

ACCESS

IM UNTERNEHMEN

GOOGLE MAPS

Zeigen Sie Adresdaten mit Google Maps direkt im Access-Formular an (ab S. 37).



In diesem Heft:

EINFACH FILTERN

Erweitern Sie Formulare um eine einfache Funktion zum Speichern und Abrufen benutzerdefinierter Filterkriterien.

DYNAMISCHER ADRESSBLOCK

Nutzen Sie eine VBA-Prozedur, um Adressblöcke per Mausclick zusammenzustellen.

E-MAILS MIT VORLAGE

Schreiben Sie blitzschnell E-Mails auf Basis selbstdefinierter Vorlagen an Kunden.

SEITE 14

SEITE 53

SEITE 24

Adressen sichtbar machen

In fast jeder Datenbank dreht es sich um Adressdaten. Auch unsere Beispiele verwenden meist eine Adresstabelle als Grundlage. Aber meist bleibt es bei den nackten Daten wie Straße, PLZ, Ort und Land. Das wollen wir in dieser Ausgabe ändern: Wir liefern ein Beispiel, wie Sie eine Kundenadresse per Karte visualisieren können.



Als Lieferant für die gewünschten Daten nutzen wir Google-Maps. Die Karten, die Google Maps liefert, kommen meist im Internet Browser daher. Auch wir kommen nicht um das Einbinden eines Browsers herum – aber dank des Webbrowser-Steuerelements ist das ja kein Problem. Im Beitrag **Google-Maps als Kartendienst im Formular** lesen Sie ab S. 37, wie Sie Ihre Anwendung um die Anzeige der im aktuellen Formular dargestellten Adresse als Position in einer Online-Landkarte erweitern.

Adressen spielen auch im Beitrag **Dynamischer Adressblock** ab S. 53 eine Rolle. Für die Anschrift in Berichten oder auch zu anderen Gelegenheiten benötigen Sie die Adressdaten eines Kunden als Adressblock. Im Beitrag erfahren Sie, wie Sie das Aussehen eines solchen Adressblocks konfigurieren und die zu erzeugenden Adressen durch Aufruf einer einfachen VBA-Prozedur zurückerhalten.

Wie Sie diese Adressblöcke nutzen, zeigen wir beispielsweise im Beitrag **Mails mit Vorlage** (ab S. 24). Hier gehen wir davon aus, dass immer wieder Kundenanfragen auf dem Tisch landen, die nicht per E-Mail übertragen werden, sondern etwa telefonisch oder auch per Briefpost oder Fax. Wollen Sie hierauf schnell in Form einer Antwort-E-Mail reagieren, müssen Sie zunächst einmal den Kunden in der Kundendatenbank ausfindig machen, seine E-Mail-Adresse in eine neu erzeugte E-Mail kopieren und dann manuell den Betreff und den Inhalt der E-Mail schreiben. Unser Beitrag zeigt, wie Sie per Schaltfläche direkt in den Kundendetails ein Formular öffnen, das die einfache Auswahl einer Vorlage für die E-Mail an den Kunden ermöglicht, die darin enthaltenen Platzhalter mit den

Daten des Kunden füllt und schließlich eine neue E-Mail auf Basis dieser Daten erstellt. Diese müssen Sie nur noch abschicken.

Oft gefragt ist das Thema Filtern von Daten in der Datenblattansicht von Formularen. Der Benutzer hat zwar einige praktische Möglichkeiten, die Daten direkt über die Benutzeroberfläche von Access zu filtern. Wenn er aber einen aus mehreren Kriterien bestehenden Filter erstellt hat, bleibt dieser nicht bestehen – es gibt auch keine Möglichkeit, einen oder mehrere solcher Filter zu speichern. Also haben wir im Beitrag **Filter speichern und wieder abrufen** ab S. 14 einem Formular die Funktionen hinzugefügt, mit denen er die selbst einstellten Kriterien einfach unter einer selbst gewählten Bezeichnung speichern und wieder abrufen kann. Das spart eine Menge Zeit!

Wenn ein Benutzer Änderungen an einem Datensatz durchführt und dann feststellt, dass er beispielsweise zwei Felder wie PLZ und Ort vertauscht hat, möchte er vielleicht die alten Daten wiederherstellen. Das gelingt allerdings nicht gezielt, da man nur das zuletzt geänderte Feld oder aber den kompletten Datensatz zurücksetzen kann. Im Beitrag **Datensatzänderungen auf einen Blick** (ab S. 2) zeigen wir, wie Sie die Daten vor der Änderung neben den Eingabefeldern anzeigen und dem Benutzer so ermöglichen, diese Daten wieder herzustellen.

Und nun: Viel Spaß beim Lesen!

Ihr Michael Forster

Datensatzänderungen auf einen Blick

Wenn der Benutzer Änderungen an den in einem Formular angezeigten Daten vornimmt, sieht er nur noch die geänderten Daten. Was aber, wenn er beispielsweise einen Wert in ein falsches Feld eingetragen hat – zum Beispiel indem er Vorname und Nachname vertauscht? Dann möchte er die Änderung vermutlich rückgängig machen. Aber wie, wenn er den vorherigen Wert nicht mehr in Erinnerung hat? Also erleichtern wir ihm die Arbeit, indem wir die alten Werte neben den entsprechenden Textfeldern anzeigen.

Die hier vorgestellte Technik bezieht sich natürlich auf Detailformulare – in der Datenblatt- oder Endlosansicht lassen sich Steuerelemente zur Anzeige der vorherigen Daten nur schlecht anzeigen. Wir beginnen also mit einem neuen, leeren Formular namens **frmDatensatzänderungenAufEinemBlick**, dessen Eigenschaft **Datenherkunft** wir mit dem Namen der Tabelle **tblKunden** füllen.

Diese finden Sie bereits in der Beispieldatenbank.

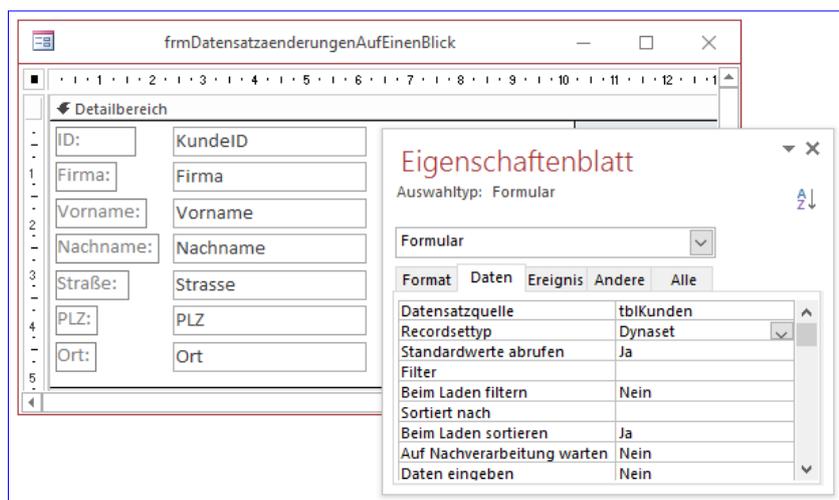


Bild 1: Entwurf des Ausgangsformulars

Anschließend können Sie alle Felder dieser Tabelle aus der Feldliste in den Detailbereich der Entwurfsansicht des Formulars ziehen. Das Ergebnis sieht etwa wie in Bild 1 aus.

Damit das Formular die alten Werte der jeweiligen Felder anzeigen kann, fügen wir für jedes Feld, das der Benutzer bearbeiten kann, ein weiteres Textfeld samt Bezeichnungsfeld hinzu. Die Bezeichnungsfelder erhalten jeweils den Text **Alt:** als Beschriftung. Die Textfelder benennen wir genauso wie die eigentlichen Textfelder – mit dem Unterschied, dass wir den Namen noch das Suffix **_Alt** anhängen.

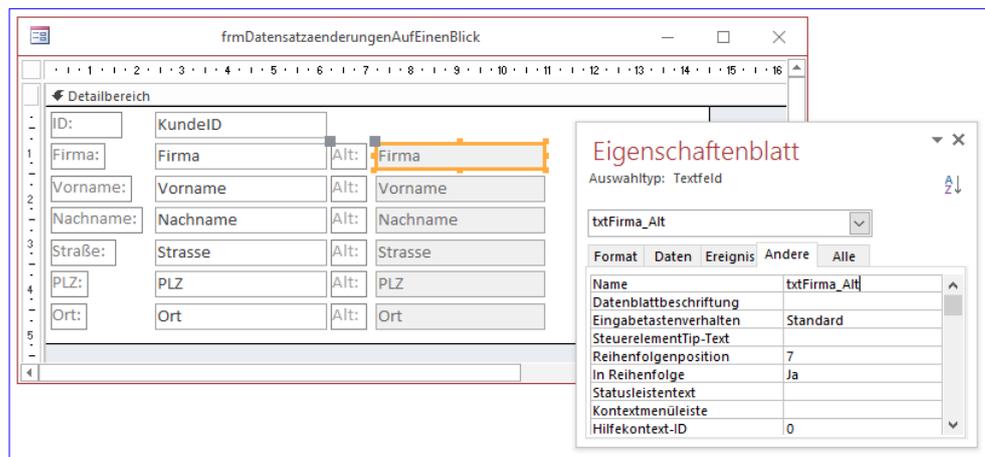


Bild 2: Anlegen der Textfelder für die alten Werte

Das Textfeld zur Anzeige des alten Wertes für das Feld **txtVorname** soll also **txtVorname_Alt** heißen (s. Bild 2).

Hier haben wir nun bereits den Wert **Aktiviert** der Textfelder mit den vorherigen Werten auf **Nein** eingestellt, damit der Benutzer gleich erkennen kann, dass er hier keine Werte eingeben kann.

Es fehlt noch eine Einstellung, denn die Textfelder sollen ja auch erst angezeigt werden, wenn der Benutzer einen anderen als den vorhandenen Wert eingegeben hat.

Also stellen wir die Eigenschaft **Sichtbar** für die Textfelder auf der rechten Seite auf **Nein** ein (Sie können alle Textfelder markieren, indem Sie einen Rahmen aufziehen und dann die Eigenschaft für alle Textfelder gleichzeitig einstellen).

Das Formular sieht nun in der Formularansicht vorerst wie in Bild 3 aus. Beim Ändern eines der Einträge tut sich natürlich noch nichts, denn wir haben ja auch noch keinen entsprechenden Code hinterlegt.

Code zum Einblenden und Füllen der Alt-Felder

Nun haben wir uns das so vorgestellt, dass wir einfach das Ereignis **Bei Geändert** für das jeweilige Steuerelement implementieren und dafür die folgende Ereignisprozedur hinterlegen:

```
Private Sub txtFirma_Dirty(Cancel As Integer)
    With Me!txtFirma_Alt
        .Visible = True
        .Value = Me!txtFirma.
    End With
End Sub
```

Für das Feld **txtFirma** soll also eine Prozedur ausgelöst werden, welche die Eigenschaft **Visible** des Textfeldes **txtFirma_Alt** auf **True** einstellt und den Wert des Textfeldes **txtFirma_Alt** auf den alten Wert (**OldValue**) des Textfeldes **txtFirma**.

Bild 3: Formular in der Formularansicht

Dies klappt allerdings nicht wie gewünscht: Sobald Sie etwa ein Zeichen an den Inhalt des Textfeldes **txtFirma** anfügen wollen (also ohne diesen zu markieren und zu überschreiben), werden zwar die gewünschten Aktionen für das Textfeld **txtFirma_Alt** durchgeführt, aber der Inhalt des Textfeldes **txtFirma** wird komplett durch das soeben eingegebene Zeichen überschrieben (s. Bild 4).

Schauen wir uns an, was zu diesem unerwarteten Verhalten führt. Ein kurzer Test, bei dem wir erst die erste, dann die zweite Anweisung innerhalb der Prozedur ausblenden, zeigt, dass die Zuweisung des alten Wertes an die **Value**-Eigenschaft von **txtFirma_Alt** dafür sorgt, dass der alte Wert von **txtFirma** komplett überschrieben wird.

Hier geschieht Folgendes: Wenn Sie das erste Zeichen in das Textfeld **txtFirma** eingeben, wird bereits die Prozedur

Bild 4: Verhalten bei Eingabe in ein Textfeld

txtFirma_Dirty ausgelöst. Das Textfeld **txtFirma_Alt** wird eingeblendet und erhält als Wert den alten Wert des auslösenden Textfeldes. Bei einer solchen Zuweisung wird immer auch der Fokus auf das aufnehmende Feld verschoben!

Der Fokus gelangt dann zwar wieder zum Feld **txtFirma** zurück, aber beim Eintritt in das Feld wird die Markierung so eingestellt, wie dies standardmäßig beim Eintritt in ein Feld vorgesehen ist.

Cursorverhalten bei Eintritt in Feld

Dieses Verhalten wird normalerweise Access-weit eingestellt, und zwar in den Optionen von Access. Hier finden Sie im Bereich **Clienteneinstellungen** unter **Bearbeiten** die Eigenschaft **Cursorverhalten bei Eintritt in Feld** (s. Bild 5).

Der Standardwert ist **Ganzes Feld markieren**, was hier auch geschehen ist und das beobachtete Verhalten verursacht (sollten Sie eine andere Einstellung vorfinden, haben Sie auch ein entsprechendes anderes Verhalten beobachten können).

Wie können wir nun den Inhalt des Feldes transferieren, ohne dass nach dem Zurücksetzen des Fokus auf das auslösende Feld wieder der komplette Inhalt markiert und ersetzt wird? Hilft es eventuell, wenn wir die Einstellung für diese Aktion per Code ändern? Wohl kaum, denn wir wissen ja nicht, an welcher Stelle der Benutzer die Änderung durchgeführt hat.

Die einzige Möglichkeit, die wir haben, ist, uns die aktuelle Cursorposition zu merken und die Länge der Markierung

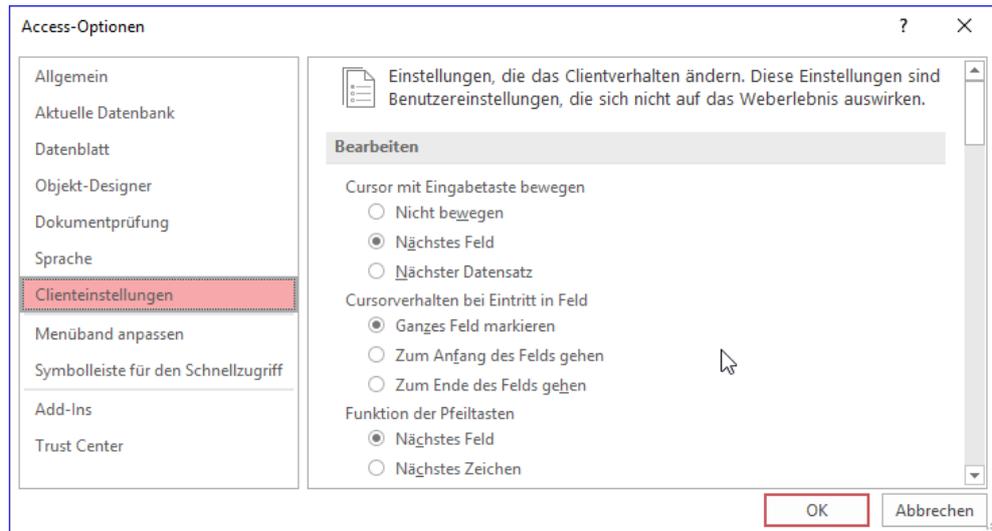


Bild 5: Option zum Einstellen des Verhaltens beim Eintritt in ein Feld

und die Position vor der Rückkehr des Fokus zum auslösenden Feld wiederherzustellen.

Dies erledigen wir, indem wir zunächst zwei Variablen zur Prozedur hinzufügen, mit denen wir die Position der Einfügemarke sowie die Breite der Markierung speichern. Dann füllen wir diese beiden Variablen mit den entsprechenden Werten für das Textfeld **txtFirma**. Anschließend folgen die bereits vorhandenen Anweisungen der Prozedur. Nach dem Verschieben des Fokus auf das Textfeld **txtFirma_Alt** und vor dem Erhalt des Fokus durch das Textfeld **txtFirma** stellen wir die Markierungsposition und -breite wieder her:

```
Private Sub txtFirma_Dirty(Cancel As Integer)
    Dim lngSelStart As Long
    Dim lngSelLength As Long
    lngSelStart = Me!txtFirma.SelStart
    lngSelLength = Me!txtFirma.SelLength
    With Me!txtFirma_Alt
        .Visible = True
        .Value = Me!txtFirma.OldValue
    End With
    Me!txtFirma.SelStart = lngSelStart
    Me!txtFirma.SelLength = lngSelLength
End Sub
```

Nun erhalten wir das gewünschte Verhalten, wie Bild 6 zeigt. Wir haben hier den Buchstaben **n** aus dem Firmennamen gelöscht, der anschließend genau wie erwartet angezeigt wird.

Übrige Felder mit Funktionalität ausstatten

Nun wollen wir noch schnell die übrigen Felder mit der hier implementierten Funktion ausstatten. Dazu legen Sie einfach die leeren Prozeduren für das Ereignis **Bei Geändert** an und kopieren den Inhalt der weiter oben bereits vorgestellten Prozedur für das Feld **txtFirma** ein.

Außerdem passen Sie dort noch die Feldnamen an, indem Sie etwa für das Textfeld **txtVorname** jeweils **txtFirma** durch **txtVorname** und **txtFirma_Alt** durch **txtVorname_Alt** ersetzen:

```
Private Sub txtVorname_Dirty(Cancel As Integer)
    Dim lngSelStart As Long
    Dim lngSelLength As Long
    lngSelStart = Me!txtVorname.SelStart
    lngSelLength = Me!txtVorname.SelLength
    With Me!txtVorname_Alt
        .Visible = True
        .Value = Me!txtVorname.OldValue
    End With
    Me!txtVorname.SelStart = lngSelStart
    Me!txtVorname.SelLength = lngSelLength
End Sub
```

Dies müssen Sie nun nur noch für alle übrigen Felder durchführen und erhalten so einen sehr gut wartbaren Code. Äh ... nein, doch nicht. Genau genommen haben wir nun eine ganze Reihe von Prozeduren erstellt, die alle das gleiche tun – mit dem Unterschied, dass jeweils zwei andere Textfelder referenziert werden.

Code optimieren

Eine erste Möglichkeit, den Code zumindest zuverlässiger vervielfältigen zu können, ist die

folgende Variante, die wir für das Ereignis **Bei Änderung** des Textfeldes **txtNachname** implementiert haben:

```
Private Sub txtNachname_Dirty(Cancel As Integer)
    Dim lngSelStart As Long
    Dim lngSelLength As Long
    Dim strTextfeld As String
    strTextfeld = "txtNachname"
    lngSelStart = Me(strTextfeld).SelStart
    lngSelLength = Me(strTextfeld).SelLength
    With Me(strTextfeld & "_Alt")
        .Visible = True
        .Value = Me(strTextfeld).OldValue
    End With
    Me(strTextfeld).SelStart = lngSelStart
    Me(strTextfeld).SelLength = lngSelLength
End Sub
```

Hier haben wir den Namen des betroffenen Textfeldes, nämlich **txtNachname**, in einer Variablen namens **strTextfeld** gespeichert. Danach haben wir alle Bezüge auf dieses Steuerelement in der Prozedur, die etwa **Me!txtNachname** lauten, in **Me(strTextfeld)** geändert. Für die Stellen, an denen es eine Referenz auf **Me!txtNachname_Alt** gibt, verwenden wir **Me(strTextfeld & "_Alt")**.

Hier profitieren wir also davon, dass wir die Textfelder zum Anzeigen der alten Werte immer konsequent mit dem Suffix **_Alt** ausgestattet haben.

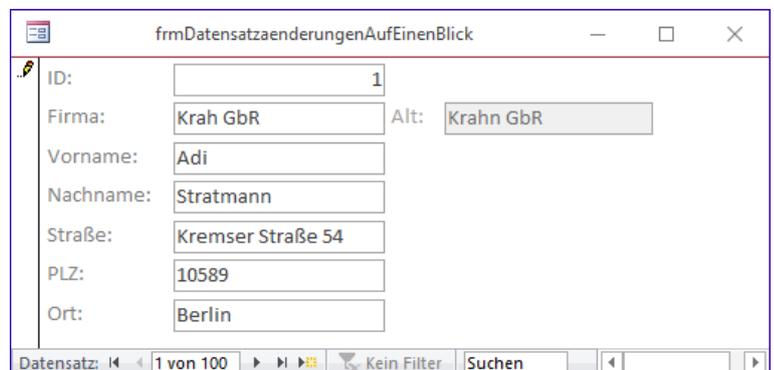


Bild 6: Gewünschtes Verhalten

Undo-Ereignis und Undo per Escape im Griff

Wenn Sie auf das Rückgängigmachen von Änderungen an den Daten eines Formulars reagieren wollen, haben Sie mehrere Möglichkeiten. Sie können die beiden Undo-Ereignisse des Formulars selbst und des aktiven Steuerelements nutzen (zumindest theoretisch). Oder Sie verwenden die Tastatur-Ereignisse, um die zum Rückgängigmachen von Änderungen bevorzugt verwendete Escape-Taste abzufangen. Wir stellen die beiden Techniken vor und erklären Vor- und Nachteile.

Warum Undo abfangen?

Als Erstes stellt sich die Frage, warum man Undo-Ereignisse überhaupt abfangen sollte. Ein gängiger Anwendungsfall ist, den Benutzer vor dem Verwerfen umfangreicher Änderungen an den Daten eines Datensatzes in einem Formular zu fragen, ob er die Änderungen wirklich verwerfen will.

Einen anderen Anwendungsfall schauen wir uns im Beitrag **Datensatzänderungen auf einen Blick** an (www.access-im-unternehmen.de/1085). Hier wollen wir zu einem Textfeld, dessen Inhalt der Benutzer geändert hat, ein weiteres Textfeld einblenden, das den ursprünglichen Inhalt des Textfeldes anzeigt.

Wenn der Benutzer beispielsweise die Escape-Taste verwendet, um die Änderungen rückgängig zu machen, sollen die Textfelder, welche die Version vor der Änderung anzeigen, natürlich auch wieder ausgeblendet werden.

Undo-Ereignisse

Access kennt zwei Undo-Ereignisse. Eines kann für das aktuelle Formular implementiert werden, das andere für beliebige Steuerelemente, welche an Felder der Datenherkunft gebunden werden können.

Wir wollen das Formular aus der Lösung zum oben genannten Beitrag als Beispielformular nutzen. Es verwendet eine Tabelle namens **tblKunden** als Datenherkunft und zeigt alle Felder dieser Tabelle im Detailbereich an (s. Bild 1).

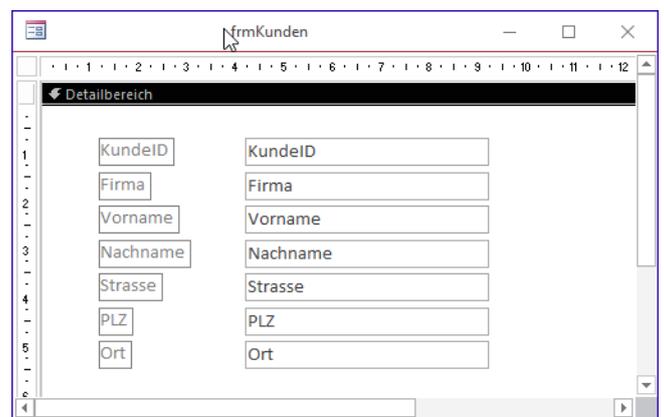


Bild 1: Formular in der Entwurfsansicht

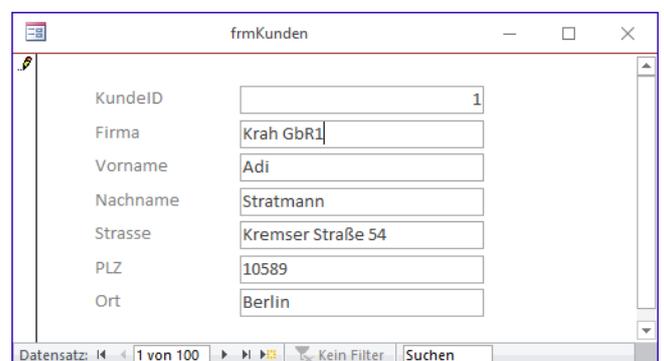


Bild 2: Datensatz in Bearbeitung

Undo per Escape

Die Escape-Taste ist das erste Mittel, wenn es darum geht, die Änderungen in einem Formular rückgängig zu machen. Dabei gibt es verschiedene Stufen.

Wenn wir einen Datensatz in der Formularansicht des Formulars anzeigen und dann den Wert eines Feldes dieses Datensatzes ändern, zeigt der Datensatzmarkierer

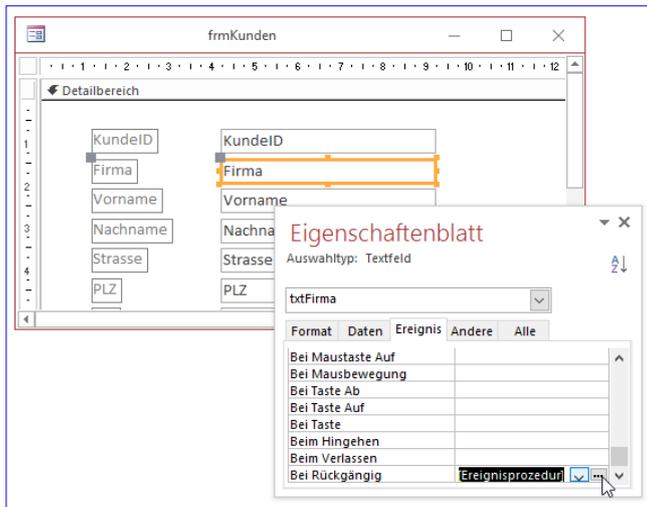


Bild 3: Anlegen des **Undo**-Ereignisses für ein Textfeld

einen Stift als Symbol für den Zustand »In Bearbeitung« an (s. Bild 2). Wenn Sie nun Änderungen an weiteren Feldern vornehmen, ändert der Datensatzmarkierer sein Aussehen nicht mehr. Intern jedoch gibt es bezüglich der Betätigung der Escape-Taste zwei Stufen der Bearbeitung:

- Wenn nur ein einziges Feld geändert wurde und der Benutzer betätigt die Escape-Schaltfläche, wird nicht nur die Änderung im aktuellen Feld rückgängig gemacht, sondern auch der komplette Datensatz wird wieder als »nicht bearbeitet« gekennzeichnet.
- Wurden hingegen in mehr als einem Feld Änderungen durchgeführt, sorgt das erste Betätigen der Escape-Taste dafür, dass die Änderung im aktuellen Feld rückgängig gemacht wird, das zweite Betätigen macht dann alle Änderungen im aktuellen Datensatz wieder rückgängig.

Dies können Sie leicht anhand des Beispielformulars **frmKunden** nachvollziehen.

Auslösen der Undo-Ereignisse

Nun wollen wir uns anschauen, wie wir durch Betätigen der Escape-Schaltfläche die beiden **Undo**-Ereignisse auslösen können. Dazu legen wir zunächst eine Ereignispro-

zedur für das Ereignis **Bei Rückgängig** eines Textfeldes an – in Bild 3 beispielsweise für das Textfeld **txtFirma**.

Die dafür hinterlegte Ereignisprozedur soll zunächst nur eine Meldung im Direktbereich des VBA-Editors ausgeben:

```
Private Sub txtFirma_Undo(Cancel As Integer)
    Debug.Print "txtFirma_Undo"
End Sub
```

Die gleiche Prozedur legen wir für die übrigen Textfelder an:

```
Private Sub txtNachname_Undo(Cancel As Integer)
    Debug.Print "txtNachname_Undo"
End Sub
...
```

Schließlich markieren Sie das Formular und legen auch für dieses für die Ereignisseigenschaft **Bei Rückgängig** eine Ereignisprozedur wie die folgende an:

```
Private Sub Form_Undo(Cancel As Integer)
    Debug.Print "Form_Undo"
End Sub
```

Damit können wir nun experimentieren:

- Wenn Sie den Wert des Textfeldes **txtFirma** ändern und dann die Escape-Schaltfläche betätigen, lösen Sie nacheinander die Ereignisse **txtFirma_Undo** und **Form_Undo** aus.
- Wenn Sie den Wert des Textfeldes **txtFirma** und dann den Wert des Textfeldes **txtVorname** ändern und dann die Escape-Taste drücken, lösen Sie das Ereignis **txtVorname_Undo** aus. Betätigen Sie die Escape-Taste dann erneut, wird das Ereignis **Form_Undo** ausgelöst.
- Wenn Sie beliebige Werte in einem oder mehreren Textfeldern ändern, dann den Fokus auf ein nicht geänder-

Filter speichern und wieder abrufen

Der Benutzer einer Access-Datenbank will sich nicht damit beschäftigen, Where-Ausdrücke zu definieren, mit denen er die Daten seiner Datenbank filtern kann. Er braucht aussagekräftige Bezeichnungen, mit denen er die gewünschten Daten erhält – am besten mit nicht viel mehr als einem Mausklick. Das ist eine schöne Aufgabe: Also statten wir unsere Anwendung mit einer solchen Möglichkeit aus, die der Benutzer sogar noch mit selbst angewendeten Filterkriterien anreichern kann. Und das, ohne auch nur einen Hauch von SQL zu beherrschen.

Ausgangsdatenbank

Die Beispieldatenbank enthält eine Tabelle **tblKunden** und eine Tabelle **tblAnreden**, wobei die Tabelle **tblKunden** über die Fremdschlüsselfelder **LieferAnrede** und **RechnungAnrede** mit der Tabelle **tblAnreden** verknüpft ist. Zusätzlich haben wir eine Abfrage namens **qryKunden** angelegt, welche die beiden Tabellen **tblKunden** und **tblAnreden** zusammenfasst. Dabei haben wir in dieser Abfrage die beiden Felder **LieferAnredeID** und **RechnungAnredeID** der Tabelle **tblKunden** weggelassen und dafür die Bezeichnung der Anreden aus der Tabelle **tblAnreden** hinzugefügt. Diese Abfrage dient als Datenherkunft eines Unterformulars namens **sfmKundenfilter**. Die Felder haben wir vollständig in den Detailbereich der Entwurfsansicht gezogen. Damit das Unterformular die Daten in der Datenblattansicht anzeigt, stellen Sie die Eigenschaft **Standardansicht** auf diesen Wert ein. Das Unterformular haben wir in ein Hauptformular namens **frmKundenfilter** gezogen, das neben diesem Unterformular natürlich noch die Steuerelemente zum Anwenden der Filter enthalten soll. Dieser Aufbau sieht wie in Bild 1 aus.

Ziel der Lösung

Das Ziel ist nun, dem Hauptformular ein Kombinationsfeld hinzuzufügen, mit dem der Benutzer einen vordefinierten Filter auswählen kann. Hat er diesen ausgewählt, soll der Filter direkt auf die Daten im Unterformular angewendet

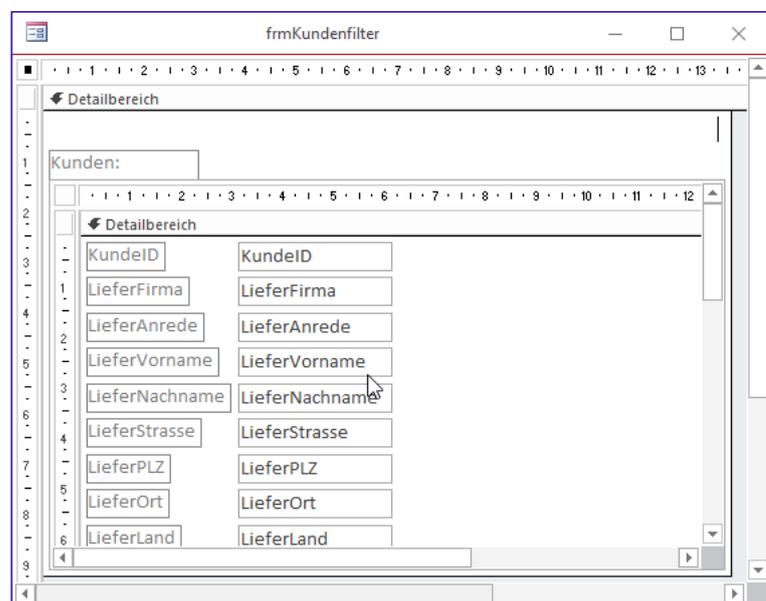


Bild 1: Basisausstattung des Formulars **frmKundenfilter** samt Unterformular **sfmKundenfilter**

werden. Das bedeutet, dass wir die Filter auch an irgendeiner Stelle vordefinieren und auch abrufbar machen müssen. Was bietet sich da besser an als eine Tabelle?

Tabelle zum Speichern der Filter

Die Tabelle, in der wir die Filterkriterien speichern wollen, heißt **tblFormularfilter** und ist wie in der Entwurfsansicht aus Bild 2 aufgebaut. Neben dem Primärschlüsselfeld enthält die Tabelle drei Felder:

- **Bezeichnung:** Bezeichnung des Filters, wie er auch im Kombinationsfeld des Formulars zur Auswahl angeboten werden soll

- **Filter:** Filterkriterium im SQL-Format
- **Formularname:** Name des Formulars, für das der Filter vorgesehen ist. Durch dieses Feld ermöglichen wir den Einsatz der Lösung in mehreren Formularen.

Da jede Bezeichnung nur einmal für jedes Formular verwendet werden soll, legen wir außerdem noch einen eindeutigen Index über die beiden Felder **Bezeichnung** und **Formularname** fest (s. Bild 3).

Bezeichnung und **Formularname** fest (s. Bild 3).

Um gleich mit dieser Tabelle arbeiten zu können, geben wir ein paar Beispiele für Filterkriterien in die Datenblattansicht der Tabelle ein (s. Bild 4).

Einbau des Filterkombinationsfeldes

Nun wollen wir das Kombinationsfeld zum Filtern der Daten des Unterformulars nach einem der vordefinierten Filter hinzufügen. Dazu legen Sie ein einfaches Kombinationsfeld im oberen Bereich des Formulars an und nennen es **cboFormularfilter**. Tragen Sie für die Eigen-

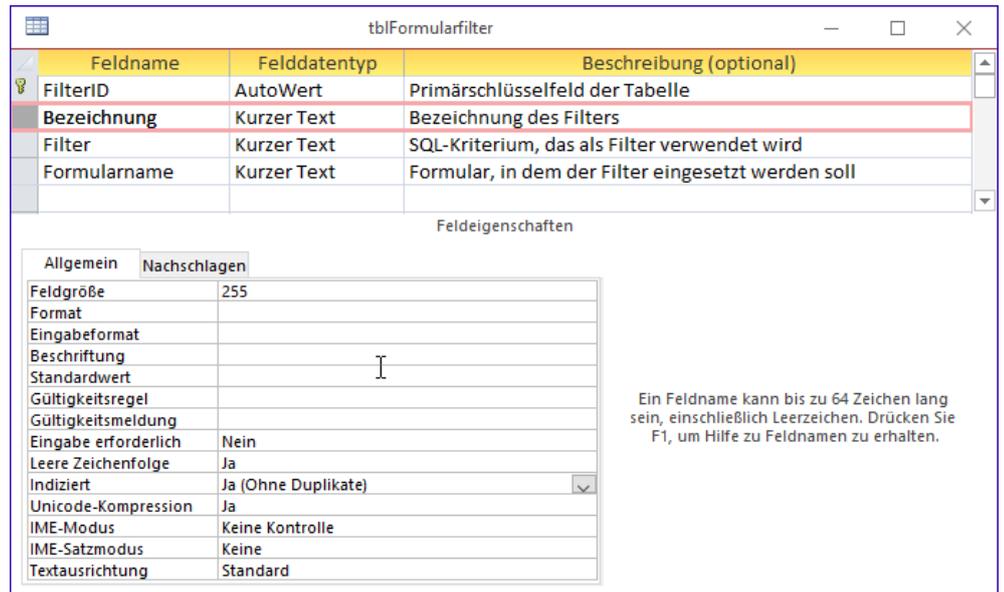


Bild 2: Tabelle zum Speichern der Filterkriterien

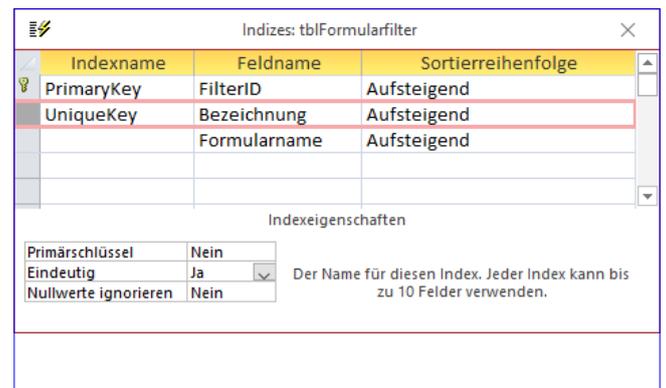


Bild 3: Eindeutiger Index über die beiden Felder **Bezeichnung** und **Formularname**

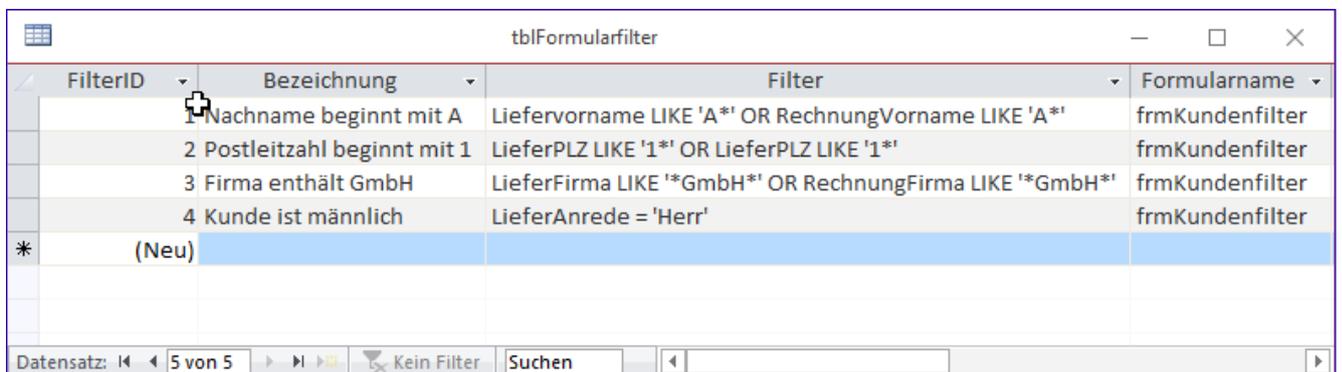


Bild 4: Die Tabelle **tblFormularfilter** mit einigen Beispieldatensätzen

schaft **Datensatzherkunft** des Kombinationsfeldes die Abfrage **tblFormularfilter** ein. Stellen Sie für die Eigenschaft **Spaltenanzahl** den Wert **3** und für die Eigenschaft **Spaltenbreiten** den Wert **0cm;;0cm** ein (s. Bild 5). Was bedeuten diese Werte? Als Datensatzherkunft verwenden wir die Tabelle **tblFormularfilter**.

Wir benötigen das erste Feld als gebundene Spalte, das zweite als anzuzeigenden Wert und das dritte enthält das Filterkriterium, das wir auslesen wollen, wenn der Benutzer einen der Einträge ausgewählt hat. Also teilen wir dem Kombinationsfeld drei Spalten zu, von denen die erste und die dritte die Breite 0cm aufweisen sollen – also quasi ausgeblendet sind. Nur die mittlere Spalte soll erscheinen, und das in voller Breite. Daher finden Sie zwischen den beiden Semikola keine explizite Breitenangabe.

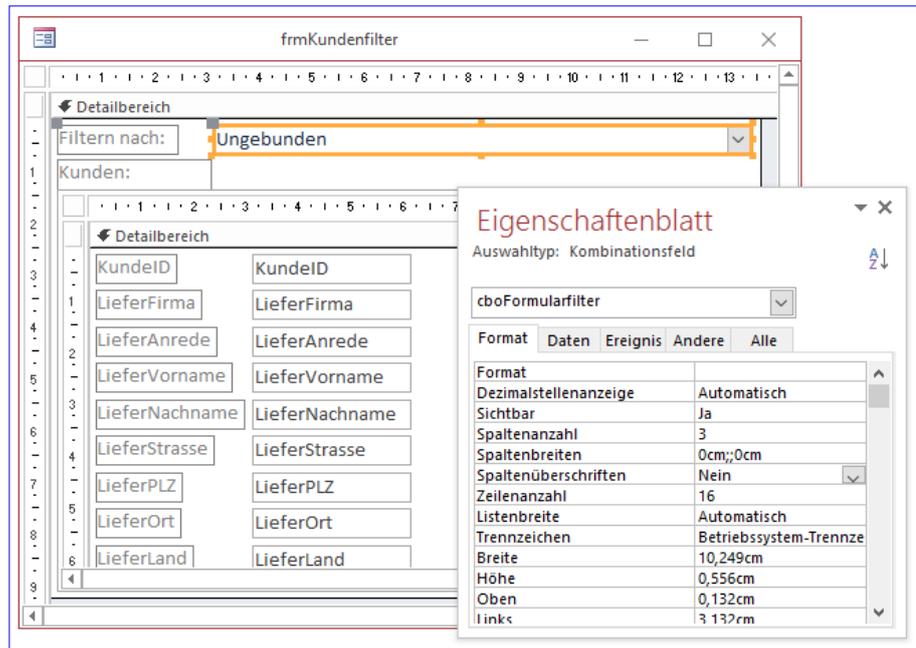


Bild 5: Hinzufügen des Kombinationsfeldes zur Auswahl des Filters

Nach Formular filtern

Nun haben wir allerdings in der Tabelle **tblFormularfilter** bereits ein Feld namens **Formularname** hinterlegt, mit dem wir festlegen wollen, für welches Formular der jeweilige Filter verwendet werden kann. Wir können ja durchaus ein Kundenformular, ein Artikelformular oder auch ein Lieferantenformular mit einem solchen Filter ausstatten.

Wie aber filtern wir die Datensatzherkunft eines Kombinationsfeldes nach dem Namen des Formulars, in dem es angezeigt wird? Dies erledigen wir am einfachsten, indem wir die Datensatzherkunft beim Laden des Formulars einstellen. Die dazu notwendige Ereignisprozedur wird durch das Ereignis **Beim Laden** ausgelöst und sieht wie folgt aus:

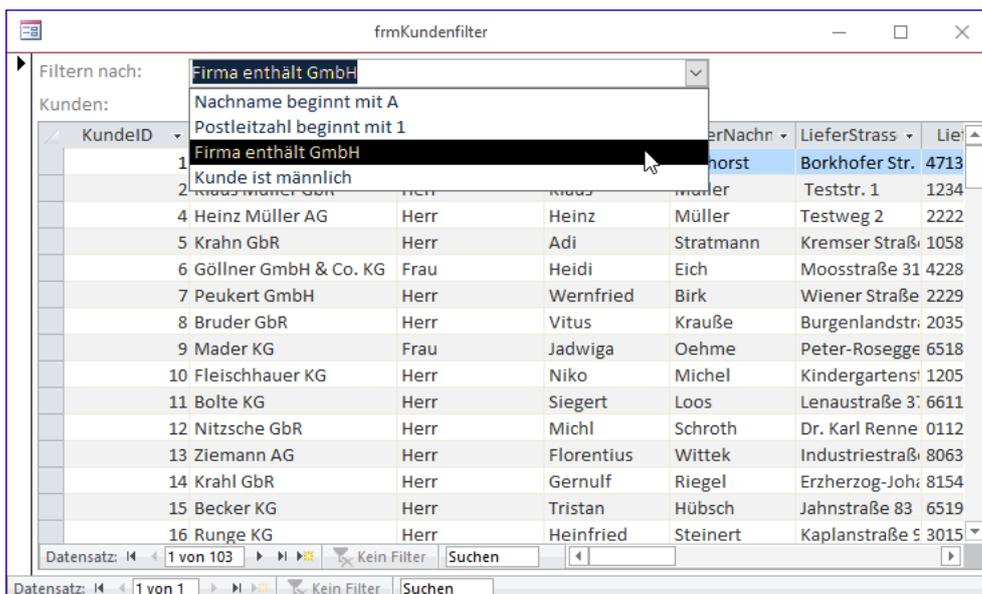


Bild 6: Kombinationsfeldes zur Auswahl des Filters

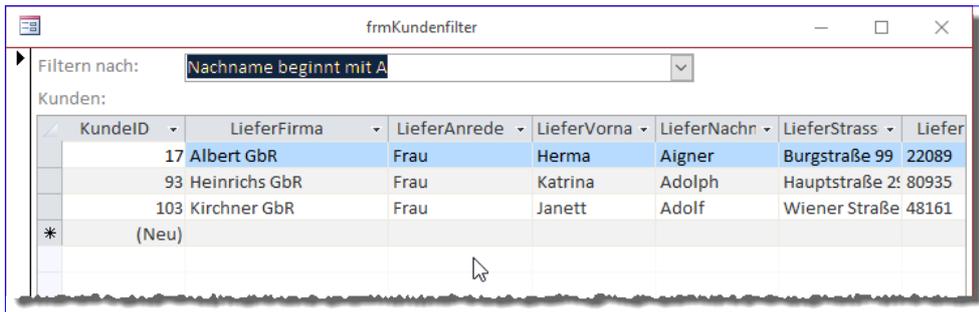


Bild 7: Der Filter in Aktion

```
Private Sub Form_Load()
    Me!cboFormularfilter.RowSource = _
        "SELECT * FROM tblFormularfilter 7
        WHERE Formularname = '" & Me.Name & "'"
End Sub
```

Danach finden wir im Kombinationsfeld bereits alle gewünschten Einträge vor (s. Bild 6).

Filterfunktion hinzufügen

Nun fehlt als Grundausstattung noch eine Funktion, die den in der Tabelle gespeicherten Filterausdruck nach der Auswahl eines der Einträge des Kombinationsfeldes auf die Daten des Unterformulars anwendet. Diese Prozedur erstellen wir für das Ereignis **Nach Aktualisierung** des Kombinationsfeldes. Sie sieht wie folgt aus:

```
Private Sub cboFormularfilter_AfterUpdate()
    Dim strFilter As String
    strFilter = Me!cboFormularfilter.Column(2)
    Me!sfmKundenfilter.Form.Filter = strFilter
    Me!sfmKundenfilter.Form.FilterOn = True
End Sub
```

Die Prozedur ermittelt mit der Eigenschaft **Column(2)** den Wert der dritten Spalte des ausgewählten Eintrags des Kombinationsfeldes und trägt diesen in die Variable **strFilter** ein (zur Erläuterung: der Index der Spalten im Kombinationsfeld startet mit **0**). Dann weist sie diesen Ausdruck der Eigenschaft **Filter** des

Form-Objekts des Unterformular-Steuerlements zu und stellt **FilterOn** auf den Wert **True** ein. Das Ergebnis sieht dann beispielsweise wie in Bild 7 aus.

Filter leeren

Nun fehlt noch eine einfache Funktion, mit welcher

der Benutzer den Filter leeren und wieder alle Datensätze im Unterformular einblenden kann. Dazu haben wir zwei Möglichkeiten: Entweder wir fügen eine Schaltfläche hinzu, welche den Filter wieder entfernt, oder wir legen im Kombinationsfeld einen Eintrag an, der etwa den Text **<Filter leeren>** anzeigt. Hier entscheiden wir uns für den ersten Weg, was zwar bedeutet, dass wir ein zusätzliches Steuerelement anlegen müssen (s. Bild 8). Allerdings ist dies für den Benutzer offensichtlicher als ein Eintrag im Kombinationsfeld, der nicht direkt sichtbar ist.

Die Schaltfläche fügen Sie am besten direkt neben dem Kombinationsfeld ein und legen für das Ereignis **Beim Klicken** die folgende Ereignisprozedur an:

```
Private Sub cmdFilterLeeren_Click()
    Me!cboFormularfilter = Null
    Me!sfmKundenfilter.Form.Filter = ""
End Sub
```

Eigene Filter definieren

Nun kommt der interessante Teil: Sie wollen ja nicht immer, wenn der Benutzer ein neues Filterkriterium wünscht,

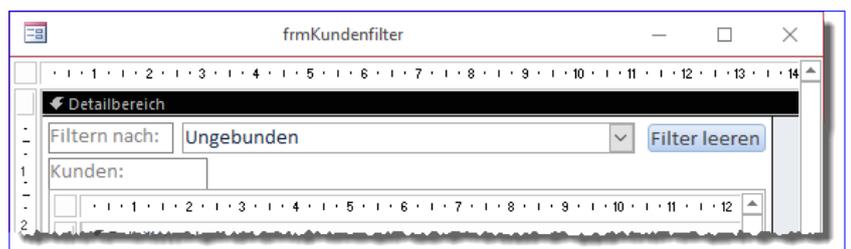


Bild 8: Schaltfläche zum Deaktivieren des Filters

felder leert und auch den Filter für das Unterformular zurücksetzt.

Das Unterformular heißt **sfmFormularfilter** und verwendet die Tabelle **tblFormularfilter** als Datenherkunft. Es zeigt alle Felder der Tabelle in der Datenblattansicht an.

Die beiden Textfelder zum Filtern der Datensätze des Unterformulars erwarten die Eingabe des Benutzers und filtern das Unterformular nach jedem getippten Zeichen neu, damit der Benutzer immer direkt das aktuelle Ergebnis sieht.

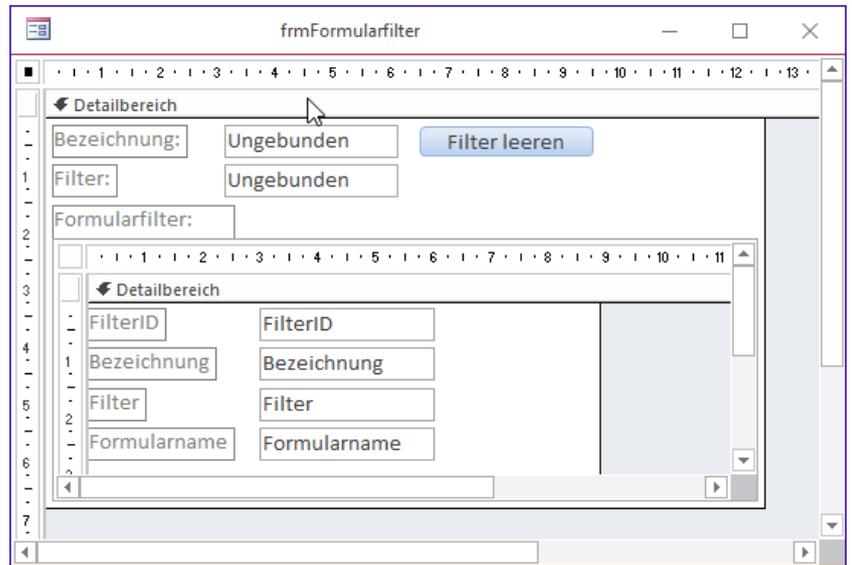


Bild 13: Formular zum Verwalten der Filter in der Entwurfsansicht

Zu diesem Zweck finden Sie im Kopf des Klassenmoduls zu diesem Formular die Deklaration zweier Variablen zum Aufnehmen des jeweils aktuellen Textes der beiden Textfelder:

```
Dim strSucheBezeichnung As String
Dim strSucheFilter As String
```

Wenn der Benutzer nun etwa ein Zeichen in das Textfeld **txtSucheBezeichnung** eintippt, löst er das Ereignis **Bei Änderung** aus. Dafür haben wir die Ereignisprozedur **txtSucheBezeichnung_Change** hinterlegt, die wie folgt aussieht:

```
Private Sub txtSucheBezeichnung_Change()
    strSucheBezeichnung = Me!txtSucheBezeichnung.Text
    Suchen
End Sub
```

Hier ermitteln wir über die Eigenschaft **Text** des Textfeldes **txtSucheBezeichnung** den aktuellen Inhalt dieses Textfeldes und tragen ihn in die Variable **strSucheBezeichnung** ein. Warum **Text** und nicht einfach **Value**? Weil **Value** sich erst nach dem Abschließen der Eingabe etwa durch Verlassen des Textfeldes ändert und wir so nicht nach der

Eingabe eines jeden Zeichens neu filtern können. Nach dem Speichern des Vergleichswertes ruft die Prozedur die Routine **Suchen** auf, die wir gleich im Anschluss erläutern.

Zuvor noch ein Blick auf die entsprechende Ereignisprozedur für das Textfeld **txtSucheFilter**. Diese ist analog aufgebaut wie **txtBezeichnungFilter_Change** und speichert den aktuellen Text des Textfeldes in der Variablen **strSucheFilter**:

```
Private Sub txtSucheFilter_Change()
    strSucheFilter = Me!txtSucheFilter.Text
    Suchen
End Sub
```

Warum aber speichern wir die Inhalte der Text-Eigenschaft jeweils in lokalen Variablen, bevor wir die **Suchen**-Funktion aufrufen? Wir könnten doch auch von der Suchen-Funktion auf die entsprechenden Werte der Textfelder zugreifen? Leider nein: Auf die **Text**-Eigenschaft kann man nur zugreifen, wenn das jeweilige Textfeld gerade den Fokus besitzt. Wir müssten also in der Routine zum Suchen erst noch den Fokus verschieben, was zu unnötigen Komplikationen führt. Da speichern wir die aktuellen Werte doch lieber schnell in zwei Variablen.

Beide Prozeduren rufen danach die Routine **Suchen** auf, die wir in Listing 2 abgebildet haben. Diese prüft zunächst, ob **strSucheBezeichnung** eine Zeichenkette mit mehr als **0** Zeichen enthält. In diesem Fall fügt sie zur Variablen **strSuchausdruck** einen Teil hinzu, der beispielsweise so aussieht:

```
AND Bezeichnung LIKE '*Liefer*'
```

Das Gleiche geschieht danach für die **String**-Variable **strSucheFilter**. Wenn beide Textfelder bereits einen Wert enthalten, sind auch die beiden **String**-Variablen gefüllt, wodurch dann zwei Ausdrücke in der Variablen **strSuchausdruck** zusammengeführt werden.

Hat **strSuchausdruck** danach also einen Inhalt mit der Länge größer **0**, wird das führende **AND** vorn im Suchausdruck mit der **Mid**-Funktion abgeschnitten. Dann landet der Suchausdruck in der Eigenschaft **Filter** des **Form**-Objekts des Unterformular-Steuerlements und der Filter wird mit **FilterOn = True** aktiviert. Sollte keine der beiden Variablen **strSucheBezeichnung** und **strSucheFilter** gefüllt sein, wird der Filter mit **Filter = ""** deaktiviert.

Das Ergebnis sieht dann etwa wie in Bild 14 aus.

Nun haben wir auch diesem Formular eine Schaltfläche namens **cmdFilterLeeren** hinzugefügt, welche beim Anklicken die folgende Prozedur auslösen soll:

```
Private Sub cmdFilterLeeren_Click()  
    Me!txtSucheBezeichnung = Null  
    Me!txtSucheFilter = Null  
    strSucheBezeichnung = ""  
    strSucheFilter = ""  
    Suchen  
End Sub
```

Damit werden zunächst die beiden Textfelder **txtSucheBezeichnung** und **txtSucheFilter** geleert und danach die entsprechenden **String**-Variablen. Der anschließende Aufruf der **Suchen**-Routine sorgt dann dafür, dass das Unterformular wieder alle Datensätze anzeigt.

Filter für alle oder nur für Formulare?

Wir müssen nun noch entscheiden, ob das Formular zur Verwaltung der Formularfilter immer alle Datensätze für die aktuelle Datenbank anzeigen soll oder ob der Benutzer

```
Private Sub Suchen()  
    Dim strSuchausdruck As String  
    If Len(strSucheBezeichnung) > 0 Then  
        strSuchausdruck = strSuchausdruck & " AND Bezeichnung LIKE '*'" & strSucheBezeichnung & "*'"  
    End If  
    If Len(strSucheFilter) > 0 Then  
        strSuchausdruck = strSuchausdruck & " AND Filter LIKE '*'" & strSucheFilter & "*'"  
    End If  
    If Len(strSuchausdruck) > 0 Then  
        strSuchausdruck = Mid(strSuchausdruck, 6)  
        Me!sfmFormularfilter.Form.Filter = strSuchausdruck  
        Me!sfmFormularfilter.Form.FilterOn = True  
    Else  
        Me!sfmFormularfilter.Form.Filter = ""  
    End If  
End Sub
```

Listing 2: Durchsuchen der Filter nach verschiedenen Kriterien

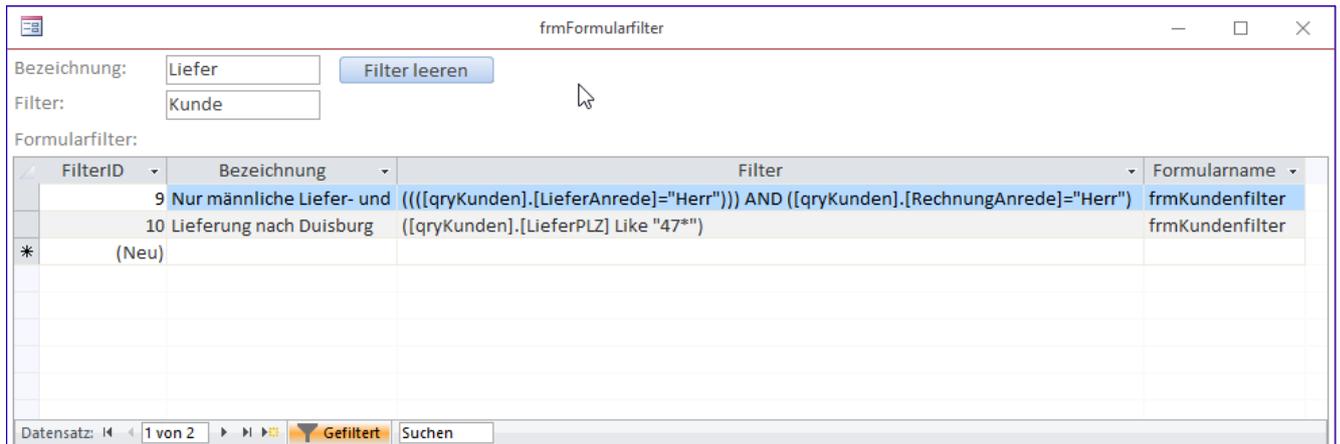


Bild 14: Formular zum Verwalten in der Formularansicht

nur die Datensätze sehen soll, die sich auf das Formular beziehen, von dem aus er das Formlar zur Filterverwaltung aufgerufen hat. Letzteres scheint logischer zu sein, denn wieso sollte man Filter verwalten, die sich auf mehrere Formulare beziehen?

Also stellen wir das Formular so ein, dass das Unterformular gleich zu Beginn nur die Filter zum aktuellen Formular anzeigt. Dazu leeren wir einfach die Eigenschaft **Datenherkunft** des Formulars und weisen beim Laden des Formulars eine entsprechend gefilterte Datenherkunft zu. Warum auf diesem Wege und nicht durch Anpassen des Filterausdrucks? Weil wir dann mit den anderen per Textfeld zu setzenden Filtern durcheinander kommen.

Also verwenden wir beim Laden des Formulars die Ereignisprozedur aus Listing 3. Diese ermittelt mit **Screen.ActiveForm.Name** den Namen des aufrufenden Formulars und stellt eine **Recordsource** zusammen, welche die Tabelle **tblFormularfilter** nach diesem Formular filtert. Dann nutzen wir noch die beim Aufruf des Formulars per Öffnungsargument übergebene ID des aktuell im Kombinationsfeld zur Auswahl des Filters ausgewählten Filter. Sofern ein solcher markiert ist, wird der Datensatzzeiger des Unterformulars gleich auf diesen Eintrag eingestellt.

Der Benutzer kann seine Filterausdrücke nun komfortabel mit diesem Formular bearbeiten, neue Filter hinzufügen und nicht mehr benötigte Filter löschen.

```
Private Sub Form_Load()
    Dim lngFilterID As String
    Dim strRecordsource As String
    Dim strFormularname As String
    lngFilterID = Me.OpenArgs
    strFormularname = Screen.ActiveForm.Name
    strRecordsource = "SELECT * FROM tblFormularfilter WHERE Formularname = '" & strFormularname & "'"
    Me!sfmFormularfilter.Form.RecordSource = strRecordsource
    If Not lngFilterID = 0 Then
        Me!sfmFormularfilter.Form.Recordset.FindFirst "FilterID = " & lngFilterID
    End If
End Sub
```

Listing 3: Beim Laden der Filterverwaltung

Mails mit Vorlage

Ein großer Teil der Kommunikation mit Kunden läuft per E-Mail. Das ist vor allem dann der Fall, wenn der Kunde ein Anliegen per E-Mail vorbringt. Manchmal muss man den Kunden aber auch aus anderen Gründen kontaktieren – beispielsweise weil eine Lieferung per Post zurückgekommen ist. Wollen Sie dann die Kundenverwaltung öffnen, die E-Mail-Adresse des Kunden ermitteln, nach Outlook wechseln, dort eine neue E-Mail an den Kunden anlegen und Ihre Nachricht samt Betreff von Hand eingeben? Nein, denn die meisten Anfragen lassen sich leicht mit einer geeigneten Vorlage erledigen.

Als Ausgangspunkt für die Lösung aus diesem Beitrag nehmen wir eine zurückgesendete Lieferung an einen Kunden (das war der konkrete Anlass für diesen Beitrag). In diesem Fall wollen wir dem Kunden eine Mitteilung schicken, dass eine Sendung zurückgeschickt wurde und ihn bitten, die beim Versand verwendete Adresse zu prüfen und gegebenenfalls zu korrigieren.

Bisher habe ich dazu die Kundenverwaltung gestartet, dort das Kundenformular geöffnet und über das Suchfeld den Datensatz zu diesem Kunden ermittelt. Dann habe ich eine neue E-Mail in Outlook angelegt, die E-Mail-Adresse in das An-Feld eingetragen, eine Betreff zur E-Mail hinzugefügt und einen entsprechenden Text geschrieben.

Dieser enthielt dann die beim Versand genutzte Adresse, die ich praktischerweise per Mausklick generieren kann

und nur noch als Adressblock in die E-Mail zu kopieren brauche.

Alles in allem eine Menge Schritte, die ich etwa bei einem Versand eines Magazins zwischen fünf und zehn Mal wiederholen muss. Für jemanden, der gern automatisierte Wege für Standardaufgaben entwickelt und nutzt, ist das eine wenig erfreuliche Aufgabe. Allerdings hat bisher auch die Motivation nicht gereicht, um hierfür eine Lösung zu programmieren – immerhin würde das doch noch etwas länger dauern als die manuelle Erstellung von fünf bis zehn individuellen Mails. Wenn es aber der Zufall so will, dass ich gerade Beiträge für Access im Unternehmen schreibe, lässt sich das prima kombinieren – ich habe eine praktische Lösung und kann diese auch noch an meine Leser weitergeben. Also: Ran ans Werk!

KundelID	Firma	Vorname	Nachname	AnredeID	Strasse	PLZ	Ort	EMail
1	Krahn GbR	Adi	Stratmann	Herr	Kremser Straße 54	10589	Berlin	adi@stratmann.de
2	Göllner AG	Heidi	Eich	Frau	Moosstraße 30	42289	Wuppertal	heidi@eich.de
3	Peukert GmbH	Wernfried	Birk	Herr	Wiener Straße 78	22297	Hamburg	wernfried@birk.de
4	Bruder GmbH	Vitus	Krauß	Herr	Burgenlandstraße 77	20355	Hamburg	vitus@krauß.de
5	Mader KG	Jadwiga	Oehme	Frau	Peter-Rosegger-Straße 72	65187	Wiesbaden	jadwiga@oehme.de
6	Fleischhauer G	Niko	Michel	Herr	Kindergartenstraße 42	12051	Berlin	niko@michel.de
7	Bolte KG	Siegert	Loos	Herr	Lenastraße 80	66115	Saarbrücken	siegert@loos.de
8	Nitzsche GmbH	Michl	Schroth	Herr	Dr. Karl Renner-Straße 97	01129	Dresden	michl@schroth.de
9	Ziemann KG	Florentius	Wittek	Herr	Industriestraße 7	80638	München	florentius@wittek.de
10	Krahl KG	Gernulf	Riegel	Herr	Erzherzog-Johann-Straße 37	81545	München	gernulf@riegel.de
11	Becker AG	Tristan	Hübsch	Herr	Jahnstraße 78	65193	Wiesbaden	tristan@hübsch.de
12	Runge GmbH	Heinfried	Steinert	Herr	Kaplanstraße 60	30159	Hannover	heinfried@steinert.de
13	Albert AG	Herma	Aigner	Frau	Burgstraße 31	22089	Hamburg	herma@aigner.de

Bild 1: Datenherkunft für die Lösung

Daten der Anwendung

Als Daten für die Anwendung nutzen wir eine Tabelle namens **tblKunden**, die in der Datenblattansicht wie in Bild 1 aussieht und deren Anreden über ein Fremdschlüsselfeld namens **AnredeID** aus der per 1:n-Beziehung verknüpften Tabelle **tblAnreden** festgelegt werden.

Ausgangspunkt

Wir gehen an dieser Stelle davon aus, dass Sie ein Kundenformular verwenden, in dem Sie die Kundendaten eines bestimmten Kunden zügig über eine entsprechende Suchfunktion anzeigen können. Wir nutzen dazu die Suchfunktion aus dem Beitrag **Flexible Suche in Formularen** (<http://www.access-im-unternehmen.de/965>) beziehungsweise **Suchen mit der clsFlexSearch-Klasse** (<http://www.access-im-unternehmen.de/964>).

Kundenformular erstellen

Falls Sie kein solches Formular verwenden, erstellen wir auf die Schnelle eines. Dazu legen Sie ein neues Formular namens **frmKunde** an.

Diesem weisen Sie als Datenherkunft die Tabelle **tblKunden** zu. Ziehen Sie alle Felder aus der Feldliste in den Detailbereich der Entwurfsansicht (s. Bild 2).

Außerdem fügen Sie die Klasse **clsFlexSearch** aus der Beispieldatenbank zu obigem Beitrag (oder aus der Datenbank zu diesem Beitrag) in das VBA-Projekt Ihrer Lösung.

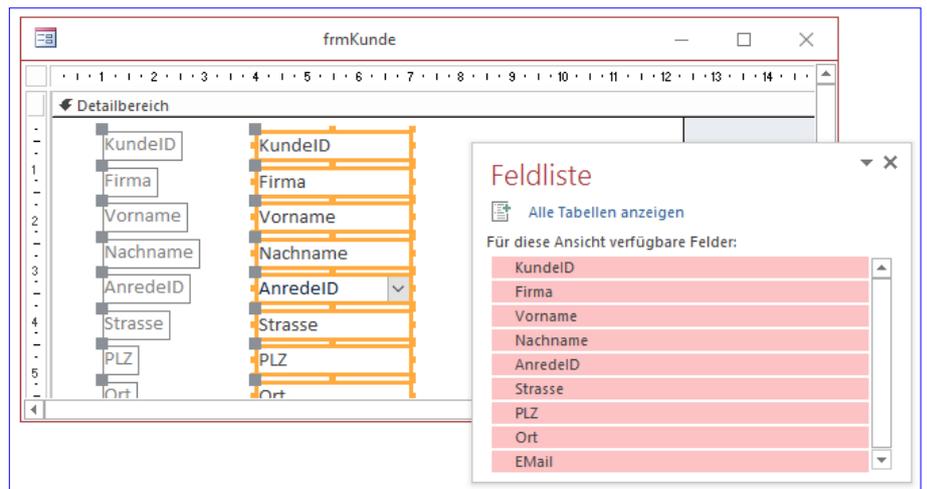


Bild 2: Entwurf des Kundenformulars

Daneben benötigen wir noch eine Abfrage namens **qryKundensuche**, welche die Felder der beiden Tabellen **tblKunden** und **tblAnreden** so zusammenführt, dass alle Felder der Tabelle **tblKunden** (außer **AnredeID**) und nur das Feld **Anrede** der Tabelle **tblAnreden** zurückgegeben werden (s. Bild 3).

Suchfeld hinzufügen

Nun fügen Sie dem Kundenformular ganz oben ein Textfeld namens **txtSuche** hinzu. Schließlich legen Sie für die Ereignisprozedur **Beim Laden** des Formulars den

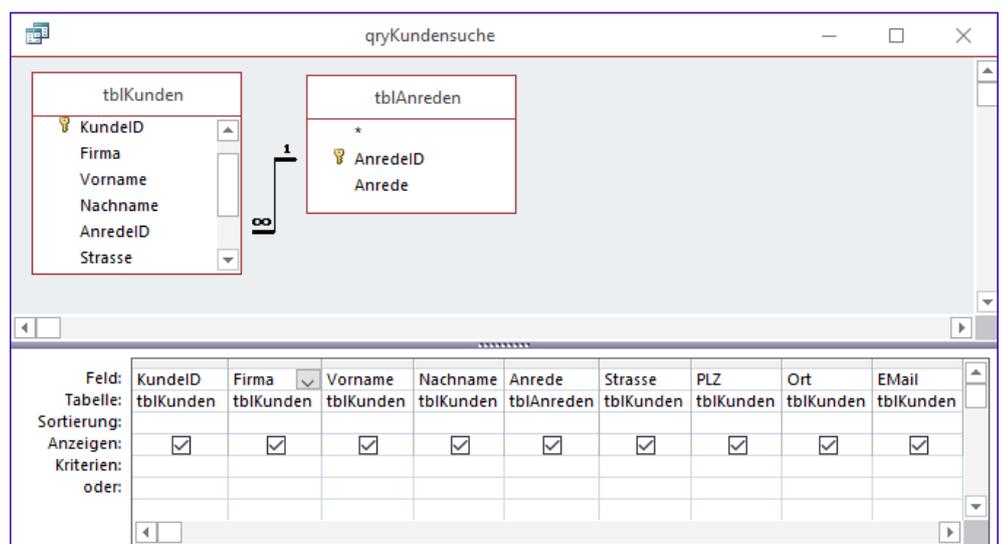


Bild 3: Abfrage als Datenherkunft für die Suchfunktion

Wert **[Ereignisprozedur]** an und klicken auf die Schaltfläche rechts daneben, um die entsprechende Prozedur zu erstellen. Diese füllen Sie dann wie folgt:

```
Private Sub Form_Load()
    Set objFlexSearch = New clsFlexSearch
    With objFlexSearch
        .Suchabfrage = "qryKundensuche"
        .PKFeld = "KundeID"
        .PKFeldEinschliessen = False
        Set .Suchfeld = Me.txtSuche
    End With
End Sub
```

Dies erstellt ein Objekt auf Basis der Klasse **clsFlexSearch** und teilt dieser mit, dass es **qryKundensuche** als Grundlage für die Suche nutzen soll, dass **KundeID** das Primärschlüsselfeld ist und dass dieses nicht in die Suche eingeschlossen werden soll.

Außerdem wird das Textfeld **txtSuche** als Suchfeld festgelegt. **objFlexSearch** müssen Sie natürlich im gleichen Formular deklarieren, und zwar wie folgt im Kopf des Moduls:

```
Dim WithEvents objFlexSearch As clsFlexSearch
```

Danach legen Sie noch eine Ereignisprozedur für das Ereignis **Beim Entladen** an, die nur die Objektvariable **objFlexSearch** leert:

```
Private Sub Form_Unload(Cancel As Integer)
    Set objFlexSearch = Nothing
End Sub
```

Schließlich definieren Sie, was geschehen soll, wenn der Benutzer einen Suchausdruck in das Textfeld **txtSuche** eingegeben hat und dann die Eingabetaste betätigt. Dazu wählen Sie aus dem linken Kombinationsfeld oben im VBA-Fenster des Klassenmoduls **Form_frmKunde** den

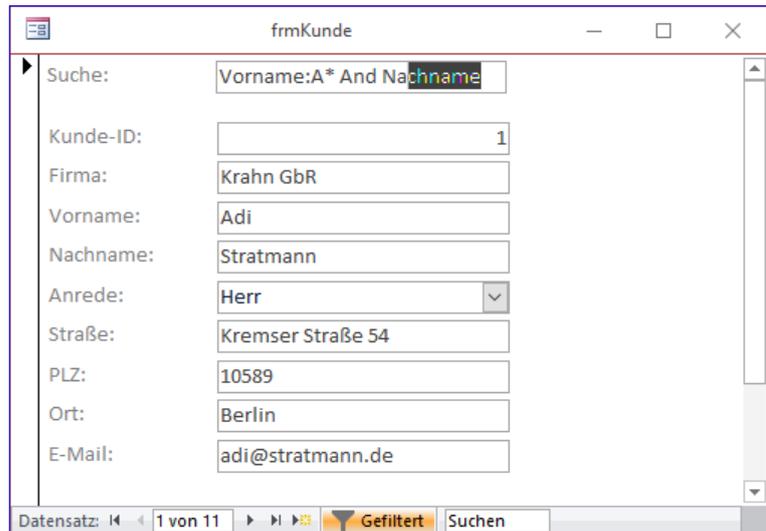


Bild 4: Suchfunktion im Einsatz

Eintrag **objFlexSearch** aus, was automatisch die leere Ereignisprozedur **objFlexSearch_Suchen** anlegt. Diese füllen Sie dann wie folgt:

```
Private Sub objFlexSearch_Suchen(strWhere As String)
    Me.Filter = strWhere
    Me.FilterOn = True
End Sub
```

Dadurch wird der Filter des Formulars auf einen Ausdruck eingestellt, der mit dem Wert des Parameters **strWhere** der Prozedur geliefert wird.

Wenn Sie das Kundenformular nun öffnen, können Sie die Feldnamen eingeben, die direkt entsprechend ergänzt werden. Ein Klick auf **Tab** vervollständigt das Suchfeld und trägt einen Doppelpunkt ein, hinter dem Sie den Vergleichswert eintragen. Mehrere Ausdrücke können Sie mit **And** voneinander trennen. Sie können aber auch einfach einen Wert wie **andre@minhorst.com** eingeben, dann werden alle Felder aller Datensätze nach diesem Wert untersucht (s. Bild 4).

Vorlagen verwalten

Nun benötigen wir eine Tabelle, mit der wir die Vorlagen für die E-Mails verwalten können. Diese Tabelle soll

Feldname	Feldtyp	Beschreibung (optional)
MailvorlageID	AutoWert	Primärschlüsselfeld der Tabelle
Bezeichnung	Kurzer Text	Bezeichnung der Mailvorlage
Betreff	Kurzer Text	Betreff inkl. Platzhalter
Inhalt	Langer Text	Inhalt inkl. Platzhalter
E-Mail	Kurzer Text	E-Mail-Adresse inkl. Platzhalter
TabelleAbfrage	Kurzer Text	Datenherkunft zum Füllen der Platzhalter
PKFeld	Kurzer Text	Primärschlüsselfeld der Datenherkunft

Feldeigenschaften	
Allgemein	Nachschlagen
Feldgröße	255
Format	
Eingabeformat	
Beschriftung	
Standardwert	
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Leere Zeichenfolge	Ja
Indiziert	Ja (Ohne Duplikate)
Unicode-Kompression	Ja
IME-Modus	Keine Kontrolle
IME-Satzmodus	Keine
Textausrichtung	Standard

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Bild 5: Die Tabelle **tblMailvorlagen**

schlicht **tblMailvorlagen** heißen und sieht im Entwurf wie in Bild 5 aus. Das Feld **Bezeichnung** haben wir mit einem eindeutigen Index versehen. Die Felder **Betreff**, **Inhalt** und **E-Mail** nehmen die Vorlagen für die entsprechenden Texte auf. Hier können Sie Gebrauch von Platzhaltern machen, die später aus den Daten der Kundentabelle gefüllt werden – mehr dazu weiter unten in der Beschreibung der Nutzung des Formulars, das diese Daten anzeigt.

Wichtig sind noch die beiden Felder **TabelleAbfrage** und **PKFeld**. **TabelleAbfrage** nimmt den Namen der Tabelle oder Abfrage auf, welche die Daten für den Kunden liefern soll, **PKFeld** den Namen des in dieser Tabelle oder Abfrage zur Identifizierung des Kunden notwendigen Primärschlüsselfeldes.

Formular zum Bearbeiten der Mailvorlagen

Die Daten dieser Tabellen wollen wir in einem separaten Formular bearbeiten. Dieses heißt **frmMailvorlagen** und sieht in der Entwurfsansicht wie in Bild 6 aus. Damit es die Daten der Tabelle **tblMailvorlagen** in alphabetischer Reihenfolge liefert, verwenden wir den folgenden SQL-Ausdruck als Datenherkunft für das Formular:

```
SELECT tblMailvorlagen.MailvorlageID,
tblMailvorlagen.Bezeichnung, tblMailvorlagen.Betreff, tblMailvorlagen.Inhalt,
tblMailvorlagen.E-Mail, tblMailvorlagen.TabelleAbfrage, tblMailvorlagen.PKFeld
FROM tblMailvorlagen
ORDER BY tblMailvorlagen.Bezeichnung;
```

Allerdings soll das Formular ja ohnehin nur die Details eines Datensatzes anzeigen, daher stellen wir die Eigenschaften **Bildlaufleisten**, **Datensatzmarkierer** und **Navigationsschaltflächen** auf den Wert **Nein** und **Automatisch zentrieren** auf **Ja** ein. Wieso dann die Datensätze nach dem Alphabet sortieren? Ganz einfach: Weil auch das Kombinationsfeld **cboAuswahl** seine Daten aus der Tabelle **tblMailvorlagen** bezieht und beide den

gleichen Datensatz anzeigen sollen. Hier bietet es sich einfach an, die Mailvorlage zu liefern, deren Bezeichnung in der alphabetischen Sortierung als erste erscheint.

The screenshot shows the design view of the form 'frmMailvorlagen'. It features a 'Detailbereich' (Detail Area) with several controls:

- Auswählen:** A dropdown menu currently set to 'Ungebunden'.
- Bezeichnung:** A text box containing the value 'Bezeichnung'.
- Tabelle/Abfrage:** A text box containing the value 'TabelleAbfrage'.
- PKFeld:** A text box containing the value 'PKFeld'.
- E-Mail:** A text box containing the value 'E-Mail'.
- Betreff:** A text box containing the value 'Betreff'.
- Inhalt:** A large text area containing the value 'Inhalt'.

At the bottom of the form, there are 'OK' and 'Neu' buttons.

Bild 6: Das Formular **frmMailvorlagen** in der Entwurfsansicht

Das Kombinationsfeld benötigt nur die beiden Felder **MailvorlageID** und **Bezeichnung**, daher verwenden wir eine verkürzte Version der Abfrage, die wir als Datenherkunft für das Formular genutzt haben, als Datensatzherkunft:

```
SELECT tblMailvorlagen.MailvorlageID, tblMailvorlagen.  
Bezeichnung FROM tblMailvorlagen ORDER BY tblMailvorlagen.  
Bezeichnung;
```

Das Formular zeigt ja nun bereits automatisch den ersten Datensatz nach alphabetischer Reihenfolge an. Damit das Kombinationsfeld den gleichen Datensatz anzeigt, fügen wir dem Formular für das Ereignis **Beim Laden** die folgende Ereignisprozedur hinzu:

```
Private Sub Form_Load()  
    Me!cboAuswahl = Me!MailvorlageID  
End Sub
```

Diese wählt den gleichen Eintrag aus, der auch im Formular angezeigt wird.

Damit nach der Auswahl eines der Einträge des Kombinationsfeldes die gewählte Mailvorlage im Formular angezeigt wird, legen Sie für das Ereignis **Nach Aktualisierung** des Kombinationsfeldes die folgende Ereignisprozedur an:

```
Private Sub cboAuswahl_AfterUpdate()  
    Me.Filter = "MailvorlageID = "  
& Me!cboAuswahl  
    Me.FilterOn = True  
End Sub
```

Dies filtert das Formular nach dem im Kombinationsfeld ausgewählten Eintrag.

Damit beim Klick auf die Schaltfläche **cmdNeu** ein neuer, leerer Datensatz im Formular erscheint, hinterlegen wir für das **Beim Klicken**-Ereignis der Schaltfläche diese Prozedur:

```
Private Sub cmdNeu_Click()  
    DoCmd.GoToRecord Record:=acNewRec  
End Sub
```

Damit erscheint ein neuer, leerer Datensatz. Dieser ist natürlich noch nicht in der Datensatzherkunft des Kombinationsfeldes enthalten.

Daher legen wir die folgende Prozedur an, die in diesem Fall den Wert des Kombinationsfeldes auf den Wert **Null** einstellt (im Falle eines Wechsels zu einem anderen Datensatz wird dieser auch im Kombinationsfeld angezeigt):

Bild 7: Das Formular **frmMailvorlagen** in der Formularansicht

```
Private Sub Form_Current()
    Me!cboAuswahl.Requery
    Me!cboAuswahl = Me!MailvorlageID
End Sub
```

Das Aktualisieren der Datensatzherkunft des Kombinationsfeldes dient dazu, den neu angelegten Datensatz direkt nach dem Wechsel zu einem anderen Datensatz in das Kombinationsfeld aufzunehmen.

Fehlt noch die Schaltfläche zum Schließen des Formulars:

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

Vorlage anlegen

In Aktion sieht das Formular nun wie in Bild 7 aus. Für das Feld **Bezeichnung** geben Sie eine sinnvolle Bezeichnung an, unter der Sie die Mailvorlage schnell finden. Das Feld **Tabelle/Abfrage** soll die Datenquelle aufnehmen, aus

der die Daten für die Platzhalter in den nachfolgenden Eigenschaften der Mailvorlage entnommen werden sollen. Wir geben hier die Abfrage **qryKunden** an, welche die Felder der Tabellen **tblKunden** und **tblAnreden** wie in Bild 8 zusammenführt.

Das Feld **E-Mail** stattdessen Sie mit dem Namen des Feldes der Datenherkunft aus, welche die E-Mail-Adresse des Kunden liefert, in diesem Fall schlicht **EMail**. **Betreff** füllen Sie mit dem gewünschten Betreff der E-Mail. In diesem Fall benötigen wir keinen Platzhalter. Das Feld **Inhalt** jedoch enthält einen Platzhalter für das Feld **Briefanrede** der Tabelle **tblAnreden** und **Nachname** aus der Tabelle **tblKunden**. Das ist noch verständlich. Aber was geschieht mit dem folgenden Platzhalter, der ja offensichtlich keinem Feld der Datenherkunft entspricht?

```
[=Adressblock(0, 5, [KundeID])]
```

Hierbei handelt es sich nicht um einen normalen Platzhalter, sondern um eine VBA-Funktion, die den Text

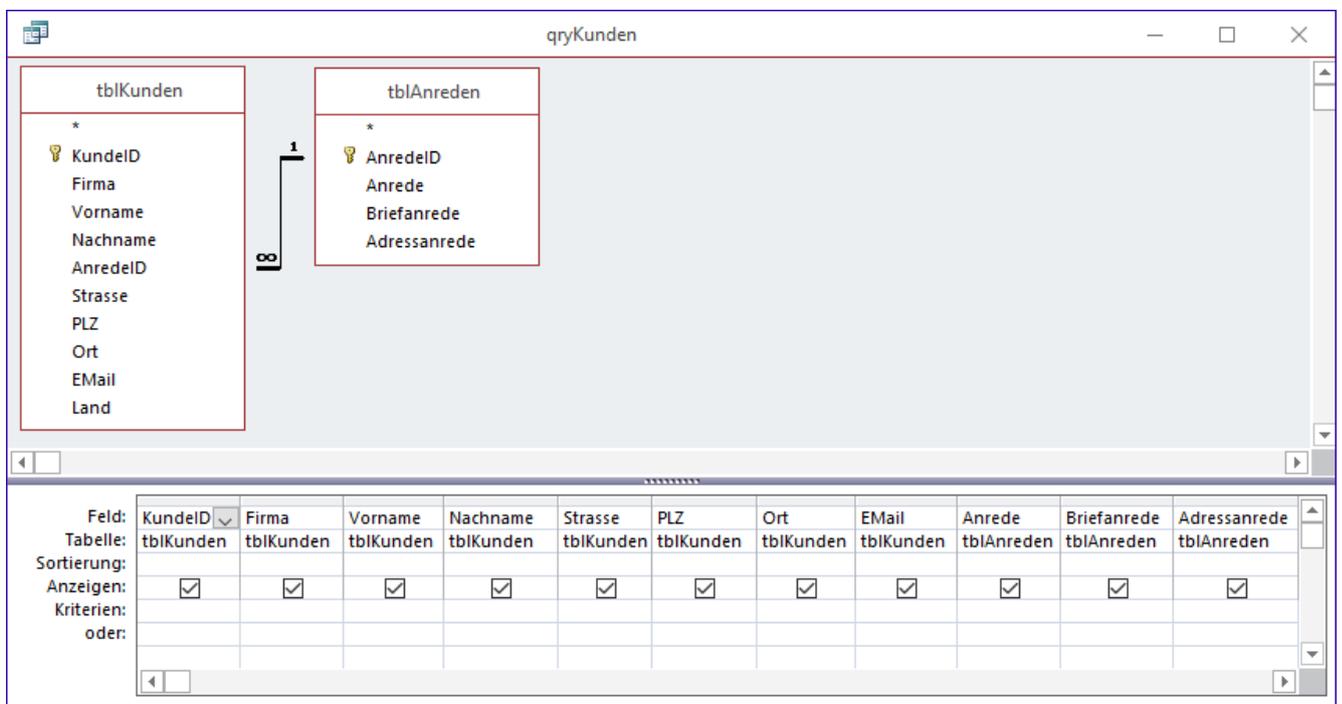


Bild 8: Die Abfrage **qryKunden** dient als Datenquelle für das Ersetzen der Platzhalter

Google-Maps als Kartendienst im Formular

Sie möchten den Anwendern Ihrer Datenbank die Möglichkeit bieten, neben den eigentlichen Adressdaten auch gleich dazugehörige Geoinformationen einzusehen? Dazu eignen sich sicher die Dienste Google-Maps oder Bing-Maps. Ein Webbrowser-Steuer-element im Formular könnte als Host für jene dienen. Doch der Weg ist steiniger als erwartet! Wie Sie die auftretenden Probleme lösen können, beschreibt dieser Beitrag.

Google Maps

Wenn Sie nach einem Ort auf der Startseite von **Google Maps** (<https://www.google.de/maps>) recherchieren, so geben Sie in das Suchfeld links oben die gewünschte Adresse ein. Google verarbeitet die Anfrage mit seinen **Geo Location Services** und zeigt das Resultat im Browser an. Dabei ändert sich die Adresszeile entsprechend. Aus der Anfrage **Duisburg Borkhofer Str.** wird dann eine **URL**, die sich aus mehreren Teilen unterschiedlicher Bedeutung zusammensetzt:

```
https://www.google.de/maps/place  
/Borkhofer+Str.,+47137+Duisburg  
/@51.4634026,6.7777953,17z  
/data=!3m1!4b1!4m5!3m4!1  
s0x47b8bfbe53cca829:0xb28d5de5549581d!8m2!3d51.463402  
6!4d6.7799893
```

Der Service hat zunächst ermittelt, dass es sich offenbar um den Ort **Duisburg** in Deutschland handeln soll, der die Postleitzahl **47137** besitzt. Die Angabe des Landes fehlt. Google geht davon aus, dass der Ort sich mit großer Wahrscheinlichkeit in dem Land befindet, von dem aus die Anfrage gesendet wurde. Dies geht etwa aus der übermittelten **IP** hervor. Der Straßename vervollständigt diesen Teil, in dem natürlich keine Leerzeichen vorkommen dürfen. Sie sind durch Pluszeichen ersetzt.

Doch damit nicht genug! Hinter einem Slash befinden sich, durch ein **@** symbolisiert, auch gleich die Geokoordinaten des gefundenen Orts. Längen- und Breitengrad kommen als Fließkommazahlen daher, wobei als Dezimaltrennzeichen der Punkt dient. Kommas trennen

hingegen die einzelnen Werte. Was jedoch hat es mit dem dritten Parameter dieses Teils auf sich, dem **17z**?

Versuch und Irrtum machen schlau, denn Google hat den Aufbau dieser Parameter nicht wirklich dokumentiert. Etliche Leute, der Autor eingeschlossen, haben teilweise herausgefunden, was sie bedeuten. Am weitesten kam wohl **M. Stickle** mit der Aufstellung in seinem Blog (<https://mstickles.wordpress.com/2015/06/12/gmaps-urls-options>). Der dritte Parameter gibt nämlich den Zoomgrad der Kartenansicht wieder.

Sie können den Wert in der Adresszeile des Browsers einfach von Hand ändern, etwa in **11z**. Der Ort wird nun von weiter oben betrachtet. Bei **4z** ist allerdings Schluss. Damit wird fast die Erdhalbkugel angezeigt. Der höchste Zoomfaktor beträgt **21z**. Bei allen anderen Werten außerhalb dieses Bereichs kommt es nicht etwa zu einer Fehlermeldung, sondern Google korrigiert den Faktor automatisch durch Auf- oder Abrunden auf den nächsten Wert.

Im dritten Teil der URL findet sich ein ominöser **data**-Parameter. Zahlen, Buchstaben und Ausrufezeichen wechseln sich scheinbar willkürlich ab. Tatsächlich haben sie eine Bedeutung, der Sie auf die Schliche kommen, wenn Sie etwa die Ansicht der Karte auf **Satellit** einstellen, oder den Layer für das Verkehrsaufkommen einschalten. Bis auf diesen **data**-Parameter bleiben alle anderen Teile der URL dabei konstant. Der **data**-Parameter bestimmt das Layout der Karte. Wenn Sie mehr über seinen Aufbau erfahren möchten, so kundschaften Sie den angegebenen Blog von **Stickle** aus. Wir kommen auf den **data**-String aber auch später noch zu sprechen.

An das Ende der URL schließt sich ein längerer String an, der wohl die **ID** der Suchanfrage enthält. Google cachet diesen Wert auch in einem **Cookie**, um Rechenzeit einzusparen. Er ist für die Suchanfrage bedeutungslos.

Sie können die URL nun dergestalt modifizieren, dass nur noch die wichtigsten Elemente enthalten sind. Das sähe dann so aus:

```
https://www.google.de/maps/place  
/@51.4634026,6.7777953,17z  
/data=!3m1!4b1!4m5!3m4!1
```

Falls die Koordinaten dieselben sind wie zuvor, so macht Google sofort wieder die ursprüngliche Version daraus, weil es sie aus dem **Cookie** ausliest. Bei geänderten Koordinaten hingegen bleibt der String meist erhalten. Allerdings ändert sich nun auch die Ansicht komplett: Das **Control Panel links** verschwindet und der Orts-**Marker** ist ebenfalls nicht mehr da.

Man kann an dieser Stelle zusammenfassen, dass sich die Google-Maps-Anfragen in einer URL unterbringen und sich dabei sowohl Zoomfaktor, wie auch Gestalt des Kartenergebnisses, beeinflussen lassen. Was will man mehr? Damit ist der erste Teil der Aufgabe in Angriff genommen, eine Google-Maps-Karte in ein Formular zu bringen.

Das Webbrowser- Steuerelement

Als Host für die **Maps**-Seiten soll ein Webbrowser-Steuerelement dienen, das Microsoft seit der Access-Version 2007 in den Fundus der Steuerelemente integriert hat. Es unterscheidet sich in nichts vom gleich-

namigen ActiveX-Steuerelement, das Windows immer mit sich führt. Lediglich die Möglichkeit, es direkt an ein Datenfeld einer Tabelle oder Abfrage zu binden, welches die zu navigierende URL enthält, ist ein Bonus-Feature.

Zum Test setzen Sie das Webbrowser-Steuerelement in ein neues Formular ein, wie dies im Formular **frmMaps** der Beispieldatenbank **GoogleMaps.accdb** geschah (s. Bild 1). Außer dem **Webbrowser Control** enthält es nur ein Textfeld rechts oben, welches den Inhalt des **URL**-Felds des aktuellen Datensatzes anzeigt. Die Tabelle **tblURLs** dient als Datenherkunft des Formulars. Sie weist neben der Autowert-**ID** nur das Feld **URL** aus und ein optionales Feld **Beschreibung**, welches hier nicht zum Einsatz kommt. An **URL** sind sowohl das Textfeld, wie auch das Webbrowser-Steuerelement gebunden.

Nach dem ersten Start präsentiert das Formular eine leere Fläche, da die erste URL den Inhalt **about:blank** hat. Würde die URL nämlich aus einem Leer-String bestehen, so käme es zu einer Fehleranzeige im Browser wegen ungültiger Adresse. Der zweite Datensatz hat den folgenden Inhalt:

```
https://www.google.de/maps/
```

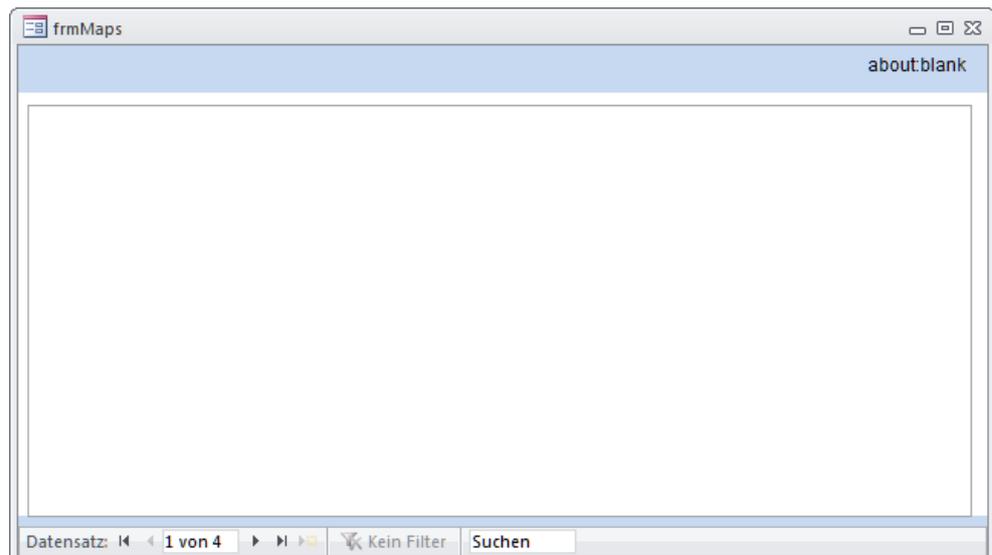


Bild 1: Im Testformular zur Tabelle **tblURLs** ist zentral ein **Webbrowser**-Steuerelement untergebracht

Sie erwarten, dass nun Google-Maps im Browser angezeigt wird? Stattdessen konfrontiert uns Google mit der Meldung aus Bild 2. Aus irgendeinem Grund ist es mit dem Webbrowser-Steuerelement nicht zufrieden und verwehrt die Navigation zu **Maps**. Doch dieses verwendet intern ja lediglich den im System installierten **Internet Explorer**, der möglicherweise in einer aktuellen Version **11** vorhanden ist (**Edge** bleibt außen vor). Warum dann diese Meldung?

Webbrowser Control: Limitierungen und deren Aufhebung

Aus Sicherheitsgründen grenzt Microsoft die Möglichkeiten im **Webbrowser Controls** stark ein. Das gilt übrigens für das Access-Steuerelement gleichermaßen wie für das ActiveX-Steuerelement. Die wichtigste Änderung besteht darin, dass das Control grundsätzlich die Version **IE7** nach außen meldet.

Genau damit ist Google nicht einverstanden, da die aktuelle Version von **Maps** einen neueren Browser für korrekte Funktion erwartet. Zudem erlaubt das Control nur eine eingeschränkte Ausführung von **JavaScript**.

Das Ende der Fahnenstange ist damit zum Glück noch nicht erreicht. Tatsächlich können Sie das Verhalten des Webbrowser Controls durch Einstellungen in der **Registry** ändern! Und dafür sind noch nicht einmal administrative Rechte erforderlich, da sich der entsprechende Zweig im Benutzerteil **HKEY_CURRENT_USER** befindet:

```
HKEY_CURRENT_USER\Software\Microsoft
    \Internet Explorer\Main\FeatureControl
```

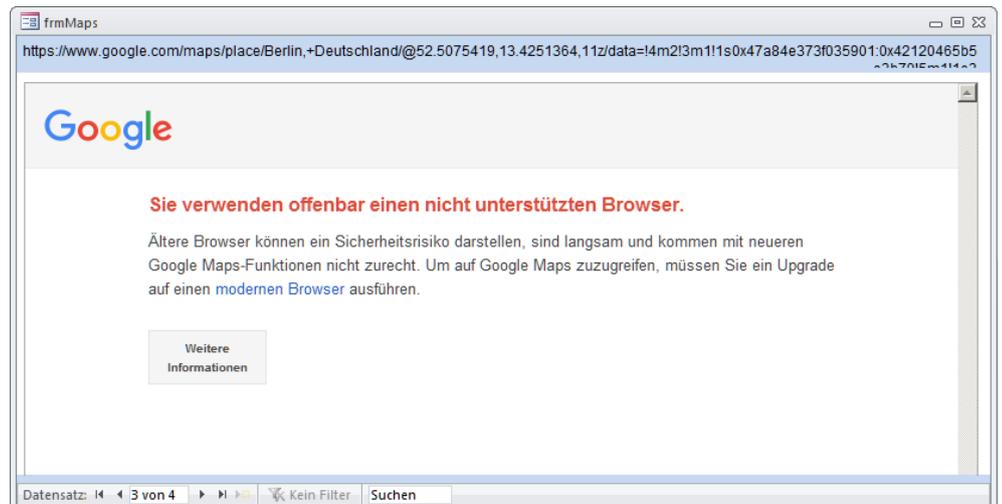


Bild 2: Statt **Maps** zeigt Google trotz korrekter URL zunächst eine Inkompatibilitätsmeldung an

Unter diesem Zweig befinden sich noch weitere Schlüssel, die verschiedene Funktionen des Controls steuern. Der wichtigste ist dabei der folgende:

```
FEATURE_BROWSER_EMULATION
```

Unter diesem Schlüssel tragen Sie einen **DWORD**-Wert ein, der den Namen **msaccess.exe** trägt. Als Wert verwenden Sie **11001**. Dieser nämlich führt dazu, dass Windows beim Start einer Access-Instanz und dem Öffnen eines Webbrowser-Steuerelements hier nachschaut, ob ein Eintrag zu finden ist. Falls Prozessname und Eintrag zueinander passen, so spendiert Windows dem zugrunde liegenden Internet Explorer die angegebene Version. Beim Wert **11001** wäre das die Version **11**. Funktionieren würde Google Maps auch noch mit der Version **9**, die dem Registry-Wert **9999** entspräche. Doch warum zu einer älteren Version greifen, wenn es eine neuere gibt? Voraussetzung ist natürlich, dass auch tatsächlich der **Internet Explorer 11** installiert ist. Passen Sie den Wert an die installierte Version an. Im **MSDN** gibt es eine Seite, die erschöpfender darüber Auskunft gibt ([https://msdn.microsoft.com/en-us/library/ee330730\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ee330730(v=vs.85).aspx)). **Edge** kann das Webbrowser Control übrigens nicht hosten. Es verwendet auch unter **Windows 10** den Kompatibilitätsmodus des **IE11**.

```
Sub SetWebControlAsIE9()  
    Dim sKey As String  
    Dim oShell As WshShell  
    Dim vKey As Variant  
    Dim nErr As Long  
    Dim vRegElement As Variant  
    Dim i As Long  
    Dim bNewStart As Boolean  
    Set oShell = New WshShell  
    sKey = "HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer" & _  
        "\Main\FeatureControl\FEATURE_BROWSER_EMULATION\msaccess.exe"  
    On Error Resume Next  
    vKey = oShell.RegRead(sKey)  
    nErr = Err.Number  
    On Error GoTo 0  
    If nErr = -2147024894 Then  
        oShell.RegWrite sKey, 11001, "REG_DWORD" '10001 / 11001 für IE10 oder IE11, 9999 für IE9  
        Debug.Print "IE9 Mode set"  
        bNewStart = True  
    Else  
        If vKey <> 11001 Then  
            oShell.RegWrite sKey, 11001, "REG_DWORD" '10001 / 11001 für IE10 oder IE11, 9999 für IE9  
            Debug.Print "IE9 Mode set"  
            bNewStart = True  
        Else  
            Debug.Print "IE9 Mode is already set"  
        End If  
    End If  
End Sub  
...
```

Listing 1: Die Prozedur **SetWebControlAsIE9** macht das Webbrowser-Control zum **Internet Explorer 11**

Bedenken brauchen Sie bei der Modifikation der Registry hier nicht zu haben. Zum einen sind Einträge unter dem Benutzerzweig recht unverdächtig, zum anderen betrifft die Änderung nur ausschließlich Access-Prozesse. Übrigens bedienen sich auch andere Entwickler dieser Möglichkeit. Wir fanden in diesem Zweig auch Einträge prominenter Anwendungen, wie **Visual Studio**, **RapidPHP** oder **Starmoney**.

Da die Änderungen an der Registry keine Administratorrechte erfordern, können Sie auch von Access aus über VBA getätigt werden. Dies geschieht auch in der Beispieldatenbank. Das Intro-Formular ruft beim Klick auf den grünen Button die Funktion **SetWebControlAsIE9**



Bild 3: Automatische Aufforderung zum Neustart von Access über die Prozedur **SetWebControlAsIE9** und eine Messagebox

des Moduls **mdlWebcontrol** auf. Diese liest zunächst aus der Registry aus, ob der Wert für die **Browser Emulation** bereits gesetzt ist. Falls nicht, so holt sie das umgehend nach. Allerdings wirkt sich die Änderung noch nicht unmittelbar aus. Erst nach einem Neustart von Access gibt das

Webbrowser Control die Version **IE11** aus. Deshalb empfiehlt die Routine in diesem Fall auch gleich den Neustart (s. Bild 3) und bewerkstelligt diesen auch noch. Schauen wir uns diese Routine (ersterer Teil in Listing 1) einmal genauer an.

Da Access und VBA keine Methoden zum Auslesen oder Schreiben der Registry haben, benötigen wir den Verweis auf eine externe Komponente. Für unseren einfachen Zweck gelingt das am effizientesten mit der Bibliothek **Windows Script Host Object Model**, die im Objektkatalog nachher den Namen **IWshRuntimeLibrary** trägt und auf der Datei **wshom.ocx** beruht. Diese Bibliothek ist Bestandteil von Windows und kann deshalb bedenkenlos zum Einsatz kommen. Ihre Klasse **WshShell** enthält Methoden zur Modifikation der Registry.

Die Objektvariable **oShell** nimmt eine Instanz dieser Klasse per **New** auf. Der Zweig der Registry ist in der String-Variablen **sKey** gespeichert, und zwar inklusive des Wertnamens. Die Funktion **RegRead** auf den Schlüssel gibt dann den Wert als Variant in der Variablen **vKey** zurück. Da es zum Fehler kommt, wenn der Registry-Schlüssel nicht existiert, ist der Aufruf der Funktion in eine Fehlerbehandlung über **On Error Resume Next** eingebettet. Der Fehlerwert beträgt in diesem Fall **-2147024894**, der in der folgenden Bedingung dazu führt, dass der Schlüsselwert über die Methode **RegWrite** neu geschrieben wird. Die Klasse **WshShell** erlaubt also einen komfortablen Zugriff auf die Registry mit sehr wenig Aufwand.

Existiert der Schlüsselwert zwar, so ist zu prüfen, ob er dem gewünschten entspricht. Das übernehmen die folgenden Zeilen. Weicht der Inhalt von **vKey** von **11001**

```
Sub RestartAccess()
    If MsgBox("Die Datenbank und Access müssen neu gestartet werden," & vbCrLf & _
        "weil die Einstellungen für das WebControl geändert wurden." & vbCrLf & _
        "Jetzt neu starten?", vbQuestion Or vbYesNo, "Bestätigen:") = vbYes Then

        Dim sExec As String
        Dim oShell As New WshShell

        sExec = Chr$(34) & SysCmd(acSysCmdAccessDir) & "msaccess.exe" & Chr$(34) & _
            " " & Chr$(34) & CurrentDb.Name & Chr$(34)

        oShell.Exec sExec
        Application.Quit acQuitSaveNone
    End If
End Sub
```

Listing 2: Die Routine **RestartAccess** übernimmt gegebenenfalls den Neustart von Access

ab, so erfolgt abermals das Schreiben in den Schlüssel. Ansonsten passiert außer der informativen Ausgabe im VBA-Direktfenster weiter nichts. Wichtig aber ist noch eine Kleinigkeit: Wurde in die Registry geschrieben, so erhält die Bool-Variable **bNewStart** den Wert **True**. Dieser Wert wird am Ende der Routine ausgewertet:

```
If bNewStart Then
    RestartAccess
End If
```

RestartAccess ist eine weitere Prozedur im Modul, die den Neustart von Access veranlasst (s. Listing 2). Wird die Nachfrage der **MsgBox** bestätigt, so wird wieder das **Shell**-Objekt bemüht. Seine Methode **Exec** entspricht ungefähr der **Shell**-Methode von VBA, ist aber etwas einfacher zu handhaben. In **sExec** wird ein String zusammengestellt, der eine neue Instanz von Access startet, wobei über **SysCmd** erfahren wird, wo sich die **msaccess.exe** befindet. Als Parameter für den Prozess wird der Name der aktuellen Datenbank (**CurrentDb.Name**) übergeben. In der Folge startet eine zusätzliche Instanz der Datenbank. Die aktuelle Instanz wird aber über **Quit** geschlossen. Das passiert so schnell, dass es kaum wahrnehmbar ist. Im Ergebnis befinden wir uns in der Datenbank mit dem neu aktivierten **Webbrowser Control**.

Die Routine **SetWebControlAsIE9** enthält noch einen zweiten Teil, der drei weitere Einstellungen für das Control vornimmt. Das passiert analog zum ersten Teil. Der Unterschied besteht nur darin, dass der Code-Block nicht dreimal wiederholt wird, sondern eine Schleife auf das Array **vRegElement** die einzelnen Registry-Schlüssel anspricht:

```
vRegElement = Array("FEATURE_BLOCK_LMZ_SCRIPT",  
    "FEATURE_BLOCK_LMZ_OBJECT",  
    "FEATURE_BLOCK_LMZ_IMG")  
For i= 0 To 2  
    sKey = "HKEY_CURRENT_USER\Software\Microsoft\" & _  
        "Internet Explorer\Main\FeatureControl\" & _  
        vRegElement(i) & "\msaccess.exe"  
    ...  
    oShell.RegWrite sKey, 1, "REG_DWORD"  
    ...  
Next i
```

Falls es noch nicht aufgefallen sein sollte: **RegWrite** übernimmt in einem Rutsch die Anlage des Registry-Schlüssels und des Werts! Einfacher geht's nicht! Bei allen alternativen Methoden muss das separat erfolgen.

Neben den Einstellungen, die so über die Registry vorgenommen werden können, gibt es noch einige weitere, die die weitgehende Übereinstimmung des Webbrowser-Steuerelements mit dem Internet Explorer herstellen können. Dazu jedoch muss eine Windows-API-Funktion bemüht werden, die sich **CoInternetSetFeatureEnabled** nennt. Das Setzen dreier neuer Eigenschaften ist in die Prozedur **ChangeWebControlFeature** ausgelagert (s. Listing 3).

Die Konstanten für Steueranweisungen, wie **FEATURE_BEHAVIOURS**, stehen im Kopf des Moduls. Sie bestimmen das anzusprechende Feature des Webbrowser Controls.

```
Public Declare Function CoInternetSetFeatureEnabled Lib "urlmon.dll" _  
    (ByVal FeatureEntry As eINTERNETFEATURELIST, ByVal Flags As eFEATURESetting, ByVal bEnable As Long) As Long  
  
Public Declare Function CoInternetIsFeatureEnabled Lib "urlmon.dll" _  
    (ByVal FeatureEntry As eINTERNETFEATURELIST, ByVal Flags As eFEATURESetting) As Long  
  
Sub ChangeWebControlFeature()  
    Dim ret As Long  
    Dim lret As Long  
    If CoInternetIsFeatureEnabled(FEATURE_BEHAVIORS, SET_FEATURE_ON_PROCESS) <> 0 Then  
        ret = CoInternetSetFeatureEnabled(FEATURE_BEHAVIORS, SET_FEATURE_ON_PROCESS, 1)  
        lret = ret  
    End If  
    If CoInternetIsFeatureEnabled(FEATURE_ZONE_ELEVATION, SET_FEATURE_ON_PROCESS) <> 0 Then  
        ret = CoInternetSetFeatureEnabled(FEATURE_ZONE_ELEVATION, SET_FEATURE_ON_PROCESS, 1)  
        lret = lret + ret  
    End If  
    If CoInternetIsFeatureEnabled(FEATURE_RESTRICT_ACTIVEXINSTALL, SET_FEATURE_ON_PROCESS) <> 0 Then  
        ret = CoInternetSetFeatureEnabled(FEATURE_RESTRICT_ACTIVEXINSTALL, SET_FEATURE_ON_PROCESS, 0)  
        lret = lret + ret  
    End If  
    Debug.Print IIf(lret = 0, "Features adjusted", "Feature adjustment failed")  
End Sub
```

Listing 3: ChangeWebcontrolFeature setzt neue Eigenschaften für das Webbrowser Control per API

Für alle drei Aufrufe ist die Anweisung **SET_FEATURE_ON_PROCESS** angegeben. Das besagt, dass die Eigenschaft nur auf den laufenden Prozess, also die Access-Instanz, wirken soll. Dabei fragt die zweite API-Funktion **ControlPanelFeatureEnabled** jeweils zuerst ab, ob die Eigenschaft schon gesetzt ist. Die Prozedur **ChangeWebControlFeature** muss, da sie nur den Prozess betrifft, immer beim Start der Datenbank ausgeführt werden. Auf die Bedeutung der einzelnen Features möchten wir an dieser Stelle nicht weiter eingehen. In der MSDN ([https://msdn.microsoft.com/en-us/library/ee330720\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ee330720(v=vs.85).aspx)) sind die Features näher erläutert.

Zurück zu unserem Testformular. Nach dem Durchlaufen der Anpassungsroutinen und Neustart von Access rufen Sie **frmMaps** erneut auf. Und nun präsentiert sich beim zweiten Datensatz tatsächlich die Startseite von Google Maps (s. Bild 4). Alles ok! Der dritte Datensatz verweist direkt auf eine zuvor in Maps im Browser ermittelte Ortsadresse:

<https://www.google.com/maps/place/Berlin,+Deutschland/@52.5075419,13.4251364,11z/data=!4m2!3m1!>

Auch diese wird nun korrekt angesteuert (s. Bild 5). Da es sich nicht um eine genaue Adresse handelt, sondern

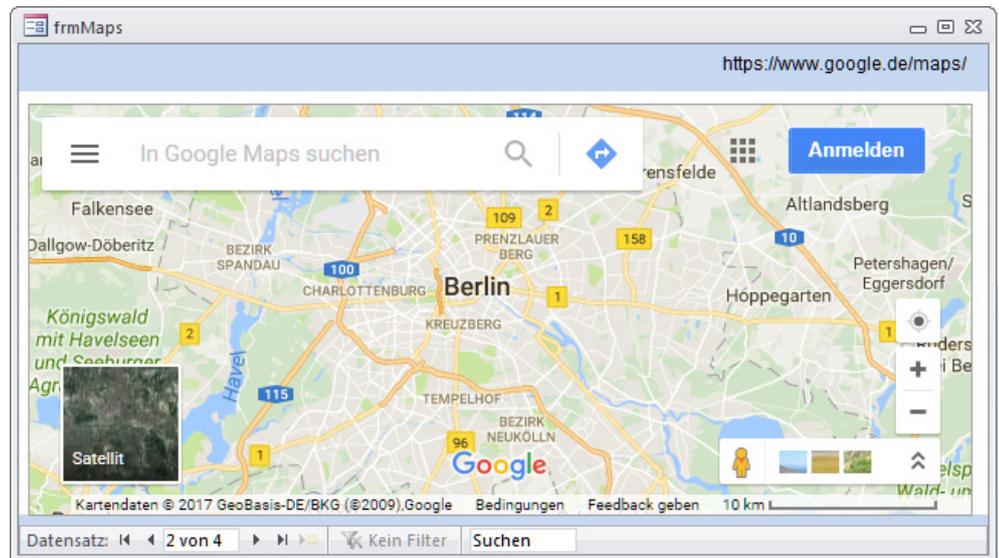


Bild 4: Google Maps nach der Anpassung des Webbrowser-Steuerlements über Registry und API

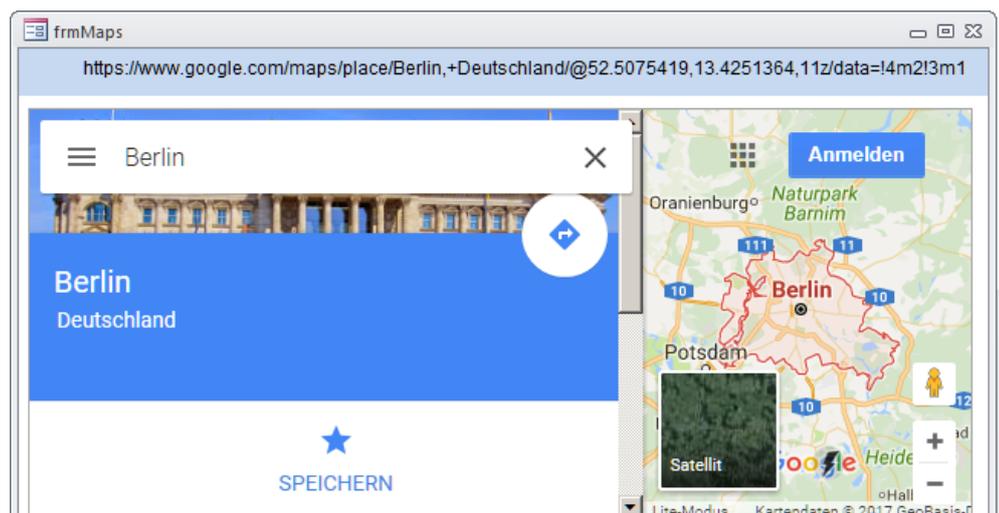


Bild 5: Google Maps zeigt nach Ansteuerung einer Such-URL das richtige Ergebnis mit Steuer-Panel

eine Stadt, wird diese in der Karte als Gebiet umrahmt. Störend ist bei dieser Formulargröße allerdings, dass das **Control Panel** von Google Maps mehr Platz einnimmt als der Kartenausschnitt. Leider lässt sich dieses Panel nach allem bisherigen Wissen nicht über Steuerparameter ausblenden. Da bleibt nur der Klick auf den Schließen-Button des Panels, der dann allerdings auch die Ortsmarkierung in der Karte entfernt ...

Bei genauerem Hinsehen finden Sie in der Karte einen kleinen Blitz links neben den +/-Elementen zu Zoomen

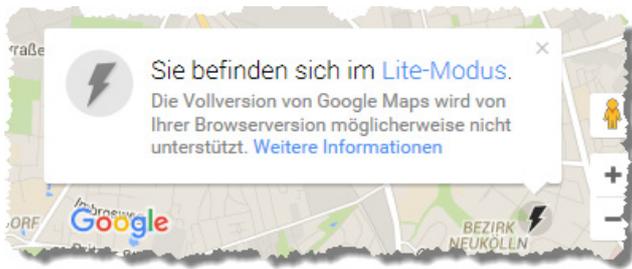


Bild 6: Trotz Webbrowser-Control-Anpassungen schaltet Google-Maps dennoch in den sogenannten Lite-Modus

der Karte eingebaut. Der fehlt beim Aufruf der Seite in einem Browser. Ein Klick auf ihn fördert ein Popup zutage, welches die Meldung in Bild 6 enthält. Offenbar hat Google nun doch ermittelt, dass hier nicht mit einem vollwertigen Browser gearbeitet wird. Das liegt daran, dass bestimmte externe Objekte des Web-Dokuments hier nicht über **JavaScript** ansprechbar sind. Wir konnten im **Lite-Modus** jedoch keine wesentlichen Einschränkungen feststellen. Alle Ansichten, inklusive **Street View**, konnten anstandslos eingeschaltet werden und verhielten sich erwartungsgemäß.

Adressen im Webbrowser-Steuer-element automatisch über Formular ansteuern

Mit den bisherigen Erkenntnissen im Gepäck kann es an die Realisierung einer Beispiellösung gehen. Ziel ist es, ein einfaches Adressformular zu Kunden der Datenbank um eine Kartenansicht zu bereichern. Hierfür kommt das Datenmodell in Bild 7 zum Einsatz. In der Tabelle **tblKunden** sind Adressen gespeichert.

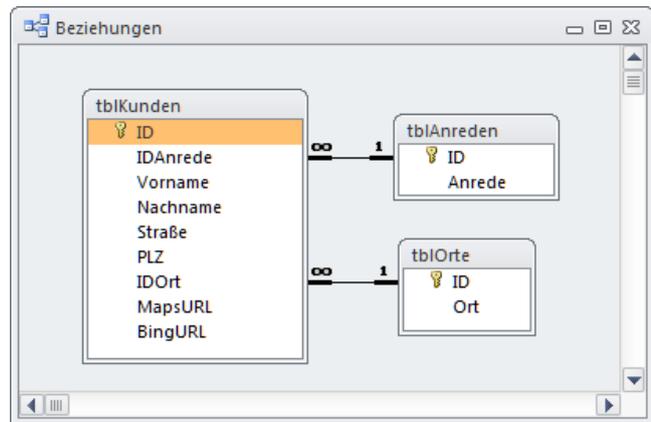


Bild 7: Beziehungs-Layout der Beispieldatenbank (Kundenadressen)

Feldname	Feldtyp	Beschreibung
ID	AutoWert	
IDAnrede	Zahl	
Vorname	Text	
Nachname	Text	
Straße	Text	
PLZ	Text	
IDOrt	Zahl	
MapsURL	Text	
BingURL	Text	

Bild 8: Der Entwurf der Tabelle **tblKunden** zeigt die **URL**-Felder

tblAnreden und **tblOrte** sind verknüpfte Detailtabellen. Sie sind auch im Entwurf der Kundentabelle (s. Bild 8) in den Feldern **IDAnrede** und **IDOrt** als Nachschlagetabellen für die hier verwendeten Kombinationsfelder eingetragen.

ID	IDAnrede	Vorname	Nachname	Straße	PLZ	IDOrt	MapsURL	BingURL
1158	Dr. jur.	Per-Olof	Zutz	Northeimer Wende 24	30419	Hannover	about:blank	about:blank
1199	Dr.	Danela	Zwania	Dettersweg 5	63589	Linsengericht	about:blank	about:blank
3882	Dr. jur.	Karan	Zwenzner	Gravensteiner Str. 6	25764	Wesselburen	about:blank	about:blank
433	Dr. jur.	Leif-Erik	Zwiersch	Grohmannstr. 2	80933	München	about:blank	about:blank
5261	Dr.	Kerstin	Zwingenberg	Kaiserin-Augusta-Str. 34A	12103	Berlin	about:blank	about:blank
5158	Dr. jur.	Lucas	Zygon	Pilgramer Str. 244	12623	Berlin	about:blank	about:blank
1425	Dr.	Jutta-Renate	Zywicki	Freiherr-vom-Stein-Str. 17	67466	Lambrecht (Pfalz)	about:blank	about:blank
724	Dr.	Estela	Adamiuk	Gartenweg 62	75181	Pforzheim	https://www.google.com/maps/@48.8980004,8.748593,15	about:blank
1251	Dr. jur.	Franco	Ababneh	Hauptstr. 74	66851	Queidersbach	https://www.google.com/maps/@49.377572,7.6493634,13	about:blank
1534	Dr.	Mabell	Abdel Aziz	An den Pappeln 2A	61440	Oberursel (Taunus)	https://www.google.com/maps/@50.18532,8.58108,15z/d	https://www.bing.com/maps?where1=An%20den%20P
62	Dr.	Gerhild	Acikgöz	Florianssiedlung 12	96342	Stockheim	https://www.google.com/maps/@50.2919379,11.2921136,	https://www.bing.com/maps/default.aspx?where1=Fl
2523	Dr.	Bozena	Acikkol	Bismarckstr. 30	57250	Netphen	https://www.google.com/maps/@50.91236,8.05687,15z/d	about:blank
5224	Dr. jur.	Chester	Dabernig	Ostendstr. 25	12459	Berlin	https://www.google.com/maps/@52.4581876,13.5330504,	about:blank
4963	Dr. jur.	Horstdieter	Cabanis	Schmiljanstr. 25	12161	Berlin	https://www.google.com/maps/@52.5073181,13.3745434,	about:blank
2852	Dr.	Lisa	Dabels	Clubheim 26	11111	Neuss	https://www.google.com/maps/@52.5594696,13.1190944,	about:blank
5650	Dr.	Florenzia	Abd El-Hakim	An der Priesterkoppel 39	13158	Berlin	https://www.google.com/maps/@52.5959671,13.3836114,	https://www.bing.com/maps?where1=An%20der%20P
6850	Dr. jur.	Xaver	Aberschanska	Schäferstr. 42	19053	Schwerin	https://www.google.com/maps/@53.6234129,11.4039678,	http://www.bing.com/maps/default.aspx?where1=Sch
3106	Dr.	Nadiye	Auracher	Hinter den Brüdern 2	38640	Goslar	https://www.google.com/maps/place/Hinter%20den%20	about:blank
5926	Dr. jur.	Adellino	Auscher	Wundtstr. 5	14059	Berlin	https://www.google.com/maps/place/Wundtstr.%205,%	about:blank
*	(Neu)						about:blank	about:blank

Bild 9: Die Tabelle **tblKunden** ist mit etwa 7000 Fake-Adressen gefüllt, die zusätzlich die **URL** zu Google- oder Bing-Maps aufnehmen können

Dynamischer Adressblock

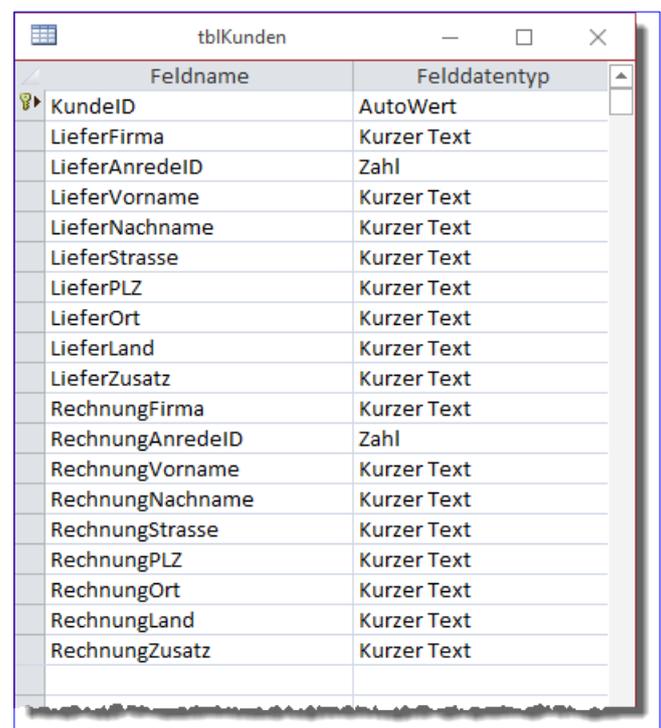
Wenn Sie einen Adressblock etwa für die Anzeige in einem Bericht oder für anderweitige Nutzung zusammenstellen wollen, stoßen Sie mitunter auf eine anspruchsvolle Aufgabe. Dummerweise sind die Kundentabellen, aus denen die Adressen bezogen werden sollen, immer unterschiedlich aufgebaut. Noch dazu gibt es manchmal nicht nur eine einzige Adresse, sondern Liefer- und Rechnungsadresse. Dabei treten wiederum Unterschiede auf, wo die primäre Adresse eingetragen wird – also die Adresse, die als Rechnungs- und Lieferadresse genutzt wird, wenn nur eine Lieferadresse oder nur eine Rechnungsadresse vorliegt. Dieser Beitrag zeigt eine Funktion, mit der Sie die Adressblöcke für Ihren Einsatzzweck komfortabel zusammenstellen können.

Beispieltabelle

Ein Beispiel für eine Kundentabelle mit Rechnungs- und Lieferdaten sieht etwa wie in Bild 1 aus. Wenn für einen Kunden eine Lieferanschrift und eine Rechnungsanschrift vorliegt ist der Umgang mit den Daten in dieser Tabelle recht unkompliziert: Sie ermitteln die Lieferadresse aus den oberen Feldern und die Rechnungsadresse aus den unteren Feldern der Tabelle.

Etwas komplizierter wird es, wenn Liefer- und Rechnungsadresse identisch sind. Dann gilt es zunächst festzustellen, ob diese in den Feldern vorliegt, die mit **Liefer...** beginnen, oder in denen, die mit **Rechnung...** beginnen.

Sie als menschlicher Beobachter erkennen dies normalerweise mit einem Blick, aber wenn wir später automatisiert bestimmen wollen, welcher der beiden Blöcke mit Adressinformationen die gesuchten Informationen liefert, brauchen wir ein auswertbares Kriterium. Wir können an dieser Stelle festlegen, dass zumindest für deutsche Adressen mindestens eine PLZ oder ein Ort vorliegen müssen. Bei den anderen Daten kann es vorkommen, dass das eine oder andere Feld fehlt. Eine Adresse kann zwar Firma und Name einer Person enthalten, aber es kann auch nur eine Firma oder nur eine Person geben. Und auch die Straße muss nicht zwangsläufig angegeben werden, denn es gibt auch Adressen, die nur über Firma, Postleitzahl und Ort definiert werden.



Feldname	Felddatentyp
KundeID	AutoWert
LieferFirma	Kurzer Text
LieferAnredeID	Zahl
LieferVorname	Kurzer Text
LieferNachname	Kurzer Text
LieferStrasse	Kurzer Text
LieferPLZ	Kurzer Text
LieferOrt	Kurzer Text
LieferLand	Kurzer Text
LieferZusatz	Kurzer Text
RechnungFirma	Kurzer Text
RechnungAnredeID	Zahl
RechnungVorname	Kurzer Text
RechnungNachname	Kurzer Text
RechnungStrasse	Kurzer Text
RechnungPLZ	Kurzer Text
RechnungOrt	Kurzer Text
RechnungLand	Kurzer Text
RechnungZusatz	Kurzer Text

Bild 1: Beispiel für eine Kundentabelle

Also legen wir an dieser Stelle fest, dass die PLZ unser Kriterium sein soll, ob in den **Liefer...**-Feldern oder den **Rechnung...**-Feldern eine Adresse vorliegen soll.

In der Tabelle **tblKunden** finden Sie übrigens die beiden Felder **LieferAnredeID** und **RechnungAnredeID**, die mit einer weiteren Tabelle namens **tblAnreden** verknüpft sind.

KundeID	LieferFirma	LieferPLZ	RechnungFirma	RechnungPLZ	RechnungOrt	RechnungLand	RechnungZusatz
1	André Minhorst Verlag	47137		47138	Duisburg	Deutschland	
2	Klaus Müller GbR	12345					
3			Claudia Schäfer GmbH	12321	Testhausen	Deutschland	

Bild 2: Verschiedene Konstellationen von Adressen

Im Folgenden wollen wir durch geeignete Konfigurationen und eine entsprechende VBA-Funktion sicherstellen, dass alle Varianten von Adressen wie in Bild 2 abgebildet in entsprechende Adressblöcke umgewandelt werden können. Das heißt, dass wir aus dem vollständigen Datensatz ganz oben die Lieferadresse aus den **Liefer...**-Feldern zusammenstellen und die Rechnungsadresse aus den **Rechnung...**-Feldern. Für den zweiten Datensatz wollen wir die Lieferadresse aus den **Liefer...**-Feldern zusammenstellen – genau wie die Rechnungsadresse. Der dritte Datensatz enthält nur Daten in den **Rechnung...**-Feldern, also sollen sowohl die Lieferadresse als auch die Rechnungsadresse aus diesen Feldern erstellt werden.

Einfache Benutzerführung

Der Benutzer/Entwickler, der mit dieser Lösung arbeitet, soll es möglichst einfach haben, den Adressblock zu definieren. Daher wollen wir einige Daten in einer Konfigurationstabelle festlegen, die der Benutzer nach Wunsch bearbeiten kann.

Diese Tabelle sieht im Entwurf wie in Bild 3 aus. Das erste Feld enthält die Bezeichnung der jeweiligen Konfiguration. Das zweite und dritte Feld dienen der Eingabe eines Schemas für die Platzhalter, die in der Lieferadresse und der Rechnungsadresse verwendet werden sollen (mehr dazu weiter unten).

Feldname	Felddatentyp	Beschreibung (optional)
KonfigurationID	AutoWert	Primärschlüsselfeld der Tabelle
Bezeichnung	Kurzer Text	Bezeichnung der Konstellation
KonfigurationLieferung	Langer Text	Konfiguration mit Platzhaltern für die Lieferadresse
KonfigurationRechnung	Langer Text	Konfiguration mit Platzhaltern für die Rechnungsadresse
Identifizierer	Kurzer Text	Feld der Quelltable für Auswahl von Rechnungs- oder Lieferdaten
TabelleAbfrage	Kurzer Text	Datenherkunft für die Adressblöcke (Tabelle oder Abfrage)
Standard	Kurzer Text	Angabe, ob die Liefer- oder die Rechnungsadresse immer vorliegt
Primärschlüsselfeld	Kurzer Text	Primärschlüsselfeld der Quelltable oder -abfrage
NichtAuszugebendeLaender	Kurzer Text	Liste der Länder, für die das Land nicht angegeben werden soll
FelderMitLaendern	Kurzer Text	Feld, welches das Land enthält

Feldeigenschaften	
Allgemein	Nachschlagen
Steuerelement anzeigen	Kombinationsfeld
Herkunftstyp	Wertliste
Datensatzherkunft	"Lieferadresse"; "Rechnungsadresse"
Gebundene Spalte	1
Spaltenanzahl	1
Spaltenüberschriften	Nein
Spaltenbreiten	
Zeilenanzahl	16
Listenbreite	Automatisch
Nur Listeneinträge	Ja
Mehrere Werte zulassen	Nein
Wertlistenbearbeitung zu	Ja
Bearbeitungsformular für	
Nur Datensatzherkunft	Nein

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Bild 3: Konfigurationstabelle

Das Feld **Standard** legt fest, ob die Quelltable standardmäßig die Lieferdaten enthält oder die Rechnungsdaten und dass, sofern die jeweils andere Adresse fehlt, die als Standard festgelegte Adresse angelegt wird. Dieses Feld legen wir als Nachschlagefeld aus, allerdings ohne fremde Tabelle als Datensatzherkunft. Stattdessen legen wir für die Eigenschaft **Herkunftstyp** den Wert **Wertliste** fest und geben für die Eigenschaft **Datensatzherkunft** die folgende Liste an:

"Lieferadresse"; "Rechnungsadresse"

Dadurch zeigt das Feld diese beiden Einträge als mögliche Werte an. Das Feld **Identifizierer** nimmt den Namen des Feldes auf, das entscheidet, ob die Daten der Liefer- oder Rechnungsadresse vorliegen oder nicht – wie oben beschrieben wollen wir hier das PLZ-Feld für die Liefer- oder die Rechnungsadresse verwenden. Wenn die Daten für die Lieferadresse immer eingetragen werden und die der Rechnungsadresse nur, wenn diese von der Lieferadresse abweicht, tragen wir etwa das Feld **RechnungPLZ** in das Feld **Identifizierer** ein, sonst das Feld **LieferPLZ**.

Das Feld **TabelleAbfrage** nimmt den Namen der Tabelle oder Abfrage auf, die als Datenquelle für die Adressblöcke verwendet werden soll. **Primärschlüsselfeld** legt fest, welches Feld das Primärschlüsselfeld der in **TabelleAbfrage** festgelegten Tabelle oder Abfrage ist. Die folgenden beiden Felder sind vor allem interessant, wenn der Versand nicht nur in das gleiche, sondern auch in andere Länder erfolgt. Wenn Sie etwa eine Sendung innerhalb Deutschlands verschicken, geben Sie nur Firma, Name, Straße, PLZ und Ort im Adressblock an. Wenn die Sendung hingegen in das Ausland geht, fügen Sie noch eine Zeile mit dem Zielland hinzu. Damit Sie definieren können, wann das Zielland angegeben werden soll und wann nicht, tragen Sie unter **NichtAuszugebendeLaender** eine Liste der Länder ein, für die das Land nicht in die Zieladresse übernommen werden soll. Unter **FelderMitLaendern** geben Sie dann ein, welche Felder die Länder enthalten, in unserem Beispiel also etwa **LieferLand** und **RechnungLand**. Bei mehreren Einträgen trennen Sie diese durch Semikola.

Formular zur Verwaltung der Konfigurationen

Diese Daten verwalten wir in einem Formular namens **frmKonfiguration**. Es ist an die Tabelle **tblKonfiguratio-**

Bild 4: Formular zum Verwalten der Konfigurationen

nen gebunden und sieht in der Formularansicht wie in Bild 4 aus. Besonderheiten gibt es bei diesem Formular nicht – die Steuerelemente werden nach dem Hineinziehen in den Detailbereich lediglich noch in der Größe und Position angepasst.

Anlegen einer Konfiguration

Noch nicht beschrieben haben wir den Inhalt der beiden Felder **KonfigurationLieferung** und **KonfigurationRechnung**. Hier gibt es verschiedene Möglichkeiten, die Felder der unter **TabelleAbfrage** als Datenquelle verwendeten Tabelle oder Abfrage einzutragen. Dazu schauen wir uns zunächst die Abfrage **qryKunden** an, welche die Daten der Tabelle **tblKunden** und **tblAnreden** in eine für unsere Zwecke optimal nutzbare Form bringt. Die Abfrage sieht im Entwurf wie in Bild 5 aus. Um diese zu erstellen, fügen Sie zunächst die Tabelle **tblKunden** und dann zwei Mal

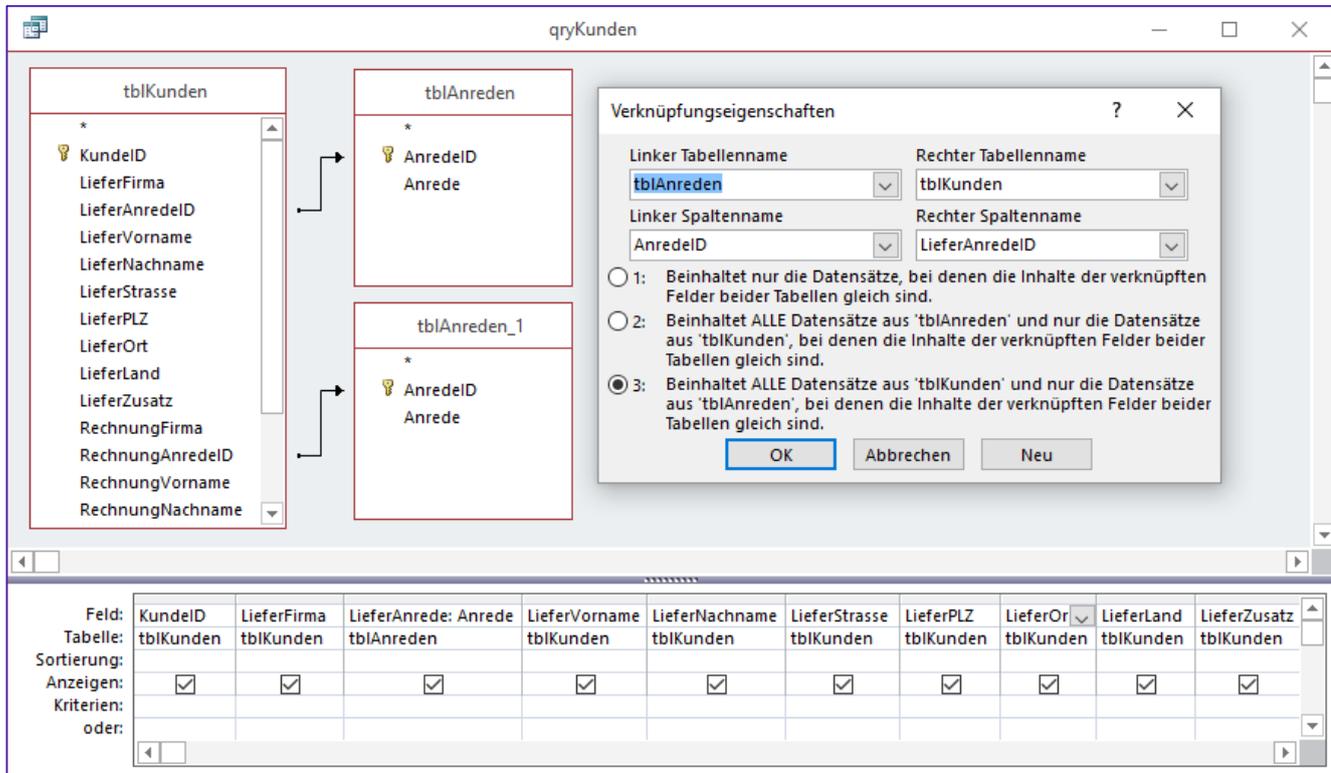


Bild 5: Abfrage, welche die Daten der beteiligten Tabellen zusammenführt

die Tabelle **tblAnreden** zum Entwurf der Abfrage hinzu. Die erste Verknüpfung zwischen dem Feld **LieferAnredeID** der Tabelle **tblKunden** und dem gleichnamigen Feld der Tabelle **tblAnreden** wird noch automatisch erstellt, die Verknüpfung zwischen dem Feld **RechnungAnredeID** der Tabelle **tblKunden** und dem Feld **AnredeID** der zweiten Instanz der Tabelle **tblAnreden** im Entwurf müssen Sie per Drag and Drop zwischen den beiden Feldern selbst hinzufügen.

Wichtig ist, dass Sie die Verknüpfungseigenschaften der beiden Verknüpfungen bearbeiten. Wenn Sie dies nicht tun und enthält beispielsweise das Feld **RechnungAnredeID** eines Datensatzes der Tabelle **tblKunden** keinen Wert und somit auch keine Verknüpfung zur Tabelle **tblAnreden**, dann wird dieser Datensatz der Tabelle **tblKunden** nicht als Ergebnis der Abfrage **qryKunden** zurückgeliefert. Damit dies dennoch geschieht, stellen Sie in den Verknüpfungseigenschaften der Beziehung die folgende Option ein:

Beinhaltet ALLE Datensätze aus 'tblKunden' und nur die Datensätze aus 'tblAnreden', bei denen die Inhalte der verknüpften Felder beider Tabellen gleich sind.

Dann werden auch Daten aus **tblKunden** angezeigt, zu denen es nicht für beide Felder **LieferAnredeID** und **RechnungAnredeID** einen verknüpften Datensatz in der Tabelle **tblAnreden** gibt.

Die beiden verknüpften Tabellen ziehen wir übrigens hinzu, um jeweils das Feld **Anrede** statt der Felder **LieferAnredeID** und **RechnungAnredeID** für die Zusammenstellung der Adressblöcke bereitzustellen.

Damit diese dann später auch unterschieden werden können, obwohl sie den gleichen Feldnamen besitzen, stellen wir diesen noch jeweils den Alias **LieferAnrede** beziehungsweise **RechnungAnrede** voran, also etwa so:

LieferAnrede: Anrede

Nun erstellen wir die Vorlage für die beiden Konfigurationen für die Liefer- und die Rechnungsadresse. Dazu geben Sie den Platzhalter für einen Feldnamen aus der Datenquelle grundsätzlich in eckigen Klammern an.

Wenn Sie also sicher sind, dass alle Felder der als Datenquelle verwendeten Tabelle ausgefüllt sind, können Sie eine Vorlage wie die folgende verwenden:

```
[LieferFirma]
[LieferAnrede] [LieferVorname] [LieferNachname]
[LieferZusatz]
[LieferStrasse]
[LieferPLZ] [LieferOrt]
[LieferLand]
```

Für die Rechnungsadresse sieht dies analog so aus:

```
[RechnungFirma]
[RechnungAnrede] [RechnungVorname] [RechnungNachname]
[RechnungZusatz]
[RechnungStrasse]
[RechnungPLZ] [RechnungOrt]
[RechnungLand]
```

Nun kommt es allerdings vor, dass Informationen fehlen. Das ist kein Problem, wenn die komplette Zeile fehlt – diese wird dann von unserer Funktion einfach weggelassen. Fehlt also etwa die Zeile mit dem Platzhalter **LieferZusatz**, weil das gleichnamige Feld für den aktuellen Datensatz keinen Wert enthält, dann wird die Zeile einfach weggelassen.

Aber was, wenn beispielsweise das Feld **Firma** fehlt? In diesem Fall würde ja die Zeile mit der Firma wegfallen – auch das wäre kein Problem. Aber in diesem Fall möchten wir zum Beispiel die Anrede in die erste Zeile holen, also etwa so:

```
Herrn
André Minhorst
```

```
Borkhofer Str. 17
47137 Duisburg
```

Mit Firma soll der Adressblock so aussehen:

```
André Minhorst Verlag
Herrn André Minhorst
Borkhofer Str. 17
47137 Duisburg
```

Dafür haben wir uns eine spezielle Notation für eine **If...Then...Else**-Bedingung ausgedacht. Sie können in einer eckigen Klammer nicht nur den Namen eines Feldes angeben, sondern bis zu drei durch das Pipe-Zeichen (**|**) getrennte Felder. Der erste Eintrag gibt dabei das Feld an, dessen Inhalt auf den Wert **Null** beziehungsweise auf eine leere Zeichenfolge geprüft wird.

Der zweite Eintrag nimmt das Feld auf, das im Adressblock ausgegeben werden soll, wenn das im ersten Eintrag angegebene Feld nicht leer ist. Der dritte Eintrag nimmt das Feld auf, das im Adressblock erscheinen soll, wenn das im ersten Eintrag angegebene Feld leer bleibt. Der folgende Platzhalter sorgt also dafür, dass die Funktion prüft, ob das Feld **LieferFirma** einen Wert enthält. Falls ja, wird der Inhalt dieses Feldes ausgegeben, falls nicht, der Wert des Feldes **Anrede**:

```
[LieferFirma|Lieferfirma|LieferAnrede]
```

Wenn wir im gleichen Zuge dafür sorgen wollen, dass, wenn das Feld **LieferFirma** einen Wert enthält, die Anrede zusammen mit dem Vornamen und dem Nachnamen in die folgende Zeile gesetzt wird, verwenden wir die folgende Konstellation von Platzhaltern:

```
[LieferFirma|LieferAnrede] [LieferVorname] [LieferNachname]
```

Anwendung der Funktion

Für die Anwendung der Funktion namens **Adressblock** finden Sie im Modul **mdlTools** der Beispieldatenbank

einige Beispiele. Die Funktion erwartet drei Parameter und liefert direkt den Adressblock als String zurück. Schauen wir uns den folgenden Aufruf an:

```
Debug.Print Adressblock(Lieferadresse, 1, 1)
```

Dieser liefert die Lieferadresse für die Konfiguration mit dem Wert **1** im Feld **KonfigurationID** und für den Kunden mit dem Wert **1** im Feld **KundeID**:

André Minhorst Verlag
Herr André Minhorst
Labor
Borkhofer Str. 17
47137 Duisburg

Wenn Sie die Rechnungsadresse für den gleichen Datensatz mit der gleichen Konfiguration ausgeben möchten, verwenden Sie einfach Rechnungsadresse als ersten Parameter:

```
Debug.Print Adressblock(Rechnungsadresse, 1, 1)
```

Das Ergebnis sieht dann so aus:

Frau
Anja Minhorst
Schnüranstr. 25
47138 Duisburg

Auf die gleiche Weise können Sie durch Variation des zweiten und dritten Parameters andere Konfigurationen und natürlich auch andere Einträge der Kundentabelle zur Ausgabe der Adressblöcke verwenden.

Bei den beiden vorangegangenen Beispielen war als Land jeweils **Deutschland** angegeben. Wie

weiter oben festgelegt, soll der Wert **Deutschland** für die beiden Felder **LieferLand** und **RechnungLand** ignoriert werden, was hier geschieht.

Wenn Sie beispielsweise den vierten Eintrag der Beispieltabelle **tblKunden**, der das Land **Schweiz** verwendet, als Adressblock ausgeben wollen, sieht der Aufruf so aus:

```
Debug.Print Adressblock(Lieferadresse, 1, 4)
```

Dies ist der resultierende Adressblock:

Heinz Müller AG
Herr Heinz Müller
Testweg 2
2222 Luzern
Schweiz

Programmierung der Funktion

Damit können wir uns nun an die Programmierung der Funktion machen, die aus der Tabelle **tblKunden** mithilfe der Konfiguration aus **tblKonfigurationen** einen Adressblock erstellt. Zunächst definieren wir eine Enumeration, welche die beiden Werte **Lieferadresse (0)** und **Rechnungsadresse (1)** festlegt:

```
Public Enum eAdressart
    Lieferadresse = 0
    Rechnungsadresse = 1
End Enum
```

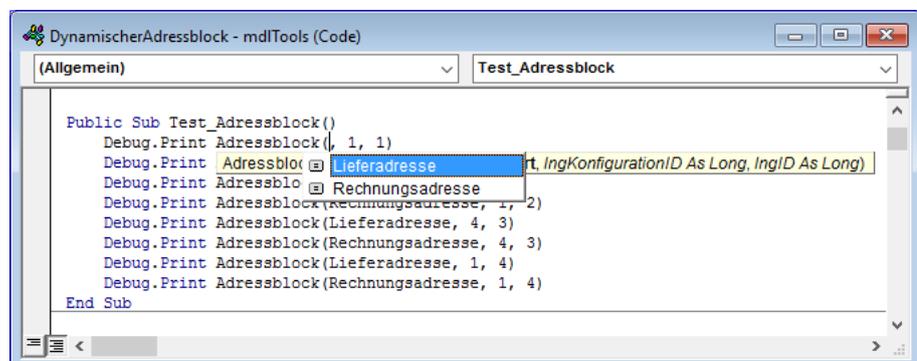


Bild 6: IntelliSense beim Funktionsaufruf

Kundenverwaltung mit Ribbon, Teil I

Im Beitrag »Ribbonklassen« haben wir gezeigt, wie Sie das Ribbon allein mit VBA-Code definieren können. Nun wollen wir diese Technik in einer kleinen Beispielanwendung demonstrieren, in der wir eine Adressenliste in einem Formular anzeigen und alle Befehle zu dieser Adressenliste im Ribbon einbauen – also etwa, um einen neuen Datensatz anzulegen, einen bestehenden Datensatz zu löschen oder die Details zu einem Datensatz anzuzeigen.

Dabei nutzen wir in diesem Fall im Gegensatz zu den meisten anderen Anwendungen den standardmäßig eingestellten Wert **Dokumente im Registerkartenformat** der Option **Dokumentfensteroptionen** in den Access-Optionen. Die Option **Dokumentregisterkarten anzeigen** wollen wir allerdings deaktivieren (s. Bild 1).

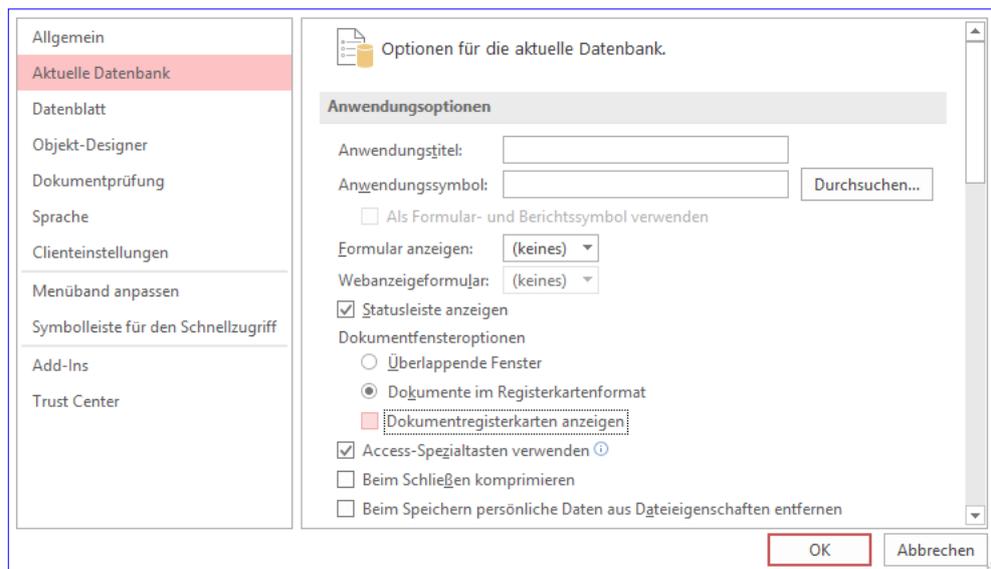


Bild 1: Option zur Anzeige von Formularen im Registerkartenformat

Hauptformular der Anwendung erstellen

Anschließend erstellen wir ein Formular, das alle Datensätze der Tabelle **tblKunden** in der Datenblattansicht anzeigen soll. Normalerweise würden wir eine solche Ansicht als Unterformular in ein Hauptformular integrieren, um noch weitere Steuerelemente für die Verwaltung der Daten in der Datenblattansicht bereitzustellen – etwa zum Anlegen neuer Datensätze, zum Löschen eines bestehenden Datensatzes oder auch um einen Datensatz zum Bearbeiten zu öffnen. Wenn wir allerdings einmal alle Befehle für den Zugriff auf die Adressdaten in das Ribbon verschieben, brauchen wir ja kein übergeordnetes Formular mehr.

Also legen Sie ein neues Formular an und speichern es unter dem Namen **frmKundenebersicht**. Diesem Formular weisen Sie als Datenherkunft die Abfrage **qryKun-**

den zu, die alle Felder der Tabelle **tblKunden** enthält und diese alphabetisch nach dem Nachnamen der Lieferanschrift sortiert (s. Bild 2).

Registerkartenformat und Screenshots

Während ich den Screenshot dieser Abfrage machen möchte, stelle ich übrigens fest, warum ich persönlich nie mit der Option **Dokumente im Registerkartenformat** arbeite: Damit lassen sich einfach keine vernünftigen Screenshots einzelner Access-Objekte machen. Eine Abfrage etwa wird komplett über das Access-Fenster verteilt. Ich müsste nun das Access-Fenster zusammenschrumpfen, um alles in adäquater Größe knipsen zu können und müsste den Screenshot dann noch manuell beschneiden, weil der Abfrageentwurf in diesem Fall kein eigenes Fenster ist, das man mit gängigen Screenshot-

Tools leicht erfassen könnte. Also werde ich während der Erstellung dieses Beitrags wohl oder übel zwischen den einzelnen Ansichten wechseln müssen ...

Hauptformular anlegen

Die soeben erstellte Abfrage **qryKunden** legen wir als Datenherkunft für ein neues Formular namens **frmKundeneuebersicht** fest. Danach ziehen Sie alle Felder der Datenherkunft aus der Feldliste in den Detailbereich des Formularentwurfs (s. Bild 3). Damit die Daten in der Datenblattansicht angezeigt werden, legen Sie für die Eigenschaft **Standardansicht** den Wert **Datenblatt** fest.

Ein Wechsel in die Formularansicht zeigt das Formular mit den gewünschten Daten, das wir nun um einige Funktionen anreichern wollen – und zwar, indem wir diese in das Ribbon der Anwendung aufnehmen.

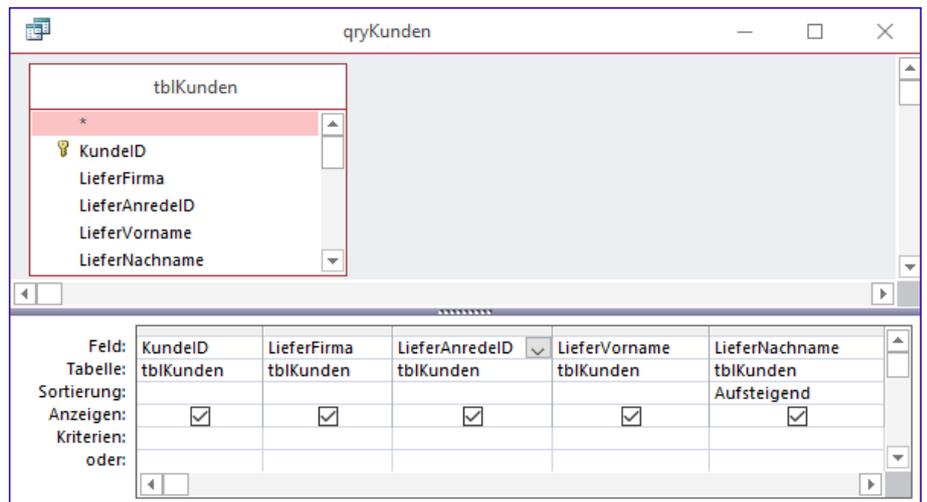


Bild 2: Datenherkunft für das Hauptformular

Um alle Features des Ribbons zu erläutern, wollen wir die Kundenübersicht allerdings nicht direkt beim Öffnen der Anwendung einblenden, sondern zunächst nur einen Ribbon-Eintrag anzeigen, der das Öffnen der Kundenübersicht erlaubt. Erst beim Öffnen dieser Übersicht sollen dann die für dieses Formular spezifischen Befehle im Ribbon angezeigt werden. Wie dies gelingt, erfahren Sie in den folgenden Abschnitten.

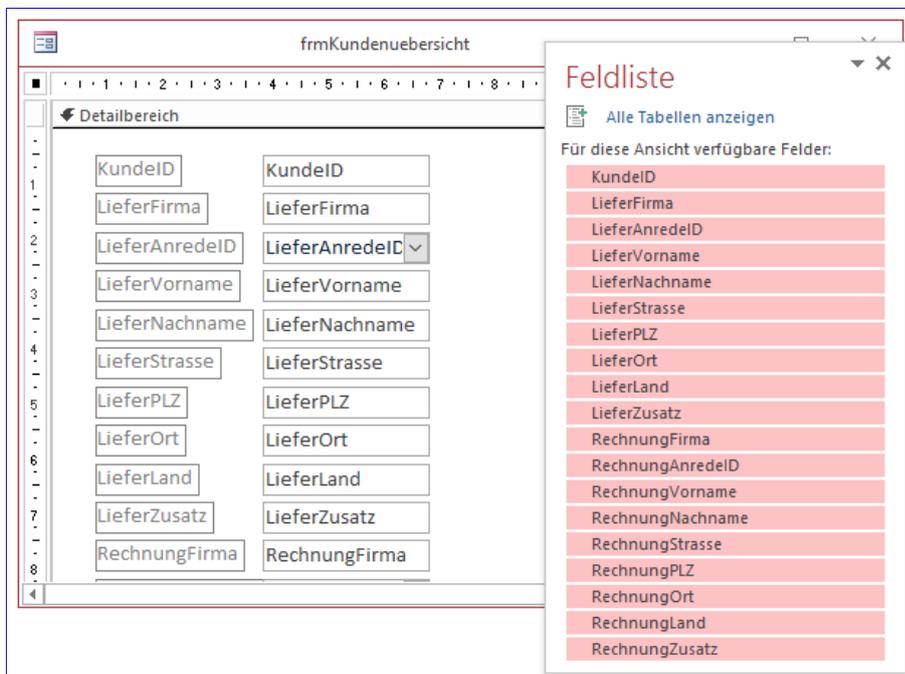


Bild 3: Entwurf des Hauptformulars **frmKundeneuebersicht**

Hinweis: Wenn Sie diese Anwendung als reine Adressverwaltung nutzen wollen, die immer die Kundenübersicht anzeigen soll, können Sie das Formular **frm-Kundeneuebersicht** natürlich auch direkt als Startformular angeben. Dann wird das von uns für dieses Formular festgelegte Ribbon automatisch mit eingeblendet.

Vorbereitungen für das Anlegen des Ribbons

Wenn Sie unsere Ribbon-Klassen nutzen wollen, um das Ribbon zu programmieren, müssen Sie zuvor einige Access-Objekte aus der Beispieldatenbank in Ihre Zieldaten-

bank importieren. Dabei handelt es sich um die folgenden Objekte (bei allen Objekten außer dem Makro **AutoExec** handelt es sich um Standard- oder Klassenmodule):

- **Startribbon**
- **Ribbons**
- **clsButton**
- **clsControls**
- **clsGroup**
- **clsGroups**
- **clsRibbon**
- **clsSeparator**
- **clsTabs**
- **clsTab**
- **mdlRibbonEnums**
- **mdlRibbons**
- **mdlTools**
- **mdlZwischenablage**
- Makro **AutoExec**

Diese Objekte fügen Sie beispielsweise zur Zieldatenbank hinzu, indem Sie beide Datenbanken öffnen und nebeneinander platzieren und dann die Objekte von einem Navigationsbereich zum anderen ziehen.

Bevor Sie das Ribbon programmieren können, benötigen Sie auch noch einen Verweis auf die Bibliothek **Microsoft**

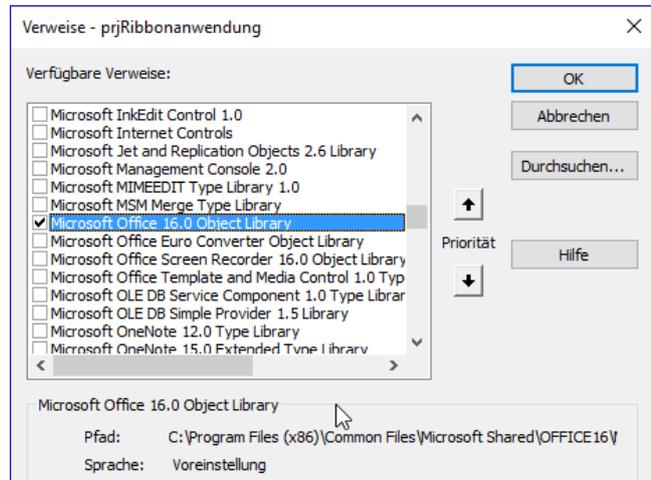


Bild 4: Verweis auf die Office-Bibliothek

Office x.0 Object Library (s. Bild 4). Diesen fügen Sie über den **Verweise**-Dialog hinzu (VBA-Editor, Menüeintrag **Extras\Verweise**).

Den Inhalt des Moduls **StartRibbon** ersetzen Sie nun durch die Codezeilen aus Listing 1. Wir wollen im Startribbon zwei Schaltflächen anzeigen: Eines zum Öffnen der Kundenübersicht und eine zum Beenden der Anwendung. Für diese Schaltflächen müssen wir Variablen des Typs **clsButton** deklarieren und mit dem Schlüsselwort **WithEvents** ausstatten, da wir für diese ja Ereignisprozeduren hinterlegen wollen, die beim Anklicken der **Button**-Elemente ausgelöst werden.

Die Prozedur **CreateRibbon** übernimmt die Erstellung des Ribbons. Sie legt zunächst ein neues Objekt des Typs **clsRibbon** an. Dazu fügt sie mit der **Add**-Methode ein neues Element zur Auflistung **Ribbons** hinzu. Für dieses stellen wir nun die Eigenschaft **StartFromScratch** auf den Wert **True** ein. Auf diese Weise werden alle eingebauten Ribbon-Elemente ausgeblendet. Dann fügen wir über die **Add**-Methode der **Tabs**-Auflistung von **objRibbon** ein neues **Tab**-Element hinzu und legen mit der Eigenschaft **Label** die Beschriftung **Kundenverwaltung** fest.

Nun legen wir auf ähnliche Weise ein **Group**-Element an und versehen es mit der Überschrift **Übersicht**. Dieses

```
Dim WithEvents btnKundenebersicht As clsButton
Dim WithEvents btnBeenden As clsButton

Public Sub CreateRibbon()
    Dim objRibbon As clsRibbon
    Dim objTab As clsTab
    Dim objGroup As clsGroup
    Set objRibbon = Ribbons.Add("Main")
    With objRibbon
        .StartFromScratch = True
        Set objTab = .Tabs.Add("tabKundenverwaltung")
        With objTab
            .Label = "Kundenverwaltung"
            Set objGroup = .Groups.Add("grpUebersicht")
            With objGroup
                .Label = "Übersicht"
                Set btnKundenebersicht = .Controls.Add(msoRibbonControlButton, "btnKundenebersicht")
                With btnKundenebersicht
                    .Label = "Kundenübersicht"
                    .Size = msoRibbonControlSizeLarge
                    .Image = "users3"
                End With
                .Controls.Add msoRibbonControlSeparator
                Set btnBeenden = .Controls.Add(msoRibbonControlButton, "btnBeenden")
                With btnBeenden
                    .Label = "Beenden"
                    .Size = msoRibbonControlSizeLarge
                    .Image = "close"
                End With
            End With
        End With
    End With
    Ribbons.CreateStartRibbon "Main"
End Sub

Public Property Get Beenden() As clsButton
    Set Beenden = btnBeenden
End Property

Public Property Get Kundenebersicht() As clsButton
    Set Kundenebersicht = btnKundenebersicht
End Property

Private Sub btnBeenden_OnAction(control As Office.IRibbonControl)
    DoCmd.Quit
End Sub

Private Sub btnKundenebersicht_OnAction(control As Office.IRibbonControl)
    DoCmd.OpenForm "frmKundenebersicht"
End Sub
```

Listing 1: Code für das Startribbon