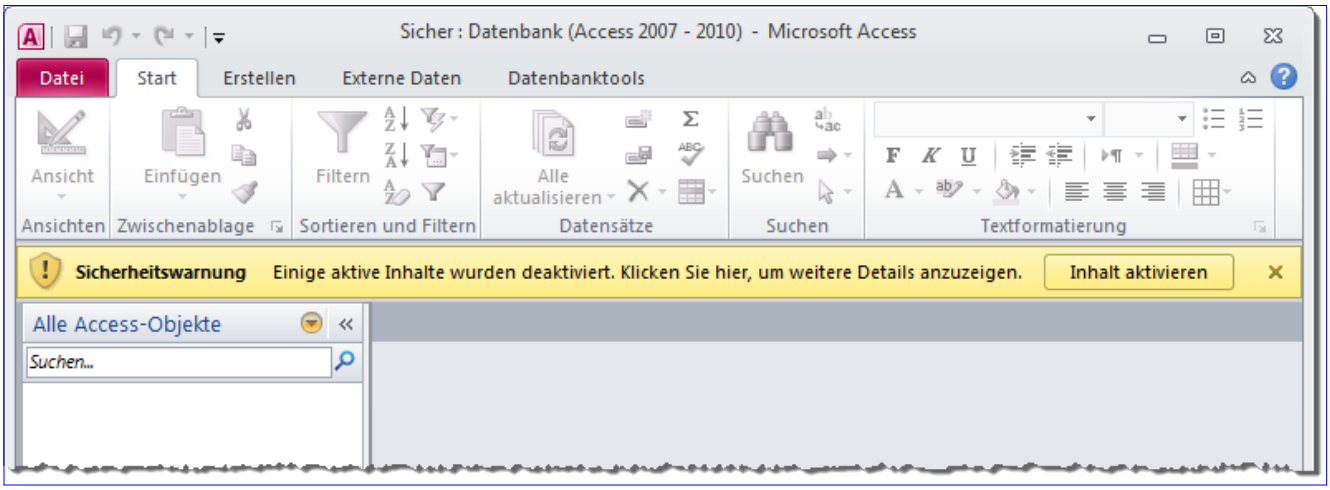


Bild 1: Sicherheitswarnung in früheren Versionen von Access

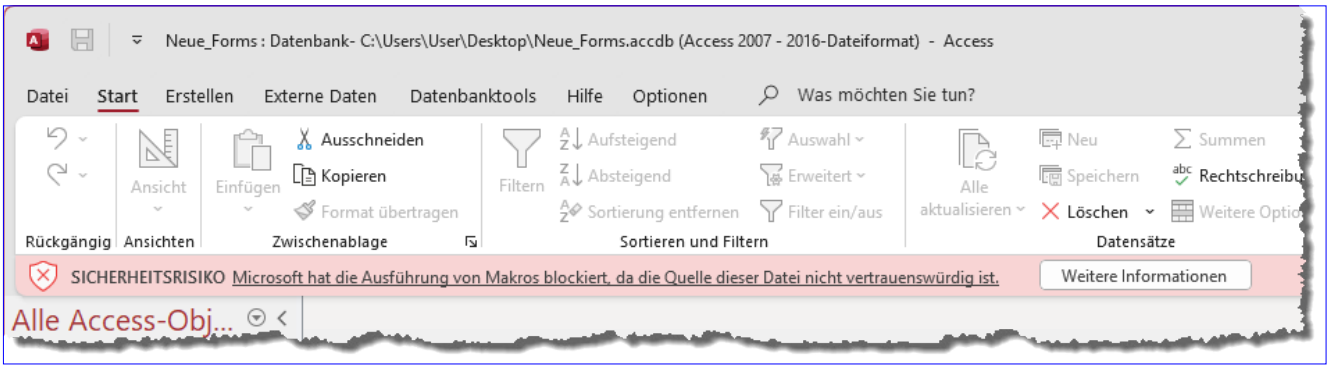


Bild 2: Sicherheitswarnung seit Januar 2023

Der erste Schritt ist den meisten Lesern vermutlich bekannt: Ist ein vertrauenswürdiges Verzeichnis bereits vorhanden, kopiert man die Datenbank dorthin und kann sie wie üblich öffnen.

Hat man noch kein entsprechendes Verzeichnis, öffnet man mit dem Ribbon-Befehl **DateiOptionen** den **Optionen**-Dialog. Dort wechseln wir zum Bereich **Trust Center** und öffnen mit einem Klick auf **Einstellungen für das Trust Center...** den Dialog **Trust Center**. Im dortigen Bereich **Vertrauenswürdige Speicherorte** fügen wir mit einem Klick auf **Neuen Speicherort hinzufügen...** das gewünschte Verzeichnis hinzu (siehe Bild 3).

Verschieben wir eine Datenbank, die zuvor die eingangs erwähnte Meldung angezeigt hat, in ein solches Verzeichnis, erscheint die Meldung nicht mehr. Es reicht also, wenn man ein solches Verzeichnis definiert und Access-Datenbanken dort öffnet.

Vertrauen per Eigenschaft

Die zweite Möglichkeit ist für den Fall vorgesehen, dass wir die Datenbank ohne Verschieben in ein vertrauenswürdiges Verzeichnis öffnen wollen. Da wir die Datenbank nicht mehr per Mausklick als vertrauenswürdig einstufen können, brauchen wir eine Alternative.

Diese sieht wie folgt aus:

- Schließen Sie die Datenbank wieder.
- Zeigen Sie mit einem Rechtsklick auf die Datenbankdatei das Kontextmenü der Datei an.
- Wählen Sie den Eintrag **Eigenschaften** aus.

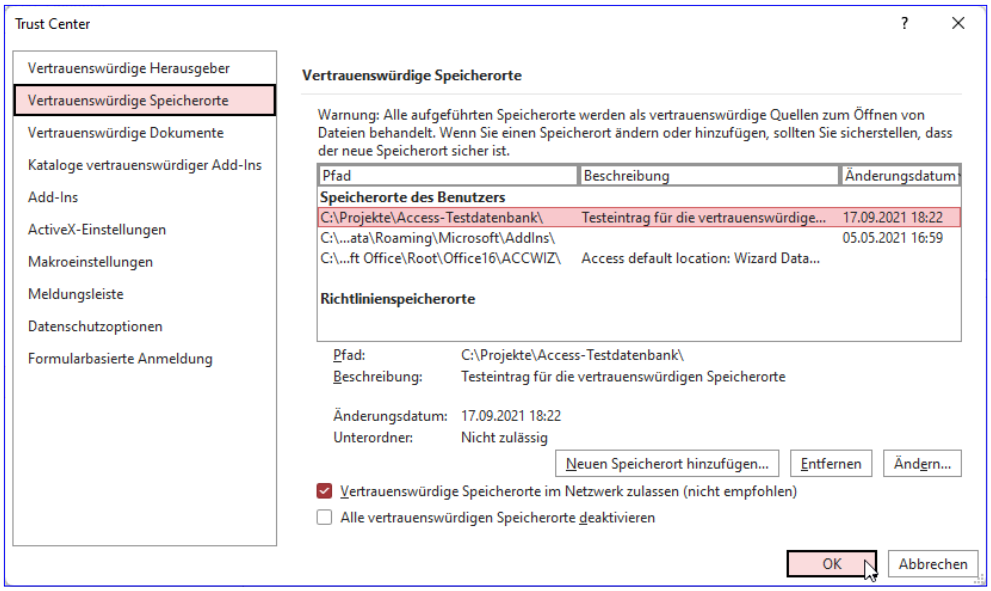


Bild 3: Anlegen eines vertrauenswürdigen Speicherortes

Tabellenfeldbeschriftungen im Griff

Wozu sollte man Tabellenfeldbeschriftungen im Griff haben? Und was ist das überhaupt? Die Eigenschaft »Beschriftung« von Tabellenfeldern kennen viele Entwickler gar nicht. Dabei könnten sie sich eine Menge Zeit sparen, wenn sie diese Eigenschaft nutzen würden. Trägt man nämlich beispielsweise für ein Feld wie »EMail« eine Beschriftung wie »E-Mail« ein, dann wird diese statt des Feldnamens als Spaltenüberschrift der Tabelle in der Datenblattansicht angezeigt. Und es geht noch weiter: Wenn man in Formularen und Berichten Steuerelemente auf Basis dieser Felder anlegt, übernimmt Access auch dort den Wert der Eigenschaft Beschriftung statt des Feldnamens für die Bezeichnungsfelder Steuerelemente. Man braucht als nur an einer Stelle eine Änderung vorzunehmen und profitiert an vielen anderen Stellen davon. Und wenn man nun nicht in jeder neuen Datenbank die Beschriftungen erneut anlegen müsste ... doch auch dafür haben wir in diesem Beitrag eine Lösung.

Eigentlich ist es gar nicht viel Arbeit: Wir erstellen eine neue Tabelle, und für Felder wie **KundeID**, **AnredeID**, **Strasse**, **EMail** et cetera hinterlegen wir für die Eigenschaft **Beschriftung** jeweils den anzuzeigende Text wie **ID/Kunde-ID**, **Anrede**, **Straße** oder **E-Mail**. In Bild 1 sehen Sie, wie wir das Feld **Strasse** entsprechend angepasst haben.

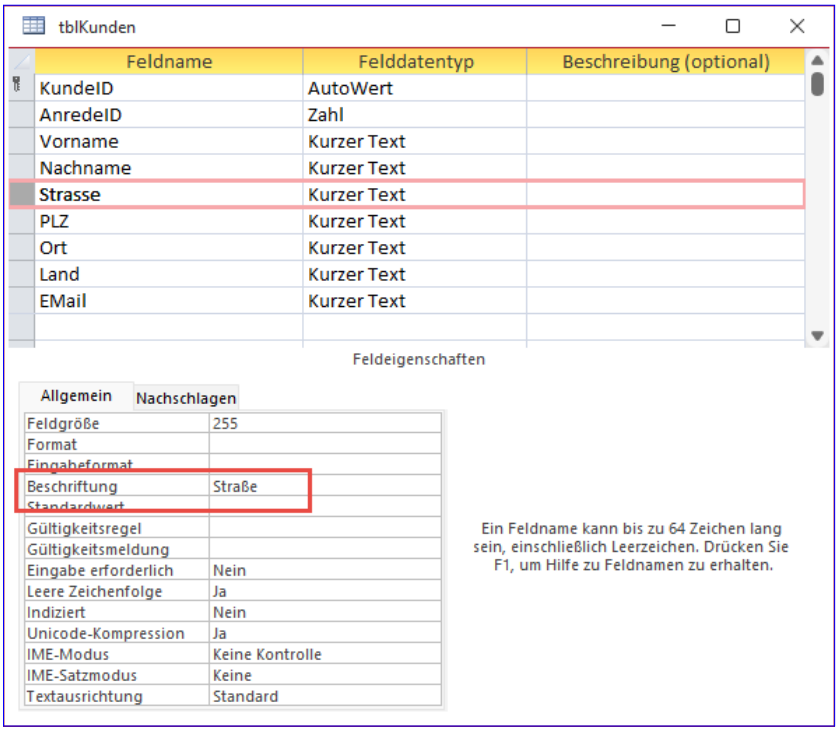


Bild 1: Anpassen der Beschriftung eines Feldes

Wechseln wir in die Datenblattansicht, sehen wir direkt die Werte der Eigenschaft **Beschriftung** der jeweiligen Felder statt der Feldnamen (siehe Bild 2).

Automatisieren der Beschriftungen

Allerdings ist das eine von vielen Aktionen, die man beim Anlegen neuer Tabellen berücksichtigen muss, und die dafür aufzuwendende Zeit summiert sich. Vor allem wenn man bedenkt, dass man diese Aufgabe im Laufe eines Entwickler-

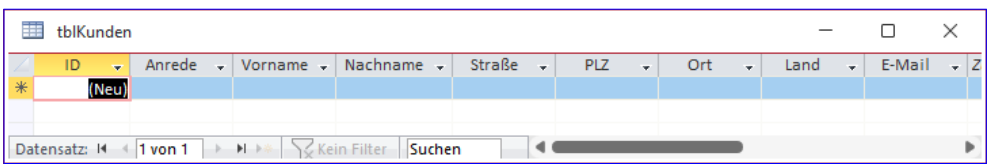


Bild 2: Die Beschriftung wird beispielsweise in der Datenblattansicht angewendet.

lebens sehr oft durchführt. Also sieht es so aus, als ob wir uns über die Automatisierung dieser Aufgabe Gedanken machen sollten!

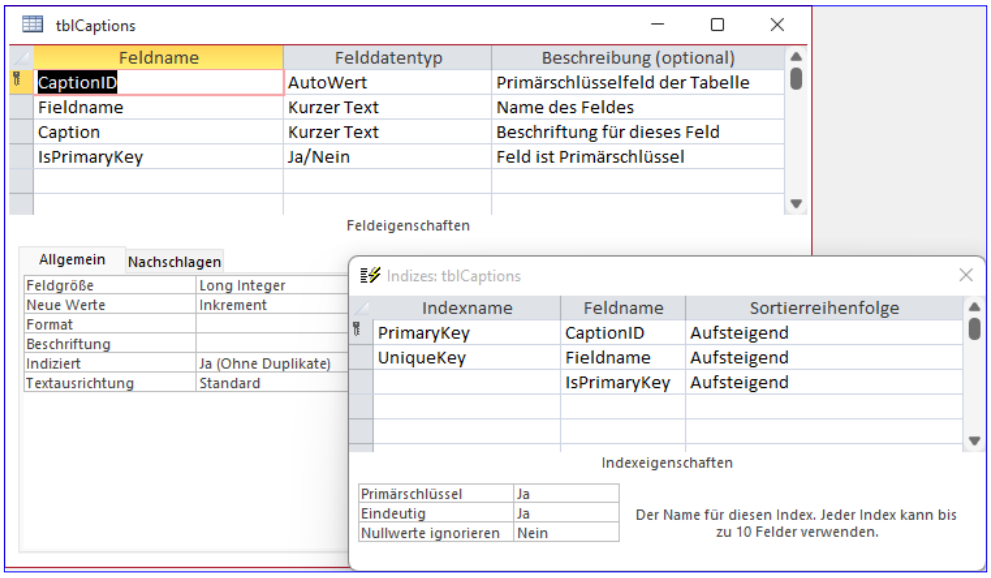


Bild 3: Tabelle zum Speichern der Zuordnung von Feldern und Beschriftungen

Automatisieren der Festlegung von Feldbeschriftungen

Aber lässt sich das überhaupt automatisieren? Ist das nicht ein individueller Vorgang, der mit jeder Datenbank, jeder Tabelle und jedem Feld einzeln durchgeführt und vom Programmierer intensiv überdacht werden muss? Nein, das ist nicht der Fall. Außer, man möchte sich mit ein wenig Routinearbeit entspannen.

Wie also können wir diesen Vorgang automatisieren? Fest steht: Für die meisten Feldnamen gibt es genau eine passende Beschriftung, zumindest wenn man sich einmal auf einen Standard festgelegt hat. Ob zum Beispiel die Inhalte von Primärschlüssel-/Autowertfeldern angezeigt werden müssen, sei dahingestellt, aber in vielen Fällen geschieht das. Dann sollte der Benutzer statt eines Feldnamens wie **KundeID** oder **BestellungID** eine passendere Beschriftung wie **Kunde-ID**/**Bestellung-ID**, **Kundennummer**/**Bestellnummer** oder schlicht **ID** angezeigt bekommen.

Da einige Entwickler, wie auch ich, für Primärschlüsselfelder und Fremdschlüsselfelder die gleichen Bezeichnungen wählen – also zum Beispiel **KundeID** als Primärschlüsselfeld in der Tabelle **tblKunden** und **KundeID** als Fremdschlüsselfeld in der Tabelle **tblBestellungen** –, wollen wir hier unterscheiden. Während wir für das Fremdschlüsselfeld **KundeID** vermutlich die Beschriftung **Kunde** wählen werden, erhält das Primärschlüsselfeld **KundeID** eher eine Beschriftung wie **Kunde-ID** oder auch nur **ID**.

Automatisierung per VBA

Unsere Idee ist, dass wir die üblichen Zuordnungen von Feldnamen und Beschriftungen in einer Tabelle speichern, die wir beispielsweise **tblCaptions** nennen. Diese soll vier Felder enthalten – **CaptionID**, **Fieldname**, **Caption** und **IsPrimaryKey**.

Für die Kombination aus den Feldern **Fieldname** und **IsPrimaryKey** definieren wir einen eindeutigen Index, damit zu jedem Feldnamen für einfache Felder und für Primärschlüsselfelder nur jeweils eine Beschriftung hinterlegt werden kann (siehe Bild 3).

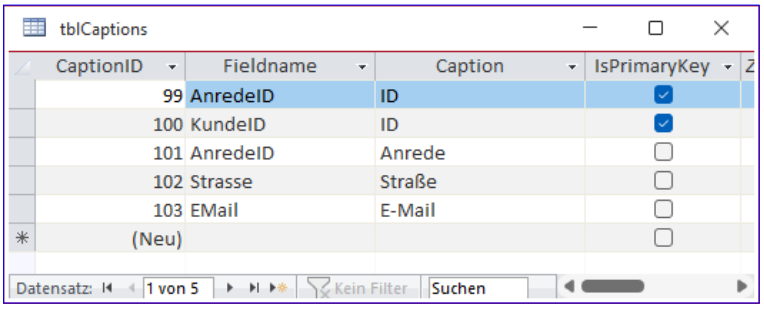


Bild 4: Die Tabelle tblCaptions mit einigen Beispieldaten

Zusätzlich wollen wir eine Prozedur programmieren, mit der wir die Felder einer Tabelle, die mit Beschriftungen ausgestattet werden soll, durchlaufen und prüfen, ob wir für die Feldnamen bereits entsprechende Beschriftungen in unserer Tabelle **tblCaptions** hinterlegt haben. Falls ja, soll die Beschriftung einfach für die Eigenschaft **Beschriftung** hinterlegt werden.

Für eine Tabelle wie **tblKunden** sieht der Inhalt der Tabelle **tblCaption** also wie in Bild 4 aus.

Beschriftungen per VBA zuweisen

Nun fehlt noch die passende Automatisierung. Diese erledigen wir mit einer VBA-Prozedur **ApplyCaptions**, die wir in Listing 1 abgebildet haben. Die Prozedur erwartet den Namen der Tabelle, deren Felder mit Beschriftungen ausgestattet werden sollen. Wir deklarieren vorsorglich schon einmal zwei Variablen für **Database**-Objekte: eines für die aktuelle Datenbank und eines für eine Add-In-Datenbank, denn wir werden die hier produzierten Elemente für die Automatisierung vermutlich nicht zu jeder neuen Datenbank hinzufügen, sondern diese über ein Add-In aufrufen wollen. Und bei Verwendung eines Add-Ins muss man unterscheiden, ob man auf die Daten der Tabellen der Add-In-Datenbank oder der aufrufenden Datenbank zugreifen möchte. Deshalb füllt die Prozedur auch die Variable **db** über die Funktion **CurrentDb** mit einem Verweis auf die Host-Datenbank und die Variable **dbc** über

```
Public Sub ApplyCaptions(strTable As String)
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim rstCaptions As DAO.Recordset
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Dim prp As DAO.Property
    Dim strCaption As String
    Dim bolPrimaryKey As Boolean
    Set db = CurrentDb
    Set dbc = CodeDb
    Set tdf = db.TableDefs(strTable)
    Set rstCaptions = dbc.OpenRecordset("SELECT * FROM tblCaptions", dbOpenDynaset)
    For Each fld In tdf.Fields
        bolPrimaryKey = IsPrimaryKey(tdf, fld)
        rstCaptions.FindFirst "Fieldname = '" & fld.Name & "' AND IsPrimaryKey = " & CInt(bolPrimaryKey)
        If Not rstCaptions.NoMatch Then
            strCaption = rstCaptions!Caption
            On Error Resume Next
            Set prp = fld.CreateProperty("Caption", dbText, strCaption)
            fld.Properties.Append prp
            If Not Err.Number = 0 Then
                fld.Properties("Caption") = strCaption
            End If
            On Error GoTo 0
        End If
    Next fld
End Sub
```

Listing 1: Anwenden von Beschriftungen auf die Felder der per Parameter übergebenen Tabelle

die Funktion **CodeDb** mit einem Verweis auf die Add-In-Datenbank. Und auch wenn wir aktuell noch direkt in der Host-Datenbank programmieren, ist das kein Problem, denn in diesem Fall greifen **CurrentDb** und **CodeDb** einfach beide auf die aktuelle Datenbank zu.

Weiter im Code: Dort referenzieren wir mit der **TableDef**-Variable **tdf** das **TableDef**-Objekt für die mit dem Parameter **strTable** übergebene Tabelle. Dann öffnen wir ein Recordset für den Zugriff auf die Daten der Tabelle **tblCaptions** und speichern den Verweis in **rstCaptions**.

Anschließend durchläuft die Prozedur alle Felder der Tabelle, deren Felder wir mit Beschriftungen ausstatten wollen, in einer **For Each**-Schleife. Innerhalb der Schleife prüft die Prozedur zunächst, ob es sich bei dem aktuell untersuchten Feld um ein Primärschlüsselfeld handelt. Dies geschieht mit der Funktion **IsPrimaryKey**, die wir weiter unten beschreiben.

Ob es sich um ein Primärschlüsselfeld handelt oder nicht, ist wie oben bereits erwähnt wichtig, da wir bei gleicher Benennung von Primärschlüsselfeldern und Fremdschlüsselfeldern gegebenenfalls unterschiedliche Beschriftungen verwenden wollen. Danach versucht die Prozedur, im Recordset mit den Beschriftungen einen Eintrag zu finden, dessen Feld **Fieldname** dem Namen des aktuell durchlaufenen Feldes entspricht und das, je nachdem, welches Ergebnis die Funktion **IsPrimaryKey** geliefert hat, auch dieses Feld in die Bedingung einbezieht.

Wurde ein Eintrag gefunden, liefert die in der kommenden **If...Then**-Bedingung überprüfte Eigenschaft **NoMatch** den Wert **False**. In diesem Fall werden die Anweisungen innerhalb der **If...Then**-Bedingung durchgeführt. Wir schreiben dort die zu dem Feld passende Beschriftung in die Variable **strCaption** und weisen diese dann der Eigenschaft **Beschriftung** zu.

Das ist nicht ganz trivial, denn im Gegensatz zum Tabellenentwurf in der Benutzeroberfläche, wo die Eigenschaft jederzeit zur Verfügung steht, ist die dahinter stehende Eigenschaft **Caption** standardmäßig gar nicht in der Auflistung der Properties eines **Field**-Objekts vorhanden. Das entsprechende **Property**-Objekt wird erst angelegt, wenn der Benutzer eine Beschriftung einträgt.

Um nicht prüfen zu müssen, ob die Property bereits vorhanden ist, versuchen wir einfach, diese bei deaktivierter Fehlerbehandlung anzulegen. Dazu nutzen wir zwei Anweisungen.

Die erste ruft die **CreateProperty**-Methode für das Feld auf und legt für die Eigenschaft **Caption** mit dem Datentyp **dbText** den Wert aus **strCaption** fest. Danach hängt sie das neu erstellte **Property**-Objekt an die **Properties**-Auflistung des **Field**-Objekts für das Feld an.

Ist die Property bereits vorhanden, löst dieser Vorgang einen Fehler aus. Das prüfen wir in der folgenden **If... Then**-Bedingung und legen in diesem Fall einfach den Wert der Eigenschaft **Caption** auf den Wert aus **strCaption** fest.

Dies wiederholen wir für alle Felder der angegebenen Tabelle und tragen so für alle Felder, für die wir in der Tabelle **tblCaptions** eine alternative Beschriftung angegeben haben, die jeweilige Beschriftung ein.

Herausfinden, ob ein Feld ein Primärschlüsselfeld ist

Die Funktion **IsPrimaryKey** erwartet das **TableDef**-Objekt und das **Field**-Objekt des zu untersuchenden Feldes als Parameter (siehe Listing 2). Sie durchläuft alle **Index**-Elemente des übergebenen **TableDef**-Objekts in einer **For Each**-Schleife. Wenn der Index ein Primärschlüssel ist, durchläuft sie außerdem alle Felder, die zu diesem Primärschlüsselindex gehören (ein Primärschlüssel kann ja auch aus mehreren Feldern bestehen). Ist das zu untersuchende Feld hier enthalten, handelt es sich bei diesem um ein Primärschlüsselfeld und wir können die Funktion wieder verlassen.

```
Public Function IsPrimaryKey(tdf As DAO.TableDef, fld As DAO.Field) As Boolean
    Dim idx As DAO.Index
    Dim fldPK As DAO.Field
    For Each idx In tdf.Indexes
        If idx.Primary Then
            For Each fldPK In idx.Fields
                If fld.Name = fldPK.Name Then
                    IsPrimaryKey = True
                    Exit Function
                End If
            Next fldPK
        End If
    Next idx
End Function
```

Listing 2: Funktion zum Prüfen, ob ein Feld ein Primärschlüsselfeld ist

Noch mehr Automation: Beschriftungen einlesen

Da wir uns nun schon ein wenig in das Thema eingearbeitet haben, können wir auch noch weiter automatisieren. Wieso sollten wir eigentlich die Felder und die dafür vorgesehenen Beschriftungen von Hand in die Tabelle **tblCaptions** eintragen?

Nicht, dass das nicht auch eine entspannende Beschäftigung wäre ... Aber noch praktischer wäre es doch, wenn wir die Beschriftung für alle vorhandenen Felder einer Tabelle auch per VBA einlesen könnten. Und genau dazu können wir die Prozedur **SaveCaptions** aus Listing 3 nutzen.

Diese Prozedur erwartet den Namen der Tabelle, die ausgelesen werden soll, als Parameter. Auch hier arbeiten wir wieder mit zwei **Database**-Objekten, um die Funktionalität gegebenenfalls schnell in ein Add-In umwandeln zu können. Wir referenzieren die zu untersuchende Tabelle als **TableDef**-Objekt (**tdf**) und die Tabelle mit den Beschriftungen als Recordset (**rstCaptions**).

Dann durchlaufen wir alle Felder der Tabelle über die **Fields**-Auflistung. Dabei prüft die Prozedur zuerst wieder, ob es sich bei dem aktuell untersuchten Feld um ein Primärschlüsselfeld handelt. Falls ja, soll dies später im Feld **IsPrimaryKey** der Tabelle **tblCaptions** vermerkt werden.

Wir wissen auch bereits, dass die **Caption**-Property nur für Felder existiert, für die bereits eine Beschriftung hinterlegt wurde. Also können wir auch hier nicht einfach auf diese Eigenschaft zugreifen, sondern müssen damit rechnen, dass beim Zugriff auf die nicht vorhandene Eigenschaft ein Fehler ausgelöst wird.

Also umgehen wir dies, indem wir die Variable **strCaption** zum Speichern der Beschriftung erst auf eine leere Zeichenkette setzen und dann bei deaktivierter eingebauter Fehlerbehandlung versuchen, auf **fld.Properties("Caption")** zuzugreifen. Ist **strCaption** danach nicht leer, haben wir ein Feld mit einer Beschriftung gefunden und wollen den Feldnamen und die Beschriftung in die Tabelle **tblCaptions** eintragen. Das erledigen wir wieder bei deaktivierter Fehlerbehandlung mit einer **INSERT INTO**-Anweisung. Das hat den Hintergrund, dass wir für die Kombination der Felder **Fieldname** und **IsPrimaryKey** einen eindeutigen Index definiert haben, und wenn es in der Tabelle bereits einen Datensatz für die Kombination dieser Felder gibt, löst der Versuch, diesen erneut anzulegen, einen Fehler aus.

Diesen behandeln wir diesmal, indem wir auf die Fehlernummer **3022** prüfen. Haben wir einen Treffer, lesen wir die für diesen Feldnamen gespeicherte Beschriftung in die Variable **strCaptionOld** ein. Unterscheidet sich dieser von

```
Public Sub SaveCaptions(strTable As String)
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Dim prp As DAO.Property
    Dim rstCaptions As DAO.Recordset
    Dim strCaption As String
    Dim strCaptionOld As String
    Dim bolPrimaryKey As Boolean
    Set db = CurrentDb
    Set dbc = CodeDb
    Set tdf = db.TableDefs(strTable)
    Set rstCaptions = dbc.OpenRecordset("SELECT * FROM tblCaptions", dbOpenDynaset)
    For Each fld In tdf.Fields
        bolPrimaryKey = IsPrimaryKey(tdf, fld)
        strCaption = ""
        On Error Resume Next
        strCaption = fld.Properties("Caption")
        On Error GoTo 0
        If Not Len(strCaption) = 0 Then
            On Error Resume Next
            dbc.Execute "INSERT INTO tblCaptions(Fieldname, Caption, IsPrimaryKey) VALUES('" & fld.Name & "', '" _
                & strCaption & "', " & CInt(bolPrimaryKey) & ")", dbFailOnError
            Select Case Err.Number
                Case 3022
                    rstCaptions.FindFirst ("Fieldname = '" & fld.Name & "' AND IsPrimaryKey = " & CInt(bolPrimaryKey))
                    strCaptionOld = rstCaptions!Caption
                    If Not strCaption = strCaptionOld Then
                        If MsgBox("Für das Feld '" & fld.Name & "' wurde bereits die Beschriftung '" & strCaptionOld _
                                & "' hinterlegt. Soll die in der Tabelle '" & strTable & "' gefundene Beschriftung '" _
                                & strCaption & "' dafür gespeichert werden?", vbYesNo + vbExclamation, _
                                "Beschriftung bereits vorhanden.") = vbYes Then
                            dbc.Execute "UPDATE tblCaptions SET Caption = '" & strCaption & "', IsPrimaryKey = " _
                                & CInt(bolPrimaryKey) & " WHERE Fieldname = '" & fld.Name & "'", dbFailOnError
                        End If
                    End If
                Case 0
                Case Else
                    MsgBox "Fehler: " & Err.Number & ", " & Err.Description
            End Select
        End If
    Next fld
End Sub
```

Listing 3: Auslesen von Beschriftungen aus der angegebenen Tabelle

Alternative MsgBox mit Überschrift

Neulich ist mir aufgefallen, dass es in Access Meldungen vom System gibt, die nicht nur normalen Text enthalten, sondern die fett gesetzte Überschriften präsentieren. Also dachte ich, dass dies doch auch für unsere eigenen Meldungsfenster möglich sein müsste und habe mich auf die Suche begeben. Und siehe da – es gibt eine Alternative zur herkömmlichen VBA-MsgBox-Anweisung, mit der wir Überschriften in fetter Schrift darstellen können!

Die gestalterischen Möglichkeiten der unter VBA verfügbaren **MsgBox**-Anweisung sind schnell ausgeschöpft. Wir können ein Icon setzen, verschiedene Kombinationen von Schaltflächen anzeigen, einen Text für die Titelzeile definieren und den anzuzeigenden Text angeben. Diesen können wir immerhin durch Einfügen der Konstanten **vbCrLf** noch auf mehrere Zeilen aufteilen – und zwei aufeinanderfolgende **vbCrLf** liefern eine Leerzeile. Ein Beispiel für einen Aufruf sieht wie folgt aus:

```
MsgBox "Text der Meldung in der ersten Zeile." _
    & vbCrLf & vbCrLf & "Noch eine Zeile mit weiteren _
        Informationen.", vbOKOnly + vbExclamation, _
                                "Titel der Meldung"
```

Führen wir diese Anweisung aus, erhalten wir die Meldung aus Bild 1.

Meldungsfenster per WizHook-Klasse

VBA bietet eine versteckte Klasse mit undokumentierten Methoden namens WizHook. Diese können wir sichtbar machen, indem wir im Objektkatalog das Kontextmenü anzeigen und dort die Option **Verborgene Elemente** aktivieren. Danach finden wir per Suche das **WizHook**-Element und auch seine Methode **WizMsgBox** (siehe Bild 2).

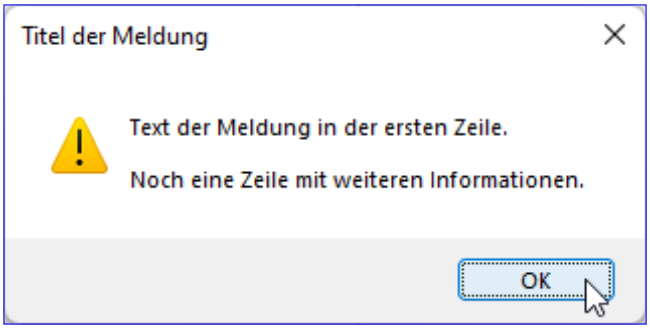


Bild 1: VBA-Meldungsfenster

Um die Funktionen der **WizHook**-Klasse zu aktivieren, müssen wir die Eigenschaft **Key** auf den folgenden Wert einstellen:

```
WizHook.Key = 51488399
```

Danach können wir die **WizMsgBox**-Funktion wie folgt aufrufen. Dabei ist anzumerken, dass Werte für alle Para-

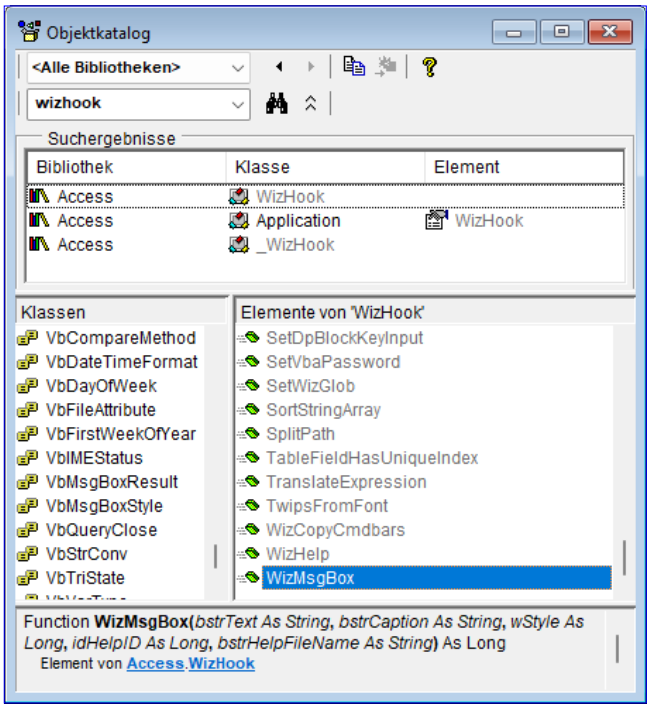


Bild 2: Die versteckten **WizHook**-Funktionen

Steuerelemente ausrichten und anpassen

Ein immer wieder auftauchender und zeitraubender Vorgang ist das Ausrichten und Anpassen von Steuerelementen. Egal, ob man Elemente aus der Feldliste in den Entwurf zieht oder neue Steuerelemente anlegt – diese landen meist nicht genau an der gewünschten Stelle, haben nicht die passende Breite oder man muss noch an der Ausrichtung zu den bereits vorhandenen Steuerelementen arbeiten. In diesem Beitrag schauen wir uns an, welche Möglichkeiten zum Ausrichten und Anpassen die Benutzeroberfläche von Microsoft Access bietet. In weiteren Beiträgen kümmern wir uns dann um Lösungen, die das Ausrichten und Anpassen vereinfachen, damit Sie sich um wichtigere und interessantere Aufgaben kümmern können.

Steuerelemente aus der Feldliste

Da Access-Formulare meist die Daten aus den Feldern der Tabellen oder Abfragen anzeigen sollen, an die sie gebunden sind, ist das Ausrichten und Anpassen von Feldern, die aus der Feldliste in den Formularentwurf gezogen wurden, wohl der wichtigste Fall. Haben wir unser Formular also über die Eigenschaft **Datensatzquelle** beispielsweise an eine Tabelle namens **tblKunden** gebunden, zeigt die Feldliste wie in Bild 1 die verfügbaren Felder an. Diese können wir nun einzeln in den Entwurf ziehen oder wir markieren gleich mehrere Felder und ziehen diese dann ins Formular. Beim Fallenlassen der Elemente sollte man den Mauszeiger nicht allzu nah am linken Formularrand platzieren, denn sonst erlebt man das erste Dilemma (siehe Bild 2).

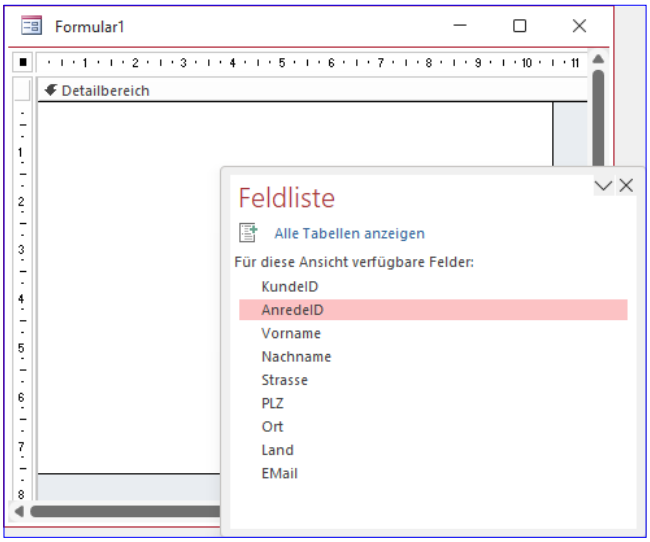


Bild 1: Ausgangssituation vor dem Hinzufügen von gebundenen Steuerelementen

Doppelpunkte und Beschriftungen

Geschieht dies, löscht man die Elemente am besten direkt wieder und beginnt noch mal von vorn. Danach sieht das Ergebnis schon besser aus, allerdings möchte man vielleicht die Texte der Bezeichnungsfelder mit Doppelpunkten ausstatten und auch die Beschriftungen gleich noch anpassen – zum Beispiel, indem man **KundeID**

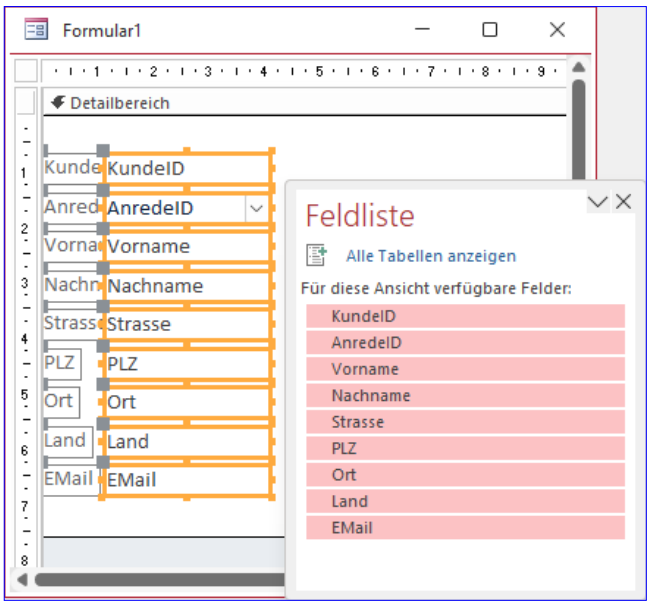


Bild 2: Das geschieht, wenn man Elemente zu nah am linken Formularrand fallen lässt.

durch **Kunde-ID**, **AnredeID** durch **Anrede**, **Strasse** durch **Straße** oder **EMail** durch **E-Mail** ersetzt. Wie das gelingt, haben wir schon an verschiedenen Stellen beschrieben, daher in Kurzfassung:

- Die Beschriftung, die in Bezeichnungsfeldern gebundener Felder erscheinen soll, stellt man direkt in der Eigenschaft Beschriftung für das jeweilige Feld im Tabellenentwurf ein. Access übernimmt diese dann direkt, wenn gebundene Steuerelemente auf Basis dieser Felder angelegt werden.

- Damit direkt ein Doppelpunkt angehängt wird, stellt man eine Eigenschaft des jeweiligen Steuerelements namens **Mit Doppelpunkt** auf den Wert **Ja** ein. Dazu öffnet man das Formular in der Entwurfsansicht und klickt im Ribbon auf das jeweilige Steuerelement, legt es jedoch noch nicht an. Das Eigenschaftenblatt zeigt nun unter **Format** ein paar Eigenschaften an, die man bei fertigen Steuerelementen nicht zu Gesicht bekommt – siehe Bild 3. Wenn man dies für alle noch zu erstellenden Formulare der aktuellen Datenbank aktivieren möchte, speichert man das Formular, noch bevor man Steuerelemente hinzugefügt hat, nach der Anpassung der Eigenschaft **Mit Doppelpunkt** unter dem Namen **Normal**. Access verwendet dieses Formular dann als Vorlage für die folgenden Formulare.

Damit können wir also schon mal ein paar Aufgaben abhaken.

Bestehende Elemente ausrichten

Wenn man von den wie weiter oben beschriebenen etwas zu schmal geratenen Bezeichnungsfeldern ausgeht und diese anpassen möchte, muss man natürlich nicht direkt von vorn beginnen. Die Bezeichnungsfelder sind gar nicht zu schmal, sie werden nur von den gebundenen Feldern überdeckt. Wir müssen also nur die gebundenen Felder etwas weiter nach rechts bewegen. Das ist leider leichter gesagt als getan. Wenn wir sie einfach wie in der Abbildung markieren und geschlossen nach rechts ziehen, bewegen sich die Bezeichnungsfelder nämlich einfach mit, weil sie mit den gebundenen Feldern verbunden sind.

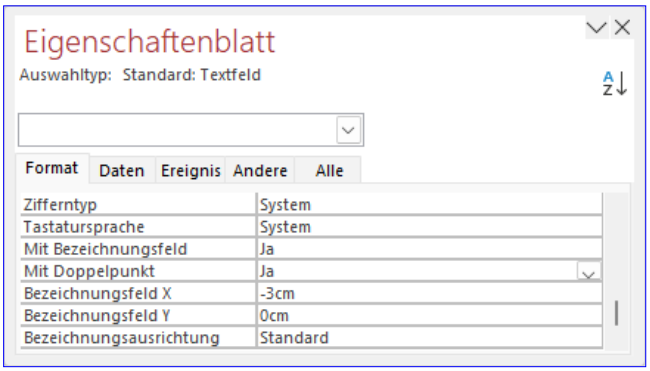


Bild 3: Doppelpunkte für Bezeichnungsfelder aktivieren

Wir können die Steuerelemente, an die ein Bezeichnungsfeld gekoppelt ist, nur auf eine Weise unabhängig vom Bezeichnungsfeld verschieben – nämlich, indem wir sie an dem grauen Kästchen oben links an der Markierung anfassen und ziehen. Das gelingt allerdings auch nur mit einem Steuerelement zur gleichen Zeit.

Also wählen wir eine Alternative: Wir »verschieben« die gebundenen Steuerelemente geschlossen nach rechts, indem wir den rechten Rand nach rechts ziehen und die Elemente damit zunächst verbreitern (siehe Bild 4). Danach stellen wir die alte Breite wieder her, indem wir den linken Rand entsprechend nach rechts schieben.

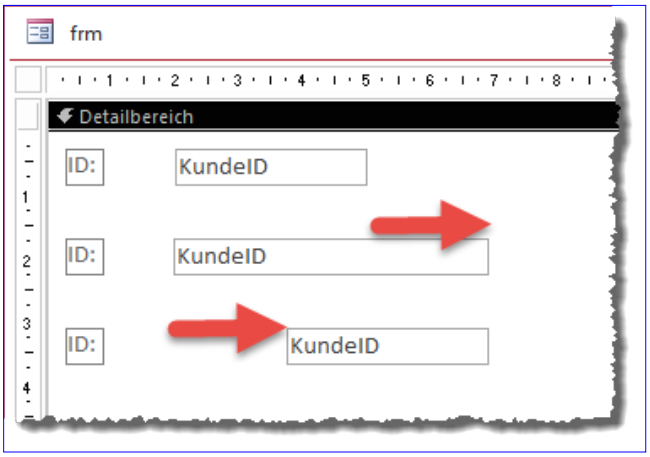


Bild 4: Steuerelemente verschieben ohne zu verschieben

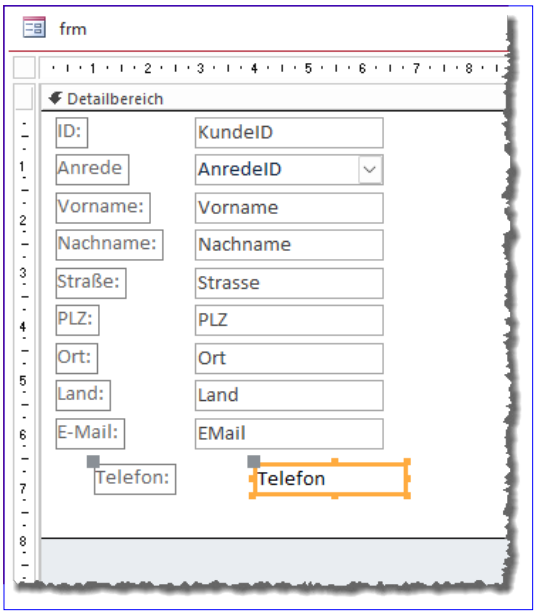


Bild 5: Neue Elemente fügen sich nicht gleich ein.

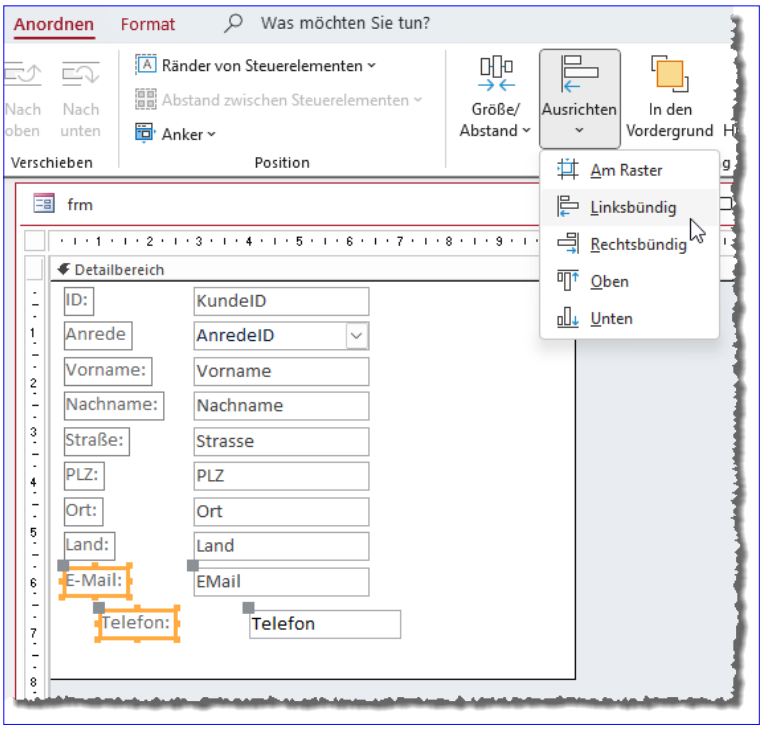


Bild 6: Links ausrichten

Hinzufügen weiterer Elemente

Sie wären nicht der erste Entwickler, der erst ein paar Steuerelemente aus der Feldliste zum Formular hinzufügt und dann noch weitere ergänzt – das kommt vor, beispielsweise, weil die zugrunde liegende Tabelle oder Abfrage im ersten Entwurf noch nicht alle benötigten Felder enthielt.

Fügen Sie also ein weiteres Textfeld hinzu, ordnet sich das in der Regel nicht einfach passend ein, sondern wir müssen die Position und die Größe des neuen Steuerelements samt Bezeichnungsfeld anpassen.

In unserem Beispiel kommt so noch ein Feld namens **Telefon** hinzu, das weder korrekt ausgerichtet ist noch die gewünschte Breite hat (siehe Bild 5).

Als Erstes kümmern wir uns um die Ausrichtung des Bezeichnungsfeldes. Um dieses auf den gleichen Abstand zum linken Rand zu bringen wie das darüber liegende Bezeichnungsfeld, markieren wir die beiden Steuerelemente zunächst. Dann können wir die Ausrichtung entweder per Ribbon-Befehl oder Kontextmenü vornehmen.

Im Ribbon befinden sich die benötigten Befehle unter **Anordnen|Anpassung und Anordnung|Ausrichten** (siehe Bild 6). Hier nutzen wir den Eintrag **Linksbündig**, um zunächst die beiden Bezeichnungsfelder auszurichten. Da die Breite der Bezeichnungsfelder üblicherweise beim Hineinziehen von Elementen aus der Feldliste bereits passt, brauchen wir hier keine weiteren Änderungen vorzunehmen.

Also wenden wir uns dem neuen Textfeld zu. Dieses soll ebenfalls linksbündig mit dem darüber liegenden Steuerelement ausgerichtet werden, was wir durch die gleichen Schritte wie zuvor erledigen.

Danach gleichen wir die Breite des neuen Steuerelements an das darüber befindliche Element an. Dazu markieren wir beide und wählen dann den Befehl **Größe/Abstand-|am breitesten** aus dem Ribbon (siehe Bild 7).

Damit fehlt nur noch die Anpassung des vertikalen Abstands der Steuerelemente zueinander. Der Abstand des

Steuerelemente ausrichten per VBA

Access bietet verschiedene Möglichkeiten, um Steuerelemente auszurichten. Diese stellen wir im Beitrag Steuerelemente ausrichten (www.access-im-unternehmen.de/1431) vor. All diese Methoden haben jedoch Vor- und Nachteile. Der größte Nachteil ist, dass sie Zeit kosten – Zeit, die man in viel schönere Arbeiten investieren könnte. Deshalb schauen wir uns in diesem Beitrag an, wie wir in der Entwurfsansicht selektierte Steuerelemente einfach per VBA ausrichten können. Dabei ist nicht die Technik die entscheidende Frage, sondern die Vorgabe, nach welchen Regeln die Ausrichtung genau erfolgen soll. In diesem Beitrag beschreiben wir, wie man per VBA solche Steuerelemente wie Textfeld, Kombinationsfeld, Listenfeld und Kontrollkästchen und ihre Bezeichnungsfelder, die untereinander angeordnet sind, ausrichten kann.

Ausgangssituation im Formular

Wir gehen davon aus, dass Sie als Entwickler auch mit dem Zusammenstellen von Formularen und dem Layout der enthaltenen Steuerelemente zu tun haben.

In vielen Fällen kann man sich die Aufgabe, Steuerelemente sauber ausgerichtet anzulegen, erleichtern – zum Beispiel durch das Ziehen der gebundenen Felder aus der Feldliste in das Formular.

Aber sobald man der Datensatzquelle des Formulars ein weiteres Feld hinzugefügt hat, das nun auch im Formularentwurf landen soll, muss man seine Position an die der anderen anpassen.

Da ist man mit den im oben genannten Beitrag vorgestellten Methoden durchaus ein paar Sekunden beschäftigt – und dabei handelt es sich um eine Tätigkeit, die man während eines Access-Entwicklerlebens sehr oft wiederholt.

Wir wollen die Ausgangssituation noch ein wenig chaotischer darstellen und gehen während der Entwicklung von Steuerelementen aus, die wie in Bild 1 angeordnet sind.

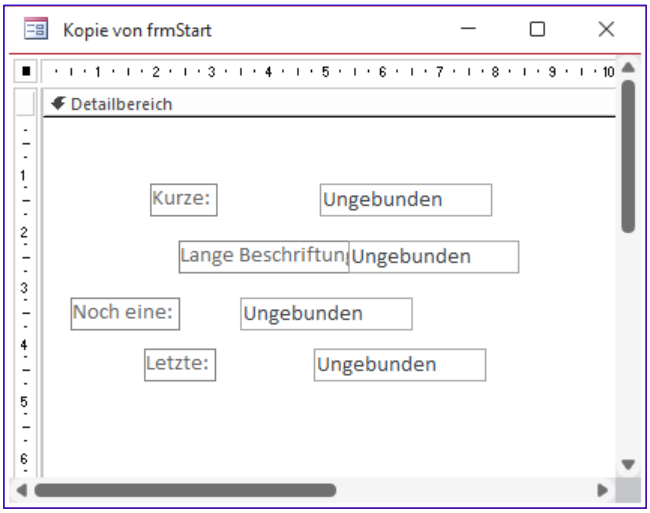


Bild 1: Ausgangssituation im Formular

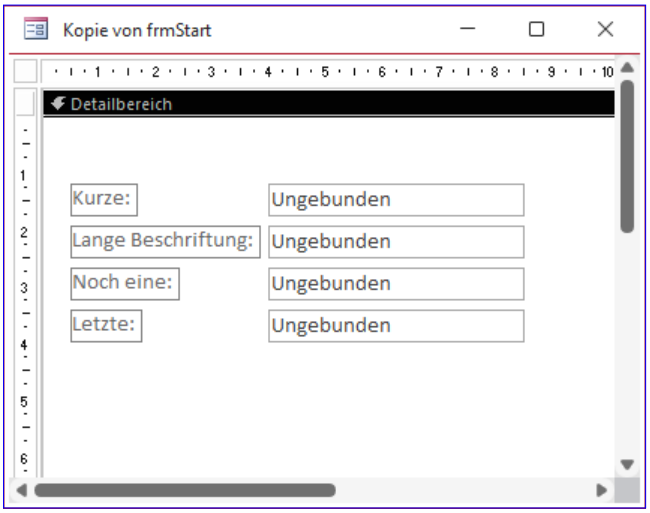


Bild 2: Fertig ausgerichtete Steuerelemente

Vorgaben zur Anordnung von Steuerelementen

Wenn man diese nun manuell anordnen würde, wäre das Markieren aller Beschriftungsfelder durch Ausziehen eines geeigneten Rahmens wohl der erste Schritt. Dann würde man aus dem Kontextmenü den Befehl **Ausrichten|Linksbündig** wählen. Als Nächstes wären dann die Textfelder an der Reihe, die man auf die gleiche Weise linksbündig anordnet.

Dann bringt man diese vielleicht noch auf eine Breite und verschiebt sie so nah an die Beschriftungsfelder heran, dass der Abstand zwischen dem breitesten Beschriftungsfeld und den Textfeldern etwa so groß ausfällt wie der vertikale Abstand zwischen den Textfeldern. Das Ergebnis sieht dann beispielsweise wie in Bild 2 aus.

Vorgang per VBA abbilden

Nun wollen wir diese ganzen Schritte per VBA erledigen. Dazu können wir grundsätzlich ähnlich vorgehen. Aber welche Rahmenbedingungen dazu wollen wir festhalten? Als Erstes benötigen wir die Position unseres Blocks vom linken und vom oberen Rand aus betrachtet. Deshalb ermitteln wir die Y-Position des obersten Steuerelements und die X-Position des am weitesten links befindlichen Steuerelements.

Danach würden wir die Position aller Bezeichnungsfelder links an der soeben ermittelten Position ausrichten.

Danach wird es schon spannend – nämlich bei der Positionierung der Steuerelemente von oben nach unten. Die erste Idee war, einfach alle markierten Steuerelemente von oben nach unten zu durchlaufen und dabei dem obersten Steuerelement den soeben ermittelten Abstand vom oberen Rand zu vergeben. Die nachfolgenden Steuerelemente würden wir dann jeweils unter dem zuvor ausgerichteten Steuerelement positionieren – mit einem kleinen zusätzlichen Abstand.

Allerdings durchläuft Access die Steuerelemente über die **Controls**-Auflistung in der Reihenfolge, die durch die Aktivierreihenfolge vorgegeben wurde. Hier stellt sich die Frage, ob wir die Steuerelemente nach der Aktivierreihenfolge ausrichten wollen oder nach der aktuellen vertikalen Anordnung. Intuitiver scheint es, die Steuerelemente nach der aktuellen Anordnung auszurichten.

Diese müssten wir dann erst einmal ermitteln – und das ist nicht gerade trivial. Unsere Lösung für diese Frage ist, den Steuerelementnamen und die Y-Position in einem zweidimensionalen Array zu speichern und die Daten in diesem Array nach der Y-Position zu sortieren, bevor wir sie in der so festgelegten Reihenfolge ausrichten.

Des Weiteren stellt sich die Frage, ob wir uns bei der Höhe für ein Bezeichnungsfeld-Steuerelement-Paar nach der Höhe des Bezeichnungsfeldes oder des Steuerelements richten. Es kann sowohl vorkommen, dass das Bezeichnungsfeld höher ist als das Steuerelement (bei einer längeren Bezeichnung), aber auch Textfelder kommen gelegentlich mit einer größeren Höhe, beispielsweise um mehrere Zeilen anzuzeigen.

Spätestens wenn Listenfelder ins Spiel kommen, sind diese meist höher als das jeweilige Bezeichnungsfeld.

Die einfachste Lösung für diesen Fall scheint zu sein, das jeweils höhere Element zu berücksichtigen.

Bei den Bezeichnungsfeldern stellt sich die Frage, ob diese bereits die richtige Breite aufweisen. Sicherheitshalber wollen wir die Breite auf den tatsächlich enthaltenen Bezeichnungstext anpassen. Damit ermitteln wir nun die maximale Breite der Bezeichnungsfelder und stellen alle Bezeichnungsfelder auf diese Breite ein. Dies ist eine Vorbereitung auf den Fall, dass der Benutzer seine Bezeichnungsfelder rechtsbündig ausrichten möchte.

Danach richten wir die eigentlichen Steuerelemente, in unserem Beispiel also die Textfelder, entsprechend aus. Dabei nutzen wir die X-Position der Bezeichnungsfelder plus ihre Breite plus wiederum einem kleinen Abstand, wie

wir ihn auch bereits für die vertikale Anordnung genutzt haben.

Sonderfall Kontrollkästchen

Während die Höhe von Textfeldern in der Regel an die jeweiligen Bezeichnungsfelder angepasst ist, sind Kontrollkästchen etwas weniger hoch und ihr Abstand vom oberen Rand des Formulars stimmt nicht mit dem des dazugehörigen Bezeichnungsfelds überein. Hier müssen wir eine Spezialbehandlung vornehmen. Wir können das Kontrollkästchen nicht auf der gleichen Y-Position wie das Bezeichnungsfeld platzieren, da es sonst optisch etwas zu weit oben angezeigt wird.

Also bauen wir beim Ausrichten des Kontrollkästchens einen zusätzlichen Abstand von oben ein. Da es normalerweise auch noch weniger hoch als ein Textfeld oder Bezeichnungsfeld ist, müssen wir in diesem Fall die Höhe des Bezeichnungsfeldes als Maßstab für die Ausrichtung des darunter liegenden Steuerelements nutzen.

Zusätzliche Einstellungen

Neben den bereits erwähnten Vorgaben wollen wir dem Benutzer noch die Möglichkeit geben, weitere Einstellungen vorzunehmen. Dazu stellen wir der Hauptprozedur einige Parameter bereit. Damit wollen wir die folgenden Einstellungen vornehmen:

- Abstand zwischen Bezeichnungsfeld und Steuerelement in Pixeln
- Vertikaler Abstand zwischen zwei Steuerelementen
- Ausrichtung der Bezeichnungsfelder – wahlweise Standard, links, mittig oder rechts zentriert oder verteilt
- Optionales Hinzufügen von Doppelpunkten rechts von den Beschriftungen, sofern noch nicht vorhanden

Doppelpunkte hinter Beschriftungen setzen

Zum Aufwärmen wollen wir die Prozedur schreiben, mit der wir den aktuell markierten Bezeichnungsfeldern einen Doppelpunkt hinzufügen, sofern dieser noch nicht vorhanden ist. Diese Prozedur heißt **AddColonToLabels** und ist in Listing 1 abgebildet.

Die Prozedur können wir auch allein aufrufen. Sie deklariert eine **Form**- und eine **Control**-Variable. Mit der **Form**-Variable **frm** referenziert sie über die Eigenschaft **Screen.ActiveForm** das aktuell markierte Formular. Die Variable **ctl** kann alle Steuerelemente referenzieren. Damit durchlaufen wir in einer **For Each**-Schleife alle Steuerelemente des referenzierten Formulars.

In der folgenden **If...Then**-Bedingung prüfen wir mit der Eigenschaft **InSelection**, ob das aktuelle mit **ctl** über die Schleife referenzierte Steuerelement derzeit markiert ist. Ist das der Fall, untersuchen wir in einer **Select Case**-Bedingung durch einen Vergleich der Eigenschaft **ControlType** mit der Konstanten **acLabel**, ob es sich um ein Bezeichnungsfeld handelt. Falls ja, prüfen wir schließlich noch, ob das letzte Zeichen ein Doppelpunkt ist. Ist das nicht der Fall, fügen wir der Eigenschaft Caption am Ende einen Doppelpunkt hinzu.

```
Public Sub AddColonToLabels()
    Dim frm As Form
    Dim ctl As Control
    Set frm = Screen.ActiveForm
    For Each ctl In frm.Controls
        If ctl.InSelection Then
            Select Case ctl.ControlType
                Case acLabel
                    If Not Right(ctl.Caption, 1) = ":" Then
                        ctl.Caption = ctl.Caption & ":"
                    End If
            End Select
        End If
    Next ctl
End Sub
```

Listing 1: Hinzufügen von Doppelpunkten zu den Bezeichnungsfeldern

```
Public Sub AlignControls(Optional lngDistanceLabelControl As Long = 100, Optional lngVerticalDistance As Long = 100, _
        Optional varAlignment As Variant = 0, Optional bolColons As Boolean = False)
    Dim frm As Form, ctl As Control
    Dim lngLabelLeft As Long, lngLabelWidth As Long, lngControlLeft As Long, lngTop As Long, lngMaxHeight As Long
    Dim lblControl As Label, varControls() As Variant, i As Integer
    If Not GetFormInDesignView(frm) Then
        MsgBox "Es ist kein Formular in der Entwurfsansicht geöffnet.", vbOKOnly + vbExclamation
        Exit Sub
    End If
    If GetSelectedControlCount(frm) < 2 Then
        MsgBox "Es müssen mindestens zwei Steuerelemente ausgewählt sein.", vbOKOnly + vbExclamation
        Exit Sub
    End If
    DoCmd.Echo False
    If bolColons Then
        AddColonToLabels
    End If
    OptimizeLabelSizes frm
    lngLabelLeft = GetMostLeftLabel(frm)
    lngLabelWidth = GetMaxLabelWidth(frm)
    lngControlLeft = lngLabelWidth + lngDistanceLabelControl + lngLabelLeft
    varControls = GetControlArray(frm)
    SortArray varControls
    lngTop = varControls(0, 0)
    For i = 0 To UBound(varControls, 2)
        Set ctl = frm.Controls(varControls(1, i))
        ctl.Left = lngControlLeft
        If Not lngTop = 0 Then
            ctl.Top = lngTop
            If ctl.ControlType = acCheckBox Then 'Korrektur für Kontrollkästchen
                ctl.Top = ctl.Top + 30
            End If
            Set lblControl = ctl.Controls(0)
            lngMaxHeight = ctl.Height
            If lblControl.Height > lngMaxHeight Then
                lngMaxHeight = lblControl.Height
            End If
            lblControl.Top = lngTop
            lblControl.Left = lngLabelLeft
            lblControl.Width = lngLabelWidth
            If Not IsMissing(varAlignment) Then
                lblControl.TextAlign = varAlignment
            End If
        End If
        lngTop = lngTop + lngMaxHeight + lngVerticalDistance
    Next i
    DoCmd.Echo True
End Sub
```

Listing 2: Hauptprozedur zum Ausrichten von Steuerelementen

URLs kodieren per VBA

URLs enthalten häufig Sonderzeichen, die von Internet-Browsern nicht interpretiert werden können. Dies können zum Beispiel Leerzeichen in HTML-Dateinamen oder Sonderzeichen in URL-Parametern sein. Sollen für den Zugriff auf eine REST-API Parameter an eine URL angehängt werden, die Sonderzeichen enthalten, müssen diese für den Browser lesbar gemacht werden. Dies geschieht durch die sogenannte URL-Kodierung, auch Encoding genannt. Dabei werden die betreffenden Zeichen durch ein Prozentzeichen gefolgt von einem numerischen Code für das jeweilige Zeichen ersetzt – für ein Leerzeichen zum Beispiel %20. Die Umwandlung solcher Zeichenketten erledigen wir mit einer VBA-Funktion, die wir in diesem Artikel vorstellen.

Die Regeln für die nachfolgend beschriebenen Funktionen sehen wir folgt aus:

- Zahlen von 0-9, Buchstaben von a-z und Buchstaben von A-Z sollen beibehalten werden.
- Die Sonderzeichen Minus (**-**), Unterstrich (_), Punkt (**.**) und Tilde (**~**) sollen ebenfalls nicht enkodiert werden.
- Alle anderen Zeichen sollen enkodiert werden.

Was genau heißt nun »enkodieren«? Es bedeutet, dass wir ein Zeichen wie beispielsweise ein Leerzeichen durch ein Prozentzeichen (**%**) und den ASCII-Code im hexadezimalen Format ersetzen. Das Leerzeichen hat den ASCII-Code **32**, was hexadezimal **20** entspricht. Aus einem Leerzeichen wird also **%20**.

Man könnte meinen, wir kämen hier mit einer einfachen VBA-Funktion aus, welche untersucht, ob das Zeichen zu den nicht zu enkodierenden Zeichen gehört und die übrigen einfach in den entsprechenden Code umwandelt.

```
Public Function URLEncode_UTF8(ByVal strURL As String) As String
    Dim bytes() As Byte
    Dim b As Byte
    Dim i As Long
    Dim strTemp As String
    If Len(strURL) > 0 Then
        With New ADODB.Stream
            .Mode = adModeReadWrite
            .Type = adTypeText
            .Charset = "UTF-8"
            .Open
            .WriteText strURL
            .Position = 0
            .Type = adTypeBinary
            .Position = 3
            bytes = .Read
        End With
        For i = 0 To UBound(bytes)
            b = bytes(i)
            Select Case b
                Case 48 To 57, 65 To 90, 97 To 122, 45, 46, 95, 126
                    strTemp = strTemp & Chr(b)
                Case 0 To 15
                    strTemp = strTemp & "%0" & Hex(b)
                Case Else
                    strTemp = strTemp & "%" & Hex(b)
            End Select
        Next i
        URLEncode_UTF8 = strTemp
    End If
End Function
```

Listing 1: Die Funktion **URLEncode_UTF8**

Zugriff auf lexoffice per REST-API und VBA

lexoffice ist ein Online-Buchhaltungsdienst, der es Unternehmen ermöglicht, Rechnungen, Bestellungen und Bankkonten zu verwalten und Berichte zu erstellen. Der Service bietet ein einfaches Dashboard, mit dem Benutzer Geschäftsinformationen leicht auffinden und mit wenigen Klicks auswerten können. Rechnungen lassen sich online oder über die mobile App erstellen, Umsätze auf Bank- oder Paypalkonten liest lexoffice automatisch ein und es ermöglicht die Zuordnung von Ein- und Ausgangsrechnungen zu den Umsätzen. Aber das Beste ist: lexoffice bietet eine REST-API-Schnittstelle, die wir von einer Access-Datenbank aus per VBA ansteuern können. Grund genug, diese Schnittstelle einmal genauer anzusehen!

Von Access zu lexoffice

Die besten Beiträge schreibt das Leben, und so ist auch für die Entstehung dieses Beitrags ein Schritt des Autors für das aktuelle Thema verantwortlich. Die Buchhaltung soll umgezogen werden, um den aktuellen Randparametern gerecht zu werden – zum Beispiel den Grundsätzen ordnungsgemäßer Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff (**GoBD**). Als Werkzeug für diese Aufgaben haben wir **lexoffice** ausgewählt.

Damit entstehen einige Aufgaben:

- Einige offene Rechnungen von 2022, die erst in 2023 beglichen wurden, sollen noch aufgenommen werden.
- Diese Rechnungen müssen als PDF importiert werden und wir müssen diesen noch Daten zum Rechnungsempfänger und einige weitere Daten hinzufügen.
- Um die Kundendaten nicht manuell eintragen zu müssen, wollen wir die relevanten Datensätze mit der API aus der Datenbank nach **lexoffice** übertragen.

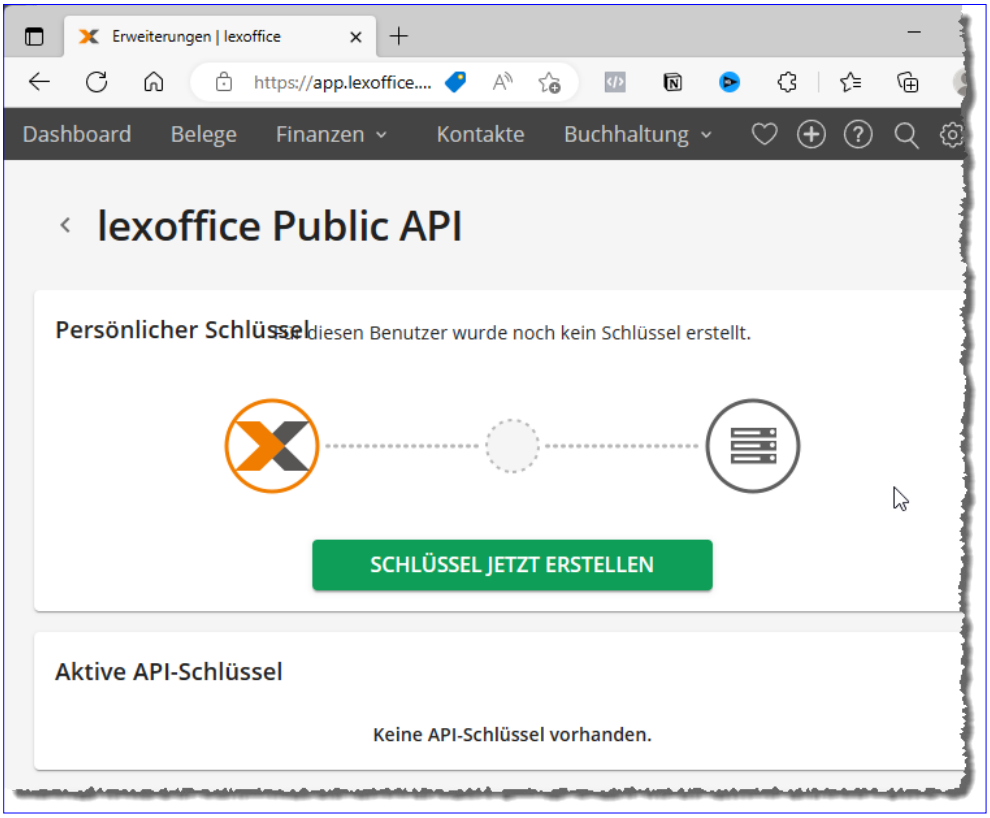


Bild 1: Anfordern eines persönlichen Schlüssels für die REST-Api

- Schließlich sollen neue Bestellungen direkt komplett mit **lexoffice** verwaltet werden. Diese kommen zum Teil aus dem Shopsystem, wozu es eine Schnittstelle gibt (die hier nicht Thema sein wird) und teilweise aus der Access-Bestellverwaltung. Bei letzteren handelt es sich um Rechnungen für die Verlängerung von Abonnements.

Vorbereitungen für den Zugriff auf die API von lexoffice

Wie bei den meisten API, auf die wir von außerhalb zugreifen wollen, benötigen wir auch hier eine Möglichkeit zur Authentifizierung.

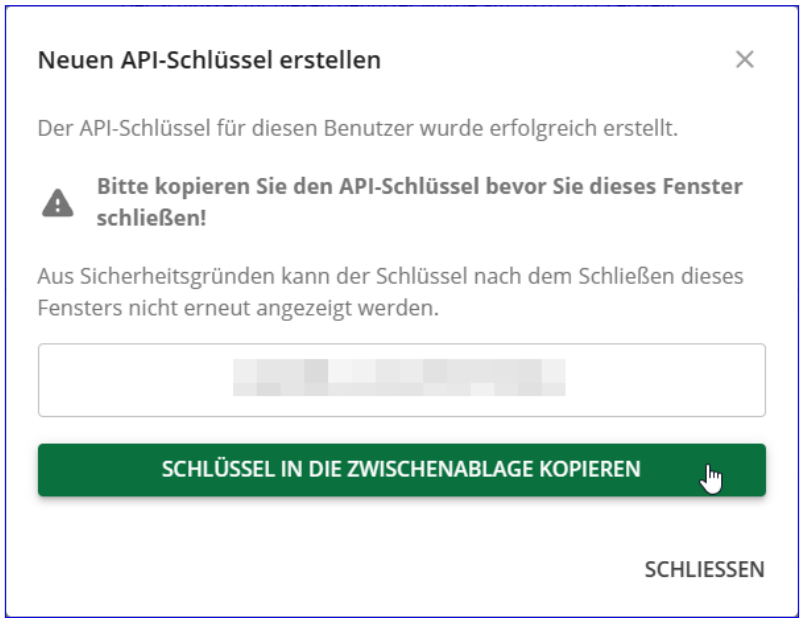


Bild 2: Kopieren des Schlüssels in die Zwischenablage

Dazu legen wir in **lexoffice** zunächst einen persönlichen Schlüssel an. Um das zu erledigen, rufen wir nach dem Einloggen in unseren Account bei lexware den folgenden Link auf:

```
https://app.lexoffice.de/addons/public-api
```

Damit landen wir auf einer Seite, wo wir per Mausklick einen passenden Schlüssel erstellen können (siehe Bild 1). Anschließend stimmen wir den Nutzungsbedingungen zu und erstellen den Schlüssel.

Im nächsten Schritt zeigt **lexoffice** den Schlüssel bereits an und bietet die Möglichkeit, diesen per Schaltfläche in die Zwischenablage zu kopieren (siehe Bild 2).

Um diesen dauerhaft zu sichern, legen wir in einem neuen VBA-Modul eine Konstante mit diesem Schlüssel an:

```
Public Const cstrAPIKey As String = "xxxxxxxxxxx-xxxxxxxx-xxxx"
```

Anschließend schließen wir den Dialog.

Datenbank vorbereiten

Für die nachfolgend vorgestellten VBA-Routinen benötigen wir im VBA-Projekt zusätzliche Verweise. Diese legen wir über den **Verweise**-Dialog an, den wir nach dem Aktivieren des VBA-Editors mit **Extras|Verweise** öffnen (siehe Bild 3).

Wir benötigen die folgenden zusätzlichen Verweise:

- **Microsoft XML, v6.0**
- **Microsoft Scripting Runtime**

Basisfunktion zum Senden von Anfragen an lexoffice

Wir verwenden eine Basisfunktion für den Zugriff auf die REST-API von lexoffice. Diese erwartet die folgenden Parameter:

- **strURL**: URL für den Zugriff auf die REST-API. Beginnt immer mit **https://api.lexoffice.io/v1/** und mit der Bezeichnung der abzufragenden/zu bearbeitenden Elemente, zum Beispiel **contacts** oder **invoices**. Anschließend folgen weitere Parameter wie beispielsweise die

Angabe von Filterkriterien beim Abfragen von Elementen.

- **strMethod**: Methode, die angibt, ob Daten beispielsweise gelesen (**GET**) oder geschrieben werden sollen (**POST**).
- **strRequest**: Falls erforderlich, können mit diesem Parameter zusätzliche Informationen im JSON-Format übergeben werden. Enthält beispielsweise die Kundendaten, wenn ein neuer Kontakt angelegt werden soll.
- **strResponse**: Liefert die Antwort im JSON-Format, also beispielsweise Daten über einen neu angelegten Kontakt oder abgefragte Kontaktdaten.

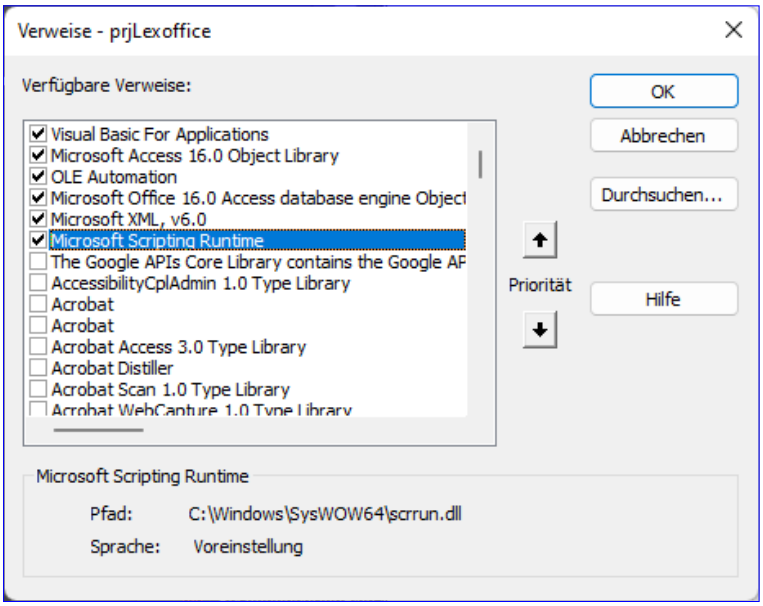


Bild 3: Verweise im VBA-Projekt

Die Funktion Request aus Listing 1 erstellt ein neues Objekt des Typs **XMLHTTP60**, mit dem wir auf die REST-API zugreifen. Dabei rufen wir als Erstes die **Open**-Methode auf, der wir die Werte der Parameter **strMethod** und **strURL** übergeben sowie für den dritten Parameter den Wert **False**. Damit geben wir an, dass der Aufruf asynchron erfolgen soll. Die folgenden Anweisungen stellen die Header für den Aufruf ein, unter anderem das Format für die hin- und herzuschickenden Daten (JSON) sowie für die

```
Public Function Request(strRequest As String, strURL As String, strMethod As String, strResponse As String) As Boolean
    Dim objXMLHTTP As MSXML2.XMLHTTP60
    Set objXMLHTTP = New MSXML2.XMLHTTP60
    With objXMLHTTP
        .Open strMethod, strURL, False
        .setRequestHeader "Content-Type", "application/json"
        .setRequestHeader "Accept", "application/json"
        .setRequestHeader "Authorization", "Bearer " + cstrAPIKey
        .send strRequest
        Select Case .status
            Case 200
                Request = True
                strResponse = .responseText
            Case Else
                MsgBox "Fehler beim Request:" & vbCrLf & .statusText & vbCrLf & .responseText
        End Select
    End With
End Function
```

Listing 1: Basisfunktion für den Zugriff auf die REST-API von lexware

Autorisierung für den Zugriff auf die Daten unseres Kontos bei **lexoffice**. Hier übergeben wir einen Ausdruck, der aus dem Text **Bearer** und dem zuvor ermittelten API-Schlüssel besteht. Die **send**-Methode schickt die Anfrage schließlich an die REST-API. Sie erhält noch die in **strRequest** angegebenen zusätzlichen Daten für die Anfrage.

Liefert die Anfrage mit der Eigenschaft status den Wert 200 zurück, war die Anfrage erfolgreich. Dann erhält die **Request**-Funktion als Rückgabewert den Wert **True** und wir füllen den Parameter **strResponse** mit der in der Eigenschaft **responseText** enthaltenen JSON-Antwort.

Anderenfalls gibt die Funktion die Fehlernummer und den Statustext in einem Meldungsfenster aus.

Kundendaten abfragen

Als Fingerübung fragen wir als Erstes alle Kundendaten ab. Dazu verwenden wir einen Aufruf der Funktion **Request** mit der folgenden URL:

```
https://api.lexoffice.io/v1/contacts
```

Nach dem Aufruf der Funktion **Request** geben wir den Inhalt von **strResponse** im Direktbereich des VBA-Editors aus und speichern das Ergebnis einer Funktion namens **GetJSONDOM** mit der Hilfsfunktion **Zwischenablage** aus dem Modul **mdlZwischenablage** in der Zwischenablage:

```
Public Sub FilteringContacts_All()
    Dim strRequest As String
    Dim strURL As String
    Dim strResponse As String
    strURL = "https://api.lexoffice.io/v1/contacts"
    If Request(strURL, "GET", strRequest, _
            strResponse) = True Then
        Debug.Print strResponse
        InZwischenablage GetJSONDOM(strResponse, _
            True, True)
    End If
End Sub
```

Mit der Response allein können wir nicht viel anfangen. Diese sieht in gekürzter Form wie folgt aus:

```
{"content":[{"id":"3a856a88-ee3a-421a-
9850-1fce2d01c8f0","organizationId":"dd8bf92c-43e5-41bf-
a57a-ba7b33eb980f","version":4,"roles":{"customer":{"num-
ber":10001}},"person":{"salutation":"Herr","firstNa-
me":"André","lastName":"Minhorst"},"addresses":{"bil-
ling":[{"supplement":"","street":"Borkhofer Str.
17","zip":"47137","city":"Duisburg","countryCo-
de":"DE"}],"shippin
g":[{"supplement":"","street":"Borkhofer Str.,
17","zip":"47137","city":"Duisburg","countryCo-
de":"DE"}]},"emailAddresses":{"other":["andre@minhorst.
com"]},"phoneNumbers":{"other":["98989898"]},"no-
te":"24284","archived":false},...}
```

Die Funktion **GetJSONDOM** haben wir im Rahmen des Beitrags **JSON-Daten auslesen** (**www.access-im-unternehmen.de/1403**) entwickelt. Sie gibt die im JSON-Dokument enthaltenen Daten wie in Listing 2 aus.

Diese Ausgabe dient als Vorlage für das systematische Durchlaufen der Daten, die zu diesem Zweck in ein Objektmodell bestehend aus Collections und Dictionaries überführt wurde – mehr dazu im oben genannten Beitrag. Um beispielsweise die E-Mail-Adressen aller Kunden im Direktbereich auszugeben, benötigen wir damit nur noch die folgenden Zeilen zur Prozedur **FilteringContacts_All** hinzuzufügen – und zwar als letzte Zeile innerhalb der **If... Then**-Bedingung:

```
AusgabeKontakte strResponse
```

Die Ausgabe der Nachnamen aller Personen würde dann wie folgt geschehen:

```
Public Sub AusgabeKontakte(strJSON As String)
    Dim objJSON As Object
    Dim objContact As Object
    Set objJSON = ParseJson(strJSON)
```

```
objJson.Item("content").Item(3).Item("id"): 3a856a88-ee3a-421a-9850-1fce2d01c8f0
objJson.Item("content").Item(3).Item("organizationId"): dd8bf92c-43e5-41bf-a57a-ba7b33eb980f
objJson.Item("content").Item(3).Item("version"): 4
objJson.Item("content").Item(3).Item("roles").Item("customer").Item("number"): 10001
objJson.Item("content").Item(3).Item("person").Item("salutation"): Herr
objJson.Item("content").Item(3).Item("person").Item("firstName"): André
objJson.Item("content").Item(3).Item("person").Item("lastName"): Minhorst
objJson.Item("content").Item(3).Item("addresses").Item("billing").Item(1).Item("supplement"):
objJson.Item("content").Item(3).Item("addresses").Item("billing").Item(1).Item("street"): Borkhofer Str. 17
objJson.Item("content").Item(3).Item("addresses").Item("billing").Item(1).Item("zip"): 47137
objJson.Item("content").Item(3).Item("addresses").Item("billing").Item(1).Item("city"): Duisburg
objJson.Item("content").Item(3).Item("addresses").Item("billing").Item(1).Item("countryCode"): DE
objJson.Item("content").Item(3).Item("addresses").Item("shipping").Item(1).Item("supplement"):
objJson.Item("content").Item(3).Item("addresses").Item("shipping").Item(1).Item("street"): Borkhofer Str., 17
objJson.Item("content").Item(3).Item("addresses").Item("shipping").Item(1).Item("zip"): 47137
objJson.Item("content").Item(3).Item("addresses").Item("shipping").Item(1).Item("city"): Duisburg
objJson.Item("content").Item(3).Item("addresses").Item("shipping").Item(1).Item("countryCode"): DE
objJson.Item("content").Item(3).Item("emailAddresses").Item("other").Item(1): andre@minhorst.com
objJson.Item("content").Item(3).Item("phoneNumbers").Item("other").Item(1): 98989898
objJson.Item("content").Item(3).Item("note"): 24284
objJson.Item("content").Item(3).Item("archived"): Falsch
```

Listing 2: Ausgabe für den Zugriff auf die Daten einer JSON-Antwort

```
    For Each objContact In objJSON.Item("content")
        On Error Resume Next
        Debug.Print objContact.Item("person").Item("lastName")
        On Error GoTo 0
    Next objContact
End Sub
```

Die Prozedur speichert das Objektmodell des JSON-Dokuments als Struktur aus Dictionaries und Collections, auf die wir über die entsprechenden Auflistungen zugreifen können. Jedes **content**-Element enthält einen der Kontakte. Diese durchlaufen wir in einer **For Each**-Schleife und referenzieren das **content**-Element dabei mit der Variablen **objContact**.

Über dieses können wir dann beispielsweise mit dem Ausdruck **objContact.Item("person").Item("lastName")** auf den Nachnamen der Person zugreifen. Da nicht sichergestellt ist, dass diese Eigenschaft für jeden Kontakt existiert, haben wir für den Zugriff die eingebaute Fehlerbehandlung deaktiviert. Damit können wir nun alle Kontakte und die vorhandenen Eigenschaften durchlaufen und diese beispielsweise in einer Tabelle der Datenbank speichern.

Einen bestimmten Kontakt auslesen

Wenn Sie nicht alle Kontakte auslesen wollen, sondern nur einen bestimmten, können Sie verschiedene Kriterien verwenden. Eines davon ist die E-Mail-Adresse. In der Prozedur **FilteringContacts_ByEMail**, die Sie im Modul **mdlLexOffice** finden, fügen wir dazu an die URL noch einen Filterparameter an:

```
strURL = "https://api.lexoffice.io/v1/contacts?email=andre@minhorst.com"
```

Dies liefert alle Kunden mit der E-Mail-Adresse **andre@minhorst.com**. Wir können die folgenden Vergleichsfelder nutzen:

- **email**: Filtert nach der E-Mail-Adresse.
- **name**: Filtert nach dem Namen.

- **number**: Filtert nach der Kundennummer, die beispielsweise in der Eigenschaft **roles/customer/number** enthalten ist.
- **customer**: Filtert nach allen Kontakten, welche die Rolle **customer** haben.
- **vendor**: Filtert nach allen Kontakten, welche die Rolle **vendor** haben.

Die Dokumentation des passenden API-Befehls finden Sie unter folgender URL:

```
https://developers.lexoffice.io/docs/#contacts-endpoint-filtering-contacts
```

Wenn wir beispielsweise nach dem Namen filtern wollen und dieser Sonderzeichen enthält wie in André, dann müssen wir diesen zuvor enkodieren. Funktionen dazu stellen wir im Beitrag **URLs kodieren per VBA** (**www.access-im-unternehmen.de/1423**) vor.

Zur Verwendung der dortigen Funktion **URLEncode_UTF8** benötigen wir noch einen Verweis auf die Bibliothek **Microsoft ActiveX Data Objects 6.1 Library**.

Die URL stellen wir dann wie folgt zusammen:

```
strURL = "https://api.lexoffice.io/v1/contacts?name=" & URLEncode_UTF8("André")
```

Kontakt nach der ID in lexoffice ermitteln

Wir können auch nach der ID eines Kontakts filtern, der in der Eigenschaft **id** gespeichert ist. Diesen übergeben wir ohne Feldname, also beispielsweise wie folgt:

```
strURL = "https://api.lexoffice.io/v1/contacts/xxxxxxxx-ee3a-421a-9850-1fce2d01c8f0"
```

Hier ist jedoch zu beachten, dass keine Liste von Kontakten zurückgeliefert wird, sondern nur das Kontakt-Objekt selbst. Wir würden also nicht wie zuvor auf den Kontakt zugreifen:

```
objJson.Item("content").Item(3).Item("person").Item("lastName")
```

Stattdessen greifen wir so darauf zu:

```
objJSON.Item("lastName")
```

Das JSON-Dokument ist in diesem Fall anders strukturiert, als wenn wir auf alle Kontakte zugreifen oder gefiltert nach der E-Mail-Adresse. Um die Daten aus dem JSON-Dokument auszulesen, kann man sich die Struktur für den Zugriff mit der Funktion **GetJSONDOM** ansehen.

Die Dokumentation des passenden API-Befehls finden Sie unter folgender URL:

```
https://developers.lexoffice.io/docs/#contacts-endpoint-retrieve-a-contact
```

Kundendaten aus der Datenbank nach lexoffice übertragen

Damit nähern wir uns der eigentlichen Aufgabe: Wir wollen die Kundendaten aus unserer Bestellverwaltung nach **lexoffice** übertragen. Dazu nutzen wir wieder die gleiche URL, diesmal allerdings mit der Methode **POST**. Außerdem müssen wir mit dem Parameter **strRequest** ein JSON-Dokument übergeben, welches die Daten des anzulegenden Kontakts enthält.

Hinzu kommt, dass wir eine Verbindung zwischen dem Kundendatensatz in unserer Rechnungsverwaltung und dem Kontakt in **lexoffice** herstellen wollen. **lexoffice** erlaubt leider keine Übermittlung der Kundennummer und erstellt eine eigene Kundennummer. Also bleibt als einzige Möglichkeit, eine Verbindung zwischen dem Kontakt in **lexoffice** und dem Kundendatensatz in der Datenbank herzustellen, das Auslesen der ID des Kontakts beim Anlegen und das Speichern dieser ID in der Kundentabelle. Dazu

Formatassistent für Textfelder, Teil 2

Im ersten Teil dieser Beitragsreihe haben wir einen Assistenten vorgestellt, mit dem Sie die Format-Eigenschaft von Feldern in Tabellen, Abfragen, Formularen und Berichten einstellen können. Diesem Assistenten fügen wir noch eine weitere nützliche Funktion hinzu. Dabei wollen wir die Daten, die in diesem Feld angezeigt werden, auslesen und im Formatassistenten anzeigen, damit der Benutzer das Ergebnis der Formatierung direkt anhand seiner eigenen Daten betrachten kann. Dazu ist, wenn wir auf die Daten der gerade angezeigten Tabelle zugreifen wollen, noch ein kleiner Trick nötig, denn wir können nicht auf die Daten zugreifen, solange die Tabelle in der Entwurfsansicht geöffnet ist. Bei Abfragen, Formularen und Berichten ist dies etwas einfacher – aber sehen Sie selbst!

Beispieldaten aus dem Feld einer Tabelle oder Abfrage beziehen

Nachdem wir ein wenig mit dem Format-Assistenten experimentiert haben, fiel uns auf, dass es eigentlich keinen richtigen Spaß macht, mit vorgegebenen Beispieldaten zu arbeiten beziehungsweise diese manuell anzupassen.

Viel cooler wäre es doch, wenn wir die Daten direkt aus dem zu formatierenden Feld beziehen könnten. Also fügen wir für die Auswahl der Beispieldaten noch eine Schaltfläche hinzu, mit der wir ein weiteres Formular öffnen können. Dieses soll alle Tabellen und Abfragen der aktuellen Datenbank zur Auswahl anbieten.

Nach der Auswahl der Tabelle oder Abfrage wollen wir dann die Felder der gewählten Datenquelle anbieten, um aus diesen das Feld mit den zu untersuchenden Daten auszuwählen.

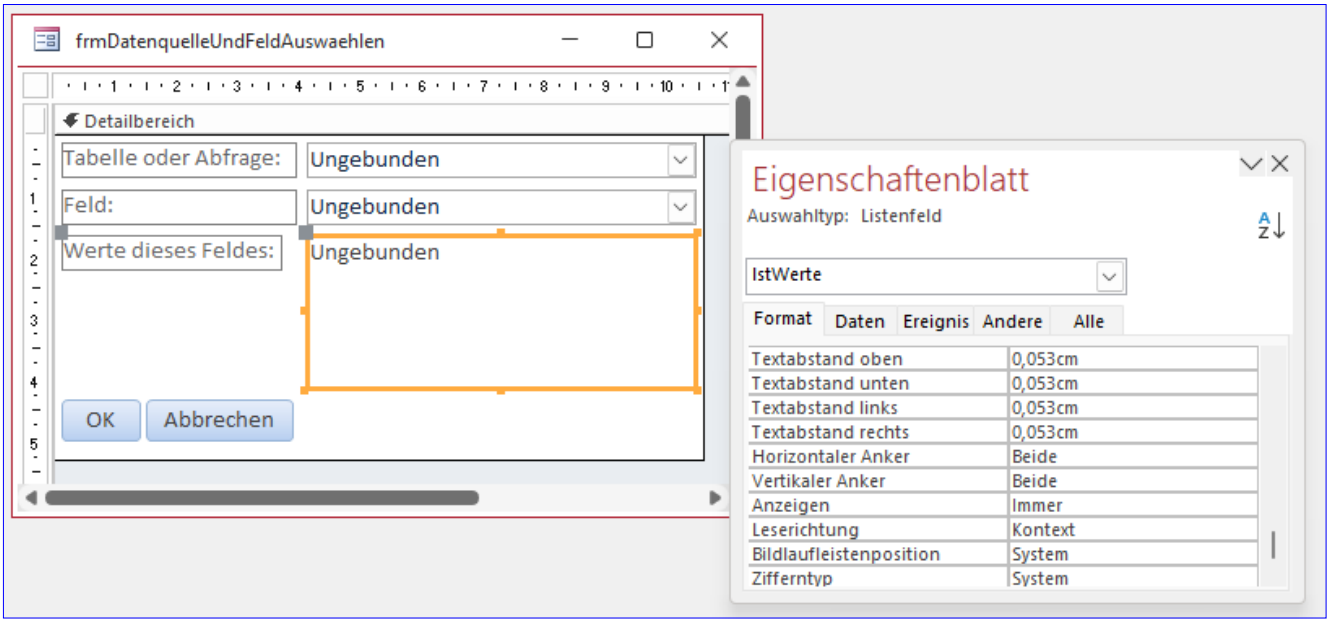


Bild 1: Formular zum Auswählen der Daten, die in der Format-Vorschau angezeigt werden sollen

Das Formular heißt **frmDatenquellenUndFeldAuswaehlen** und es enthält zwei Kombinationsfelder namens **cboTabelleOderAbfrage** und **cboFeld**. Zur Sicherheit sollen die Werte des gewählten Feldes auch noch in einem Listenfeld namens **lstWerte** angezeigt werden. Außerdem fügen wir zwei Schaltflächen namens **cmdOK** und **cmdAbbrechen** hinzu (siehe Bild 1).

Für diese hinterlegen wir jeweils eine Ereignisprozedur. Die Schaltfläche **cmdOK** soll das Formular ausblenden, damit wir die gewählten Daten von der aufrufenden Prozedur aus auswerten können:

```
Private Sub cmdOK_Click()
    Me.Visible = False
End Sub
```

Die Schaltfläche **cmdAbbrechen** soll das Formular einfach schließen:

```
Private Sub cmdAbbrechen_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Öffnen des Formulars zum Auswählen der zu formatierenden Daten

Bevor wir uns die Funktion des Formulars **frmDatenquelleUndFeldAuswaehlen** genauer anschauen, werfen wir einen Blick auf die Schaltfläche **cmdBeispielwerteEinlesen** im aufrufenden Formular **frmFormatassistent**, mit der wir das Formular **frmDatenquelleUndFeldAuswaehlen** öffnen (siehe Bild 2).

Diese Prozedur heißt **cmdBeispielwerteEinlesen_Click** und ist in Listing 1 zu finden. Sie schließt zunächst sicherheitshalber das Formular **frmDatenquelleUndFeldAuswaehlen**, falls dieses noch geöffnet sein sollte. Der Hintergrund ist, dass nur beim erneuten Öffnen per **DoCmd.Open** auch die Parameter dieser Methode angewendet werden. Das Formular **frmDatenquelleUndFeldAuswaehlen** soll als modaler Dialog geöffnet werden. Außerdem wollen wir einige Informationen als Öffnungsargument übergeben.

Wann immer wir hier mehr als eine Information übermitteln, können wir diese durch das Pipe-Zeichen (**|**) getrennt übergeben und diesen Ausdruck im aufgerufenen Formular wieder auseinandernehmen. In diesem Fall übergeben wir den Namen des Objekts (also Tabelle, Abfrage, Formular oder Bericht), den Namen des Feldes beziehungsweise Steuerelements und den Objekttyp.

Diese Informationen verarbeitet das Formular direkt in der **Beim Öffnen**-Ereignisprozedur – was dort geschieht, lesen Sie weiter unten. Wir schauen uns nun noch an, was nach der Eingabe der gewünschten Daten und dem Schließen des Formulars **frmDatenquelleUndFeldAuswaehlen** durch den Benutzer in der aufrufenden Prozedur noch geschieht. Diese prüft, ob das Formular noch geöffnet ist, was nur der Fall ist, wenn der Benutzer

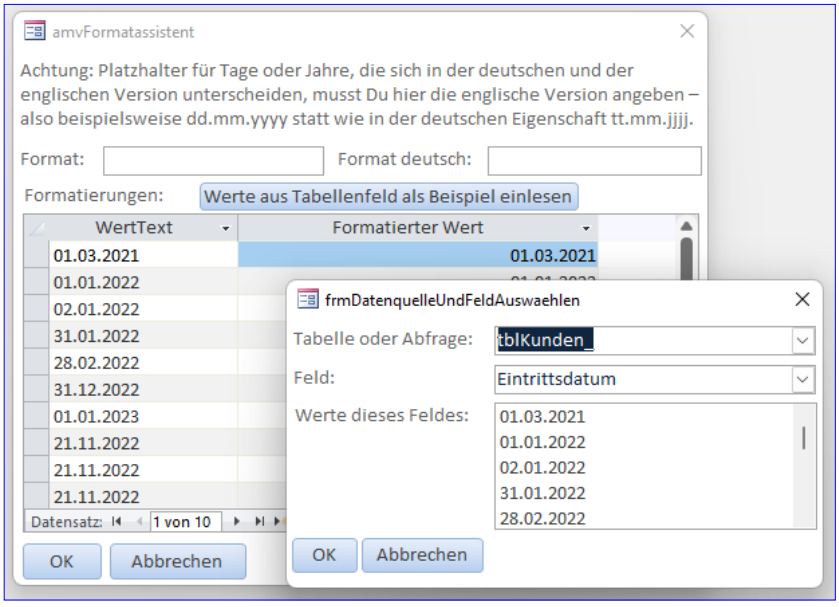


Bild 2: Öffnen des Formulars **frmDatenquelleUndFeldAktualisieren**

```
Private Sub cmdBeispielwerteEinlesen_Click()
    Dim strForm As String
    Dim strTabelleOderAbfrage As String
    Dim strFeld As String
    strForm = "frmDatenquelleUndFeldAuswaehlen"
    On Error Resume Next
    DoCmd.Close acForm, strForm
    On Error GoTo 0
    DoCmd.OpenForm strForm, WindowMode:=acDialog, OpenArgs:=strObject & "|" & strControl & "|" & lngObjectType
    If IstFormularGeoeffnet(strForm) Then
        strTabelleOderAbfrage = Forms(strForm)!cboTabelleOderAbfrage
        strFeld = Forms(strForm)!cboFeld
        DoCmd.Close acForm, strForm
    End If
End Sub
```

Listing 1: Aufruf des Formulars frmDatenquelleUndFeldAuswaehlen

dort die **OK**-Schaltfläche betätigt hat. Ist das der Fall, aktualisiert die Prozedur die Daten im Unterformular und schließt dann das Formular **frmDatenquelleUndFeldAuswaehlen**.

Damit kommen wir nun zur Programmierung des Formulars **frmDatenquellenUndFeldAuswaehlen**.

Tabellen und Abfragen im Kombinationsfeld anzeigen

Für das Füllen der Kombinationsfelder mit den Tabellen oder Abfragen der Datenbank, von der aus wir den Assistenten aufgerufen haben, brauchen wir eine Liste der entsprechenden Elemente. Das ist insofern kompliziert, als das wir nicht einfach die Eigenschaft **Datensatzherkunft** auf die Tabelle **MSysObjects** einstellen können, die uns alle Tabellen und Abfragen liefern würde.

Damit würden wir nämlich auf die entsprechende Tabelle der Add-In-Datenbank zugreifen und nicht auf die der Host-Datenbank. Also erstellen wir in der gleich beschriebenen Ereignisprozedur, die durch das Ereignis **Beim Öffnen** des Formulars **frmDatenquelleUndFeldAuswaehlen** ausgelöst wird, ein Recordset auf Basis der entsprechenden Tabelle und weisen dieses dem Kombinationsfeld über seine Eigenschaft **Recordset** zu.

Als Datensatzherkunft für das Recordset des Steuerelements **cboTabelleOderAbfrage** verwenden wir die folgende Abfrage:

```
SELECT Name, Type FROM MSysObjects WHERE Type IN (1, 4, 5, 6) AND Not Name LIKE '~*' AND Not Name LIKE 'MSys*' AND NOT Name LIKE 'USys*' AND NOT Name LIKE 'f_*' ORDER BY Name
```

Diese ermittelt die Namen und den Typ von Tabellen aller Typen (**1**, **4**, **6**) und von Abfragen (**5**). Außerdem schließt sie Systemobjekte und einige weitere Objekte aus, für die der Benutzer vermutlich nicht die **Format**-Eigenschaft einstellen möchte. Und die Abfrage liefert nicht nur den Objektnamen, sondern auch noch den Objekttyp des jeweiligen Datenbankobjekts.

Insgesamt sieht die Prozedur wie in Listing 2 aus. Die Prozedur schreibt den SQL-Ausdruck für den Zugriff auf die Tabelle **MSysObjects** in die Variable **strSQL**. Dann erstellt sie ein Recordset auf Basis dieser Abfrage und weist dieses anschließend der Eigenschaft **Recordset** des Kombinationsfeldes **cboTabelleOderAbfrage** zu.

Danach parst sie die mit dem Öffnungsargument übergebenen Daten. Dieses sieht beispielsweise wie folgt aus:

```
tblKunden|Eintrittsdatum|0
```

Diese Daten liest die Prozedur nun durch Aufteilen mit der **Split**-Funktion und anschließendes Abgreifen der Elemente des damit erzeugten Arrays ein und speichert sie in den Variablen **strObject**, **strControl** und **intObjectType**.

Dann prüft sie den Wert von **intObjectType**. Handelt es sich um eine Tabelle oder Abfrage (**acTable** oder **acQuery**), weist sie dem Kombinationsfeld cboTabelleOderAbfrage den entsprechenden Eintrag zu, also den Namen der mit dem Öffnungsargument übergebenen Tabelle oder Abfrage.

Dann ruft sie die Prozedur FelderAktualisieren auf, die wir gleich im Anschluss beschreiben.

Dem Kombinationsfeld **cboFeld** weist die Prozedur den Namen des übergebenen Feldes zu. Anschließend ruft die Prozedur die Routine **DatenAnzeigen** auf.

Im Falle eines Formulars oder Berichts wird einfach das Kombinationsfeld **cboTabelleOderAbfrage** aktiviert und aufgeklappt.

Aktualisieren der auszuwählenden Felder

Damit die Datensatzherkunft des zweiten Kombinationsfelds **cboFeld** das mit dem Öffnungsargument übergebene Feld überhaupt enthält, müssen diese nach der Festlegen des Eintrags im Kombinationsfeld **cboTabelleOderAbfrage** aktualisieren. Das erledigen wir durch den Aufruf der Prozedur **FelderAktualisieren** (siehe Listing 3).

```
Private Sub Form_Open(Cancel As Integer)
    Dim db As DAO.Database
    Dim rstTabellenUndAbfragen As DAO.Recordset
    Dim strSQL As String
    Set db = CurrentDb
    strSQL = "SELECT Name, Type FROM MSysObjects WHERE Type IN (1, 4, 5, 6) AND Not Name LIKE '~*' " _
        & "AND Not Name LIKE 'MSys*' AND NOT Name LIKE 'USys*' AND NOT Name LIKE 'f_*' ORDER BY Name"
    Set rstTabellenUndAbfragen = db.OpenRecordset(strSQL, dbOpenDynaset)
    Set Me!cboTabelleOderAbfrage.Recordset = rstTabellenUndAbfragen
    If Not IsNull(Me.OpenArgs) Then
        strObject = Split(Me.OpenArgs, "|")(0)
        strControl = Split(Me.OpenArgs, "|")(1)
        intObjectType = Split(Me.OpenArgs, "|")(2)
    End If
    Select Case intObjectType
        Case acTable, acQuery
            Me!cboTabelleOderAbfrage = strObject
            FelderAktualisieren
            Me!cboFeld = strControl
            DatenAnzeigen
        Case acForm, acReport
            Me!cboTabelleOderAbfrage.SetFocus
            Me!cboTabelleOderAbfrage.Dropdown
    End Select
End Sub
```

Listing 2: Diese Prozedur wird beim Öffnen des Formulars **frmDatenquellenUndFeldAuswaehlen** aufgerufen.

Access-Add-In per Knopfdruck erstellen

Immer, wenn ich ein neues Add-In für eine Automatisierung meiner Access-Entwicklung erstelle, führe ich die gleichen Schritte durch: Anlegen einer neuen Datenbank, Speichern als .accda, Erstellen der Tabelle USysRegInfo, Hinzufügen der Daten für die Registry, Einstellen der Datenbankinformationen, sodass diese im Add-In-Manager angezeigt werden und Programmieren der Funktion, die beim Start des Add-Ins aufgerufen werden soll. Und erst danach fängt der eigentliche Spaß an, nämlich die Programmierung der Funktion, die ich gern mit dem Add-In erledigen möchte. Weil ich das oft genug gemacht habe, baue ich im Rahmen dieses Beitrags ein Tool, mit dem ich solche Add-Ins per Mausklick erstellen kann – beziehungsweise ein Add-In, mit dem ich die aktuelle Datenbank in ein Add-In umwandeln kann.

Wenn ich schon regelmäßig Access-Add-Ins programmiere, um damit lästige, immer wieder auftretende Aufgaben zu vereinfachen – warum sollte ich dann nicht das Zusammenstellen der wichtigsten Elemente dieses Add-Ins auch automatisieren?

Ich habe erst überlegt, mir eine Vorlage zu erstellen, die ich dann kopiere und anpasse. Aber letztlich sind doch einige Schritte zu erledigen.

Selbst wenn die Tabelle **USysRegInfo** in einer solchen Vorlage bereits vorhanden ist, muss ich die dortigen Daten noch einstellen, und auch die Informationen der Access-Datenbank selbst, die später im Add-In-Manager angezeigt werden, müssen immer wieder angepasst werden.

Davon abgesehen schaffe ich es regelmäßig, die Daten in der Tabelle **USysRegInfo** nicht korrekt einzugeben, und selbst der kleinste Fehler führt dazu, dass die Registrierung nicht durchgeführt wird oder das Access-Add-In anschließend dennoch nicht gestartet werden kann.

Viel einfacher wäre es doch, wenn es ein Formular gäbe, das die wichtigsten Einstellungen vor dem Erstellen des Access-Add-Ins abfragt und diese automatisch anlegt.

Neues Access-Add-In oder aktuell geöffnete Datenbank umrüsten?

Nun stellt sich die Frage, wie wir das Erstellen eines neuen Access-Add-Ins gestalten wollen. Wir könnten beispielsweise eine herkömmliche Access-Datenbank erstellen, die ein Formular enthält, welches die Parameter für das Access-Add-In abfragt und diese dann auf die gleiche Datenbank anwendet.

In dieser Datenbank könnten wir die Tabelle **USysRegInfo** dann bereits vorbereiten und brauchen diese nur noch auf Basis der in das Formular eingegebenen Daten anzupassen – das gilt auch für die Datenbankinformationen.

Die Alternative wäre, wiederum ein Access-Add-In zu programmieren, das der aktuell geöffneten Datenbank die notwendigen Elemente hinzufügt. Das Access-Add-In müsste dann die notwendigen Informationen abfragen, die Tabelle **USysRegInfo** anlegen und mit den Daten für die Registrierung füllen, die Datenbankinformationen eintragen und das Modul mit der Funktion zum Starten des Access-Add-Ins anlegen.

Mein Weg zu Access-Add-Ins

Ich gehe an dieser Stelle einmal von meiner eigenen Vorgehensweise beim Erstellen eines Access-Add-Ins

aus. Diese sieht so aus, dass ich zu Beginn meist gar kein Access-Add-In plane, sondern einfach nur in irgendeiner Datenbankanwendung, die ich gerade programmiere, eine VBA-Prozedur hinzufüge, mit der ich eine Aufgabe automatisiere.

Diese setze ich dann ein, stelle fest, dass sie gut funktioniert, und füge dann gegebenenfalls ein Formular hinzu, von dem ich die Prozedur aufrufe.

Wenig später stelle ich dann fest, dass ich diese Funktion auch noch in weiteren Projekten nutzen könnte. Meistens kopiere ich diese dann zuerst in das betroffene Projekt und nutze es dort. Irgendwann kommt mir dann die Idee, dass ich ja auch ein Access-Add-In aus der Funktion machen könnte – und dann beginnen die Schritte, die ich bereits eingangs erwähnt habe.

Ich starte also eher mit einer Datenbankanwendung, in der bereits die Funktionen des zukünftigen Access-Add-Ins enthalten sind und wandle diese dann in ein Add-In um. Dementsprechend legen wir die Lösung dieses Beitrags auch als Access-Add-In an – um damit über das Add-In-Menü die aktuell geöffnete Datenbank in ein Add-In umwandeln zu können.

Formular zum Erfassen der Add-In-Daten

Bevor wir loslegen, benötigen wir ein Formular, mit dem der Benutzer die Daten eingeben kann, die für das Umrüsten des Add-Ins benötigt werden. Dieses Formular sieht in der Entwurfsansicht wie in Bild 1 aus. Wir verwenden hier die folgenden Textfelder:

- **txtAddInName**: Diese Bezeichnung wird in der Liste der Add-Ins im Ribbon verwendet sowie als Schlüssel in der Registry.

- **txtBeschreibung**: Die Beschreibung wird in die Eigenschaft Kommentare der Datenbank geschrieben. Sie wird im Add-In-Manager angezeigt, wenn das Add-In darin markiert wurde.

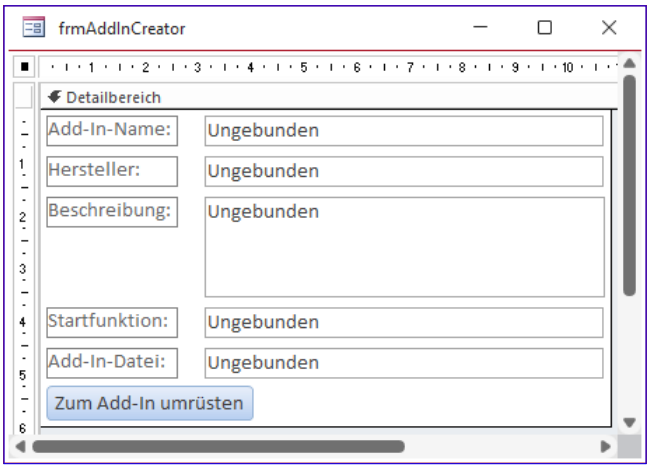


Bild 1: Entwurf des Formulars zur Eingabe der Add-In-Daten

- **txtHersteller**: Der Hersteller wird in die Eigenschaft **Firma** der Add-In-Datenbank geschrieben. Auch diese Information landet im Add-In-Manager.

- **txtStartfunktion**: Name der VBA-Funktion, die beim Starten des Add-Ins aufgerufen werden soll.

- **txtAddInDatei**: Aktueller Dateiname, bei dem gegebenenfalls noch die Dateiendung **.accdb** durch **.accda** ersetzt wird. Der tatsächliche Dateiname muss dem hier angegebenen Dateinamen entsprechen.

Damit gleich beim Öffnen des Formulars einige Beispieldaten erscheinen, füllen wir die Textfelder beim Laden des Formulars mit der folgenden Ereignisprozedur:

```
Private Sub Form_Load()
    Me!txtAddInName = "Add-In-Name"
    Me!txtBeschreibung = _
        "Dies ist die Beschreibung des Add-Ins."
    Me!txtHersteller = "Hersteller"
    Me!txtStartfunktion = "Autostart"
    Me!txtAddInDatei = _
        Replace(CurrentProject.Name, ".accdb", ".accda")
End Sub
```

Dadurch sieht das Formular nach dem Öffnen wie in Bild 2 aus.

Umwandeln der Access-Datenbank in ein Access-Add-In

Damit wir die Access-Datenbank als Access-Add-In nutzen können, sind einige Schritte nötig, die im Code der Prozedur, die durch die Schaltfläche **cmdZumAddInUmruesten** bereits skizziert sind (siehe Listing 1).

Die Prozedur deklariert und füllt zwei Database-Variablen: Die erste heißt **db** und referenziert die aktuell geöffnete Datenbank (mit **CurrentDb**), die zweite heißt **dbc** und bezieht sich auf die Add-In-Datenbank (mit **CodeDb**).

Bedenken Sie: Die Datenbank, die wir gerade programmieren, wird ja selbst ein Access-Add-In, mit dem wir die dann geladene Datenbank in ein Access-Add-In umwandeln wollen.

Die Prozedur ruft danach weitere Routinen auf. Die erste heißt **CreateUSysRegInfo** und wird mit dem Verweis auf die aktuell geöffnete Datenbank als Parameter aufgerufen. Sie hat die Aufgabe, die Tabelle **USysRegInfo** zu erstellen, welche die Informationen aufnimmt, die beim Installieren

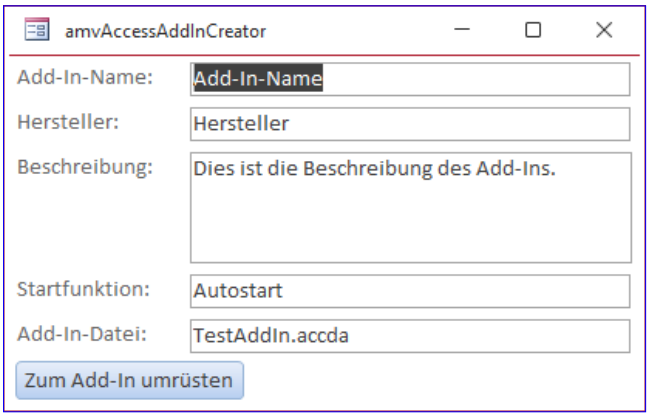


Bild 2: Das Formular zur Eingabe der Add-In-Daten in der Formularansicht

des Access-Add-Ins in die Registry geschrieben werden sollen, damit Access beim Öffnen auslesen kann, welche Access-Add-Ins verfügbar sind.

Diese Funktion prüft auch, ob die Tabelle **USysRegInfo** bereits vorhanden ist – falls ja, wird der Vorgang abgebrochen.

Der Aufruf der Prozedur **AddRegistryEntries** übergibt Verweise auf die aktuelle Datenbank, das aktuelle Ac-

```
Private Sub cmdZumAddInUmruesten_Click()
    Dim db As dao.Database
    Dim dbc As dao.Database
    Set db = CurrentDb
    Set dbc = CodeDb
    If Not CreateUSysRegInfo(db) Then
        MsgBox "Offensichtlich ist bereits eine Tabelle namens 'USysRegInfo' vorhanden." & vbCrLf _
            & "Prüfen Sie die aktuelle Datenbank." & vbCrLf & vbCrLf _
            & "Der Vorgang wird abgebrochen.", vbOKOnly + vbExclamation
        Exit Sub
    End If
    AddRegistryEntries db, dbc, Me!txtAddInName, 1, Me!txtStartfunktion, Me!txtBeschreibung, Me!txtAddInDatei
    ChangeProperties db, Me!txtAddInName, Me!txtBeschreibung, Me!txtHersteller
    AddStartfunction db, Me!txtStartfunktion
    If Not Right(CurrentProject.Name, 6) = ".accda" Then
        MsgBox "Die Dateiendung muss noch in .accda geändert werden.", vbOKOnly + vbExclamation, "Dateiendung anpassen"
    End If
End Sub
```

Listing 1: Code zum Umwandeln der aktuell geöffneten Datenbank in ein Access-Add-In

cess-Add-In, den Namen des zu erstellenden Add-Ins, den Wert **1** als Parameter für den zu erstellenden Add-In-Typ (aktuell gibt es nur den einen), den Namen der beim Start aufzurufenden Funktion, die Beschreibung und den Dateinamen der zukünftigen Add-In-Datenbank. Sie soll die benötigten Daten in die Tabelle **USysRegInfo** schreiben.

Die Prozedur **ChangeProperties** soll die Eigenschaften der Datenbankdatei ändern, die später im Add-In-Manager angezeigt werden. Dazu erhält sie als Parameter den Verweis auf die zu ändernde Datenbank, den Namen des Add-Ins, die Beschreibung und den Hersteller.

Schließlich folgt noch die Prozedur **AddStartfunction**, der wir den Verweis auf die Datenbank übergeben, welche um ein Modul mit dieser Funktion erweitert werden soll – und den Namen der anzulegenden Funktion.

Schließlich prüft die Prozedur noch, ob die Zieldatenbank bereits die Dateiendung **.accda** hat und weist den Benutzer darauf hin, diesen gegebenenfalls anzupassen.

Erstellen der Tabelle USysRegInfo

Die erste Funktion namens **CreateUSysRegInfo** erwartet einen Verweis auf die in ein Access-Add-In umzuwandelnde Datenbank als Parameter (siehe Listing 2). Sie versucht zunächst, eine bereits bestehende Tabelle namens **USysRegInfo** zu referenzieren. Gelingt dies, wird die Funktion mit dem Ergebnis **False** abgebrochen, damit keine eventuell bereits angelegten Elemente überschrieben werden.

Ist die Tabelle jedoch noch nicht vorhanden, legt die Prozedur diese mit der Methode **CreateTableDef** des **Database**-Objekts an und legt dabei direkt den Namen **USysRegInfo** fest. Danach fügt sie durch mehrfaches

```
Public Function CreateUSysRegInfo(dbAddIn As dao.Database) As Boolean
    Dim tdf As dao.TableDef
    Dim fld As dao.Field
    On Error Resume Next
    Set tdf = dbAddIn.TableDefs("USysRegInfo")
    If Not tdf Is Nothing Then
        Exit Function
    End If
    On Error GoTo 0
    Set tdf = dbAddIn.CreateTableDef("USysRegInfo")
    With tdf
        Set fld = .CreateField("Subkey", dbText, 255)
        tdf.Fields.Append fld
        Set fld = .CreateField("Type", dbInteger)
        tdf.Fields.Append fld
        Set fld = .CreateField("ValName", dbText, 255)
        tdf.Fields.Append fld
        Set fld = .CreateField("Value", dbText, 255)
        fld.AllowZeroLength = True
        tdf.Fields.Append fld
        dbAddIn.TableDefs.Append tdf
    End With
    CreateUSysRegInfo = True
End Function
```

Listing 2: Code zum Erstellen der Tabelle **USysRegInfo**