

ACCESS

IM UNTERNEHMEN

STÜCKLISTEN, TEIL II

Erweitern Sie die Stücklisten-Lösung um komfortable Kontextmenüs (ab S. 46).



In diesem Heft:

TICKETSYSTEM

Beantworten Sie Kundenanfragen mit vorgefertigten und individuell anpassbaren Texten.

SEITE 59

KOMBINATIONSFELDER

Scrollen Sie mit der Maus durch Kombinationsfelder, ohne diese aufzuklappen.

SEITE 21

VERKNÜPFTE DATEN

Schauen Sie sich verknüpfte Daten in einer frei konfigurierbaren Übersicht an.

SEITE 6

TABELLEN UND DATENMODELLIERUNG	Datensatz duplizieren	2
	Verknüpfte Daten suchen	6
FORMULARE UND STEUERELEMENTE	Kombinationsfeld rauf und runter	21
	Schnellsuche mit Verzögerung	25
ERGONOMIE UND BENUTZEROBERFLÄCHE	Ribbon für das Ticketsystem	27
SQL SERVER UND CO.	RDBMS-Zugriff per VBA: Fehlerbehandlung	35
VBA UND PROGRAMMIERTECHNIK	Lösung zum Formular-Add-In umbauen	38
LÖSUNGEN	Stücklisten, Teil II	46
	Ticketverwaltung, Teil IV	59
SERVICE	Impressum	U2
	Editorial	1
DOWNLOAD	Die Downloads zu dieser Ausgabe finden Sie wie gewohnt unter: http://www.access-im-unternehmen.de/download Benutzername: **** Kennwort: **** Die Direktlinks zu den Downloads befinden sich am unteren Rand des jeweiligen Beitrags.	

Impressum

ISBN 978-3-8092-1496-0

ISSN: 1616-5535

Mat.-Nr. 01583-5100

Access im Unternehmen

© Haufe-Lexware GmbH & Co. KG, Munzinger Str. 9, 79111 Freiburg

Kommanditgesellschaft, Sitz Freiburg

Registergericht Freiburg, HRA 4408

Komplementäre: Haufe-Lexware Verwaltungs GmbH, Sitz Freiburg,

Registergericht Freiburg, HRB 5557; Martin Laqua

Geschäftsführung: Isabel Blank, Markus Dränert, Jörg Frey, Birte Hackenjös, Randolf Jessl, Markus Reithwiesner, Joachim Rotzinger, Dr. Carsten Thies

Beiratsvorsitzende: Andrea Haufe

Steuernummer: 06392/11008

Umsatzsteuer-Identifikationsnummer: DE 812398835

Redaktion: Dipl.-Inform. (FH) Michael Forster, (Chefredakteur, V.i.S.d.P.), Dipl.-Ing. André Minhorst (Chefredaktion, extern),

Sabine Wißler (Redaktionsassistentin), Rita Klingenstein (sprachl. Lektorat), Sascha Trowitzsch (Fachlektorat)

Autor: Dipl.-Ing. André Minhorst, Sascha Trowitzsch

Anschrift der Redaktion:

Postfach 100121, 79120 Freiburg, Tel. 0761/898-0, Fax: 0761/898-3990,

E-Mail: computer@haufe.de, Internet: <http://computer.haufe.de>

Druck: Schätzl Druck & Medien GmbH & Co. KG, Donauwörth

Auslieferung und Vertretung für die Schweiz: H.R. Balmer AG, Neugasse 12, CH-6301 Zug, Tel. 041/711 4735, Fax: 041/711 0917

Das Update-Heft und alle darin enthaltenen Beiträge und Abbildungen sind urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmung und für die Einspeicherung in elektronische Systeme.

Wir weisen darauf hin, dass die verwendeten Bezeichnungen und Markennamen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen. Die im Werk gemachten Angaben erfolgen nach bestem Wissen, jedoch ohne Gewähr. Für mögliche Schäden, die im Zusammenhang mit den Angaben im Werk stehen könnten, wird keine Gewährleistung übernommen.

Access im Unternehmen wird 100!

Es ist soweit: Access im Unternehmen feiert die 100. Ausgabe. Seit Ende 2000 erscheint das Magazin alle zwei Monate und liefert auf mehr als 72 Seiten frisches Know-how rund um die Datenbank Access und verwandte Themen wie die Office-Programmierung, XML, alternative Backend-Datenbanken wie SQL Server oder MySQL oder den Zugriff auf Webservices zum Datenaustausch.



Mit der Ausgabe 1/2000, die erst Ende des Jahres erschien und gleichzeitig auch die einzige Ausgabe dieses Jahres war, hat allerdings nicht alles begonnen: Grundstein für dieses Magazin war das **Praxishandbuch Access** (erst für Access 97, später dann für Access 2000), ein Loseblattwerk, das alle zwei Monate neue Themen zum Einheften in einen stetig wachsenden Ordner lieferte. Wer nun nachzählt, ob nun wirklich schon 100 Ausgaben veröffentlicht wurden und nur 99 zählt, ist noch nicht lange genug als Leser dabei: Im Jahr 2001 gab es nämlich eine siebte Ausgabe.

Das Praxishandbuch wurde also im Jahr 2000 vom neuen Magazin **Access im Unternehmen** abgelöst, das zunächst noch jeweils mit einer Heft-CD mit den Programmierbeispielen zu den Artikeln geliefert wurde. Erst im Jahr 2003 wurden die Artikel parallel auch auf der Webseite **www.access-im-unternehmen.de** angeboten, allerdings nur für registrierte Leser des Magazins. Nach einer Eingewöhnungszeit wurde dann die Heft-CD als Beilage abgeschafft, da die Bereitstellung von Beispieldatenbanken online sinnvoller ist. Seit der Ausgabe 1/2010 finden Leser nicht nur die Artikel im HTML-Format, sondern auch das komplette Magazin als PDF-Version im Downloadbereich (**www.access-im-unternehmen.de/download**).

Eine erstaunliche Entwicklung, wenn man sich überlegt, dass wir vor vielen Jahren zu Zeiten des C64 noch komplette Listings aus Zeitschriften abgetippt haben, um ein Spiel oder ein Programm nutzen zu können – zumal diese teilweise als Maschinencode abgedruckt wurden und es somit kaum eine Möglichkeit gab, etwas aus dem Code zu lernen!

Seit dem Erscheinen der ersten Ausgabe haben insgesamt über 20 Autoren aus der Fachwelt Themen zu Access im Unternehmen beigezeichnet – einige nur mit sehr wenigen Beiträgen, andere regelmäßig. Dabei sind 870 Beiträge auf über 7.000 Seiten zustande gekommen. Die meisten dürften sich um das Thema Formulare gedreht haben, denn die Benutzeroberfläche ist nun einmal die Schnittstelle zwischen dem Benutzer und der Datenbank.

Thematisch haben wir kaum einen Bereich ausgelassen: Wir haben gezeigt, wie Datenmodelle erstellt werden, wie Sie Abfragen zur Ermittlung der gewünschten Daten formulieren, Formulare und Steuerelemente mit allerlei Finessen zur Darstellung und Bearbeitung der Daten geliefert und die Möglichkeit geboten, die Daten per Bericht zu Papier zu bringen. Wir haben den Austausch von Daten mit allen wichtigen Office-Anwendungen wie Excel, Word und Outlook beleuchtet, die Migration zu alternativen Backends wie SQL Server erläutert und gezeigt, wie Sie allerlei Internet-Services nutzen – zum Beispiel Google Earth, Paypal, Ebay, Amazon oder Facebook.

Genau so wollen wir weitermachen und freuen uns, wenn Access uns nach den wenigen Innovationen von Access 2010 bis Access 2016 in der nächsten Version auch wieder einmal neue Features liefert, die wir hier im Detail für Sie vorstellen können.

In diesem Sinne – auf die nächsten 100!

Ihr Michael Forster

Datensatz duplizieren

Manchmal möchten Sie einen neuen Datensatz auf Basis eines bereits vorhandenen Datensatzes anlegen – beispielsweise um diesen anschließend anzupassen. Dazu gibt es unter Access verschiedene Möglichkeiten – per Benutzeroberfläche oder auch per VBA. Wir schauen uns die Varianten an.

Beispieldatenbank

Für die Beispiele haben wir eine kleine Beispieldatenbank mit den Tabellen der Süd Sturm-Datenbank erstellt. Diese enthält ein Haupt- und ein Unterformular, wobei das Unterformular die Datensätze der Tabelle **tblArtikel** in der Datenblattansicht anzeigt. Damit und mit verschiedenen Schaltflächen im Hauptformular wollen wir die Möglichkeiten zum Duplizieren vorstellen (s. Bild 1).

Datensatz von Hand in der Datenblattansicht kopieren

Die einfachste Variante, einen Datensatz zu duplizieren, erfordert ein paar Mausklicks. Dazu markieren Sie den zu duplizierenden Datensatz per Mausklick auf den grauen Kasten links vom Datensatz, kopieren diesen per **Strg + C** oder Kontextmenübefehl, klicken dann auf den gleichen Bereich in der Zeile für einen neuen, leeren Datensatz und

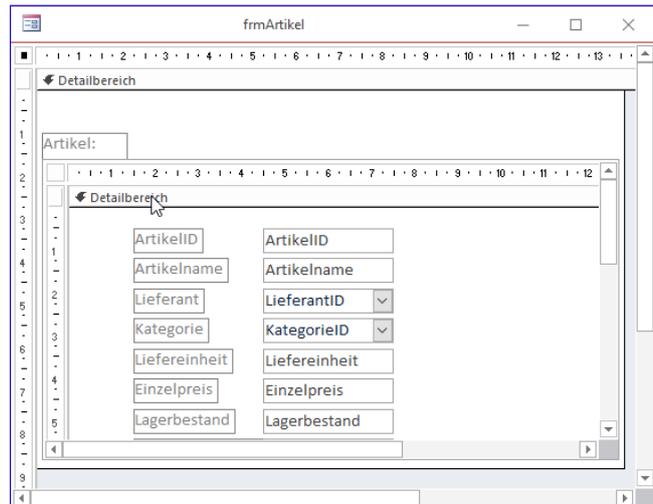


Bild 1: Beispielformular dieses Beitrags

fügen den zu duplizierenden Datensatz dann per **Strg + V** oder den entsprechenden Kontextmenübefehl an dieser Stelle ein (s. Bild 2).

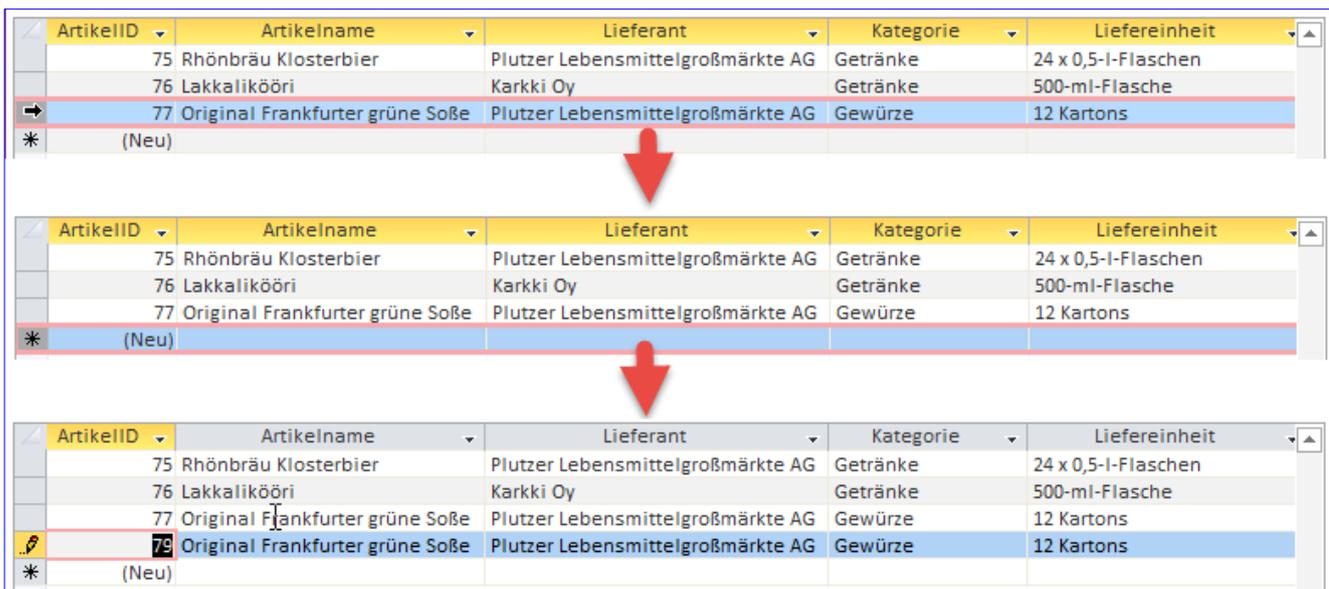


Bild 2: Datensatz von Hand in der Datenblattansicht kopieren

Diese Variante funktioniert einwandfrei (solange keine eindeutig indizierte Felder enthalten sind). Das einzige eindeutig indizierte Feld ist das Primärschlüsselfeld, und dieses wird bei dieser Methode ganz einfach durch die Autowert-Funktion mit einem neuen Wert belegt.

Duplizieren mit Tastaturbefehlen

Die Aktionen der Benutzeroberfläche lassen sich in der Regel durch VBA-Befehle der Klasse **DoCmd** abbilden – oder durch den Aufruf der **RunCommand**-Methode mit einem ihrer zahlreichen Parameter. Die **RunCommand**-Methode ist dabei wiederum eine Methode der **DoCmd**-Klasse, aber auch des **Application**-Objekts. Sie können diese Anweisung daher auch ohne Voranstellen des entsprechenden Objekts einsetzen. Also legen wir eine Schaltfläche im Hauptformular an, welche den aktuell markierten Datensatz im Unterformular duplizieren soll. Den kompletten Ablauf steuern wir fast ausschließlich mit der **RunCommand**-Anweisung:

```
Private Sub cmdDuplizierenMitDoCmd_Click()
    Me!sfmArtikel.SetFocus
    RunCommand acCmdSelectRecord
    RunCommand acCmdCopy
    RunCommand acCmdRecordsGoToNew
    RunCommand acCmdSelectRecord
    RunCommand acCmdPaste
End Sub
```

Die einzige Ausnahme ist das Setzen des Fokus auf das Unterformular, denn sonst würde beispielsweise die Anweisung **RunCommand acCmdSelectRecord** auf das aktuelle Formular angewendet werden – und da wir soeben auf eine Schaltfläche im Hauptformular geklickt haben, wäre dies das Hauptformular.

Nach dem Verschieben des Fokus auf das Unterformular rufen wir nacheinander mehrmals die **RunCommand**-Anweisung auf und bilden damit genau die Schritte ab, die wir zuvor von Hand durchgeführt haben: Die Konstante **acCmdSelectRecord** markiert den kompletten Datensatz, **acCmdCopy** kopiert diesen, **acCmdRecordsGoToNew** verschiebt den Datensatzzeiger auf den leeren, neuen Datensatz, **acCmdSelectRecord** markiert diesen und **acCmdPaste** fügt den Datensatz aus der Zwischenablage dort ein. Die Lösung funktioniert grundsätzlich, aber vielleicht möchte Sie etwas mehr Kontrolle, indem Sie den Artikel unabhängig von der aktuell angezeigten Benutzeroberfläche duplizieren.

Datensatz kopieren ohne Formular

Wenn Sie den Datensatz per VBA kopieren wollen, können Sie theoretisch also beispielsweise eine **INSERT INTO...SELECT**-Anweisung verwenden (s. Listing 1). Zu Beispielzwecken integrieren wir diese wieder in das Formular, so kann der Benutzer den zu kopierenden Datensatz auswählen und der Datensatzzeiger soll anschließend direkt

```
Private Sub cmdDuplizierenMitSelectInto_Click()
    Dim db As DAO.Database
    Dim lngArtikelID As Long
    Set db = CurrentDb
    lngArtikelID = Nz(Me!sfmArtikel.Form!ArtikelID, 0)
    If Not lngArtikelID = 0 Then
        db.Execute "INSERT INTO tblArtikel SELECT * FROM tblArtikel WHERE ArtikelID = " & lngArtikelID, dbFailOnError
        lngArtikelID = db.OpenRecordset("SELECT @@IDENTITY").Fields(0)
        Me!sfmArtikel.Form.Requery
        Me!sfmArtikel.Form.Recordset.FindFirst "ArtikelID = " & lngArtikelID
    End If
End Sub
```

Listing 1: Duplizieren des Datensatzes mit **INSERT INTO...SELECT**

Verknüpfte Daten suchen

Wenn Sie eine Anwendung programmieren, die ihre Daten nur über die Benutzeroberfläche ändert, sehen Sie die Änderungen ja immer direkt in den entsprechenden Formularen und Steuerelementen. Aber manchmal erstellen, ändern oder löschen Sie Daten auch per Mausklick auf eine Schaltfläche und die dahinter liegende Prozedur erledigt den Rest. Gerade in der Entwicklungsphase möchten Sie dann natürlich schnell sehen, ob die Daten auch wirklich wie gewünscht editiert wurden. Dazu quält man sich dann meist durch die einzelnen Tabellen und prüft das Ergebnis. Wenn sich die Daten über verknüpfte Tabellen erstrecken, wird dies erst recht anstrengend. Aber nicht mit der Lösung, die wir in diesem Beitrag vorstellen!

Beispieldatenbank

Als Beispiel dient uns die oft verwendete Süd Sturm-Datenbank. Sie hat einige verknüpfte Tabellen, mit denen wir das eingangs erwähnte Problem des schwierigen Nachvollziehens geänderter oder neuer Daten leicht nachstellen können. Im ersten Beispiel wollen wir uns die verknüpften Tabellen in der Reihenfolge **tblKunden**, **tblBestellungen**, **tblBestelldetails**, **tblArtikel** ansehen (s. Bild 1).

Die fertige Lösung

Am Ende dieses Artikels sollen Sie eine Lösung haben, mit der Sie verknüpfte Tabellen mit einer Tiefe von vier Hierarchieebenen auswählen und analysieren können.

Dazu erhalten Sie ein Formular, das vier Unterformulare enthält, von denen jedes die Daten einer von bis zu vier Tabellen anzeigt (s. Bild 2). Das oberste Unterformular soll beispielsweise die Daten der Tabelle **tblKunden** anzeigen.

Darunter finden Sie dann die Daten der Tabelle **tblBestellungen**. Wenn der Benutzer nun auf einen der Datensätze der Tabelle **tblKunden** klickt, soll das Unterformular mit den Daten der Tabelle **tblBestellungen** nur die Datensätze anzeigen, die zum aktuellen Kunden gehören.

Darunter geht es weiter: Das dritte Unterformular zeigt alle Daten der Tabelle **tblBestelldetails** an, die mit dem Datensatz des zweiten Unterformulars mit den Bestellungen verknüpft ist. Das vierte Unterformular liefert schließlich noch den dem Bestelldetail zugeordneten Datensatz der Tabelle **tblArtikel**.

Der Clou bei der Lösung ist, dass Sie die Tabellen, deren Daten in den Unterformularen angezeigt werden, nicht fest in das Formular programmieren müssen, sondern dass es für alle beteiligten Tabellen, Primärschlüsselfelder und Fremdschlüsselfelder Kombinationsfelder zur Auswahl gibt. Auf diese Weise können Sie mit dem Formular nicht

nur verschiedene Konfigurationen von verknüpften Tabellen untersuchen, sondern diese auch noch zur Laufzeit neu einstellen, um eine andere Konfiguration zu begutachten.

Nun wäre es natürlich aufwendig, immer wieder

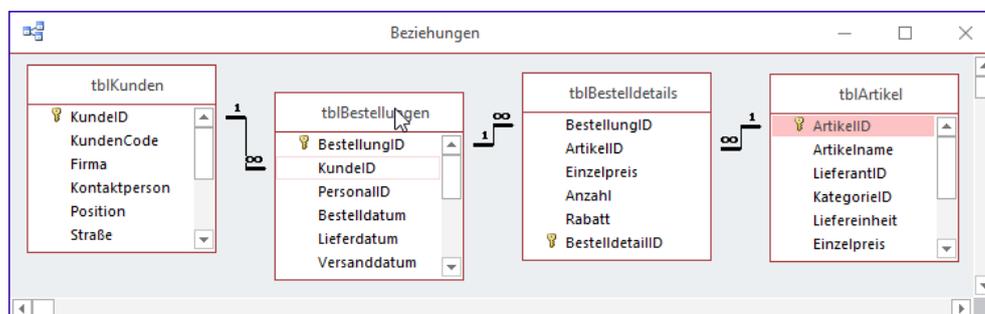


Bild 1: Tabellen der Datenbank Süd Sturm

Konfiguration: Südsturm/Kunden und Bestellungen

Datenherkunft: tblKunden **Primärschlüssel:** KundeID

KundeID	KundenCode	Firma	Kontaktperso	Position	Straße
1	ALFKI	Alfreds Futterk	Maria Anders	Vertriebsmitarb	Obere Str. 57
2	ANATR	Ana Trujillo Emp	Ana Trujillo	Inhaberin	Avda. de la Cons M
3	ANTON	Antonio Moreno	Antonio Moreno	Inhaber	Mataderos 2312 M
4	AROUT	Around the Horn	Thomas Hardy	Vertriebsmitarb	120 Hanover Sq. Lo
5	BERGS	Berglunds snabt	Christina Berglun	Einkaufsleitung	Berguvsvägen 8 Lu
6	BLAUS	Blauer See Delik	Hanna Moos	Vertriebsmitarb	Forsterstr. 57 M
7	BLONP	Blondel père et	Frédérique Citea	Marketingmana	24, place Kléber Str

Datenherkunft: tblBestellungen **Primärschlüssel:** BestellungID **Fremdschlüssel:** KundeID

BestellungID	KundeID	PersonaID	Bestelldatum	Lieferdatum	Versanddatum	Versa
10643	1	6	25.08.1997	22.09.1997	02.09.1997	1
10692	1	4	03.10.1997	31.10.1997	13.10.1997	2
10702	1	4	13.10.1997	24.11.1997	21.10.1997	1
10835	1	1	15.01.1998	12.02.1998	21.01.1998	3
10952	1	1	16.03.1998	27.04.1998	24.03.1998	1
11011	1	3	09.04.1998	07.05.1998	13.04.1998	1

Datenherkunft: tblBestelldetails **Primärschlüssel:** ArtikelID **Fremdschlüssel:** BestellungID

BestellungID	ArtikelID	Einzelpreis	Anzahl	Rabatt
10643	28	22,80 €	15	0,25
10643	39	9,00 €	21	0,25
10643	46	6,00 €	2	0,25

Datenherkunft: tblArtikel **Fremdschlüssel:** ArtikelID

ArtikelID	Artikelname	LieferantID	KategorieID	Liefereinheit	Einzelpreis	Lager
28	Rössle Sauerkra	12	7	25 x 825-g-Dosen	22,80 €	26

Bild 2: Die fertige Lösung

alle Eigenschaften einer Konfiguration neu einzugeben, wenn man diese einmal wechseln will oder das Formular gar in einer anderen Datenbank als der aktuellen einsetzen möchte. Deshalb können Sie verschiedene Konfigurationen speichern und diese immer wieder neu abrufen. Wenn Sie das Formular erstmalig öffnen, fragt dieses Sie

gleich nach dem Namen der ersten anzulegenden Konfiguration. Diese wird dann nach dem Öffnen des Formulars oben im Kombinationsfeld angezeigt.

Sie können nun mit den je nach Ebene zwei bis drei Kombinationsfeldern die Tabelle auswählen, die das entsprechende Unterformular anzeigen soll. Danach legen Sie zunächst für die obere Tabelle das Primärschlüsselfeld fest.

Damit das Formular weiß, welche Datensätze des zweiten Unterformulars zu dem im ersten Unterformular angezeigten Datensatz gehören, geben Sie für das zweite Unterformular das Fremdschlüsselfeld an, über das es mit den Datensätzen der übergeordneten Tabelle verknüpft ist.

Auf die gleiche Weise verfahren Sie mit den folgenden Kombinationsfeldern für Primär- und Fremdschlüsselfelder, bis Sie für

das vierte Unterformular nur noch das Fremdschlüsselfeld angeben müssen. Wenn Sie bereits mehrere Konfigurationen angelegt haben, können Sie diese über das Kombinationsfeld mit der Beschriftung **Konfiguration** wechseln. Mit der **Löschen-**

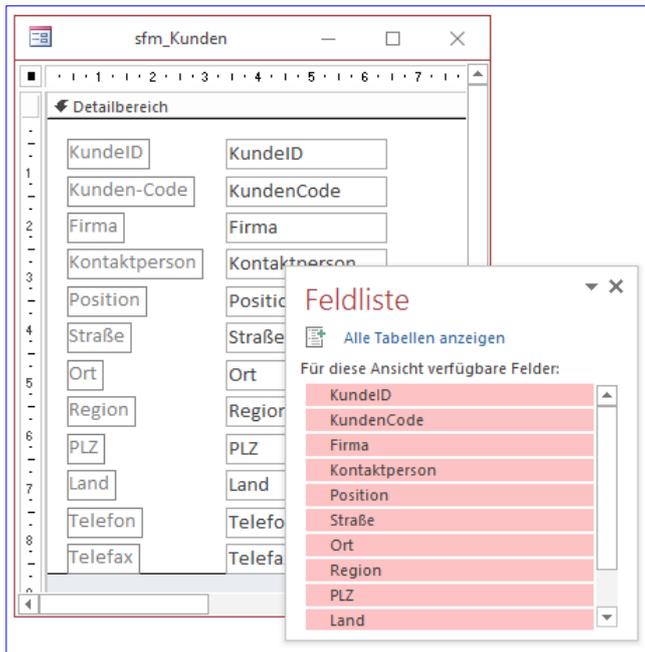


Bild 3: Entwurf des Unterformulars für die Anzeige der Kundendatensätze

Schaltfläche entfernen Sie die aktuell ausgewählte Konfiguration aus der Datenbank.

Auf geht's!

Auf den folgenden Seiten werden wir uns zunächst eine einfache, statische Variante dieser Lösung ansehen, um die Grundlagen der Vorgehensweise zu verdeutlichen. Im Anschluss schauen wir uns dann an, wie das Formular zur flexiblen Anzeige der Daten verknüpfter Tabellen programmiert wird.

Erstellen des Hauptformulars

Das Hauptformular dient zunächst nur als Container für die einzelnen Unterformulare. Erstellen Sie also ein leeres Hauptformular und speichern Sie es unter dem Namen **frmVerknuepfteTabellen**.

Gleich fügen wir die einzelnen Unterformulare und die Funktion in Form von VBA-Code hinzu.

Erstellen der Unterformulare

Die Unterformulare sind prinzipiell alle gleich aufgebaut. Wir schauen uns dies am Beispiel des Formulars **sfmKunden** an, welches die Datensätze der Tabelle **tblKunden** abbilden soll. Dazu legen Sie ein neues, leeres Formular an und stellen als Datenherkunft die Tabelle **tblKunden** ein. Gegebenenfalls können Sie auch eine Abfrage erstellen, welche beispielsweise die Kunden nach dem Feld **Firma** sortiert. Danach fügen Sie wie in Bild 3 alle Felder der Datenherkunft zum Detailbereich der Entwurfsansicht der Tabelle hinzu. Stellen Sie die Eigenschaft **Standardansicht auf Datenblatt** ein. Speichern Sie das Formular dann etwa unter dem Namen **sfmKunden** und schließen Sie es.

Unterformular in Hauptformular

Dann öffnen Sie das Hauptformular in der Entwurfsansicht und ziehen das Unterformular **sfmKunden** aus dem Navigationsbereich in den Detailbereich des Hauptformulars. Das Ergebnis sieht etwa wie in Bild 4 aus.

Weitere Unterformulare

Auf die gleiche Weise gehen Sie mit den Unterformularen für die Tabellen **tblBestellungen**, **tblBestelldetails** und **tblArtikel** vor: Legen Sie diese an, fügen Sie Datenher-

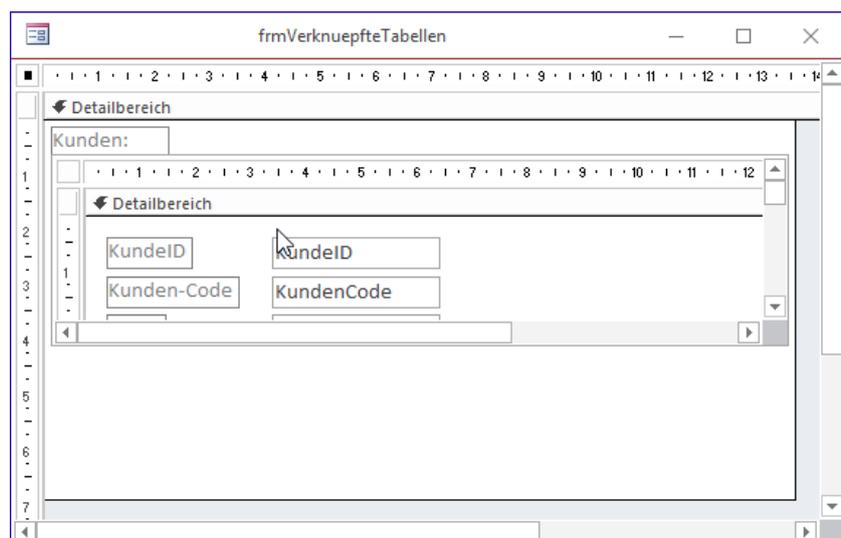


Bild 4: Das Hauptformular mit einem ersten Unterformular

kunft und Felder hinzu und ziehen Sie diese dann untereinander in das Formular **frmVerknuepfteTabellen**. Mit allen vier Unterformularen sieht das Formular dann in der Datenblattansicht wie in Bild 5 aus. Die vier Unterformulare zeigen hier jeweils alle Datensätze ihrer Datenquelle an.

Das ist natürlich schon praktischer als wenn man immer alle Tabellen einzeln öffnen muss. Allerdings wollen wir noch einen entscheidenden Schritt weitergehen: Wenn der Benutzer nun auf einen der Einträge der Tabelle **tblKunden** im Unterformular **sfmKunden** klickt, soll das nächste Unterformular nur noch die Datensätze der Tabelle **tblBestellungen** anzeigen, die über das Feld **KundeID** mit dem ausgewählten Kundendatensatz verknüpft sind.

Funktion hinzufügen

Dazu müssen wir ein paar Zeilen VBA-Code hinzufügen. Wir machen das erstmal auf die einfachste Weise. Das Ereignis, das wir benötigen, heißt **Beim Anzeigen** und wird immer dann ausgelöst, wenn ein neuer Datensatz in einem Formular den Fokus erhält.

Diesem weisen wir nun eine Ereignisprozedur zu, welche die Datensätze des untergeordneten Unterformulars so filtert, dass es nur noch die mit dem aktuell ausgewählten Datensatz verknüpften Datensätze anzeigt.

Der Plan ist also, eine Prozedur wie die folgende auszulösen, die eine Referenz auf das darunter angeordnete Formular speichert und für diese die **Filter**-Eigenschaft mit einem entsprechenden Filter-Ausdruck belegt:

```
Private Sub Form_Current()  
    Dim frm As Form
```

```
Set frm = Me.Parent!sfmBestellungen.Form  
frm.Filter = "KundeID = " & Me!KundeID  
frm.FilterOn = True  
End Sub
```

End Sub

Wenn wir das Formular nun öffnen, erhalten wir allerdings die Fehlermeldung aus Bild 6. Das ist etwas überraschend: Wir haben das Unterformular doch korrekt referenziert? Nach dem Akzeptieren des Fehlers mit Beenden schauen wir uns an, ob wir das Unterformular über den Direktbereich referenzieren können. Dazu setzen wir den folgenden Befehl ab:

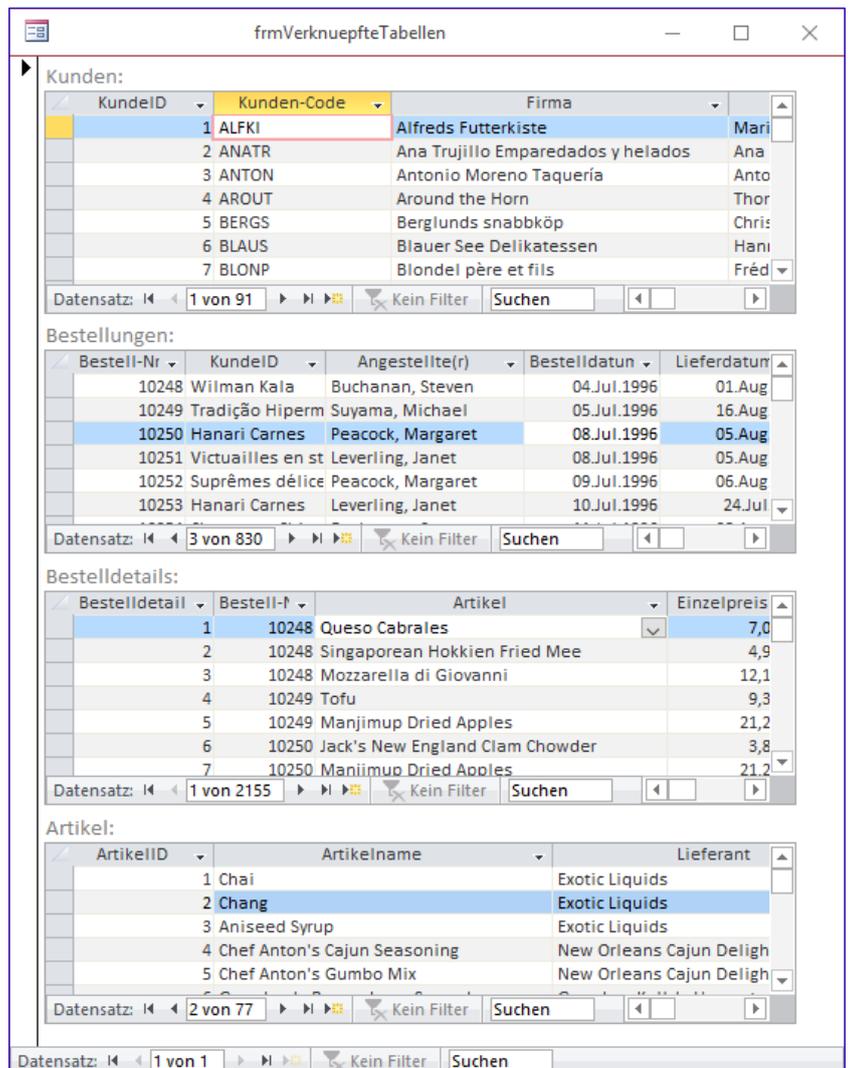


Bild 5: Das Hauptformular mit allen vier Unterformularen

```
? Forms!frmVerknuepfteTabellen!z  
    sfmBestellungen.Form.Name  
    sfmBestellungen
```

Vom Direktfenster aus können wir also auf das Unterformular zugreifen. Also scheint es eine Frage des Timings zu sein. Das Ereignis **Beim Anzeigen** wird ja nicht nur dann ausgelöst, wenn der Benutzer den Datensatz wechselt, sondern auch direkt nach dem Öffnen des Formulars. Offensichtlich ist das Unterformular **sfmBestellungen** also noch nicht geladen, wenn das Ereignis **Beim Anzeigen** des Unterformulars **sfmKunden** ausgelöst wird.

Wie können wir dieses Verhalten beeinflussen? In welcher Reihenfolge werden die Unterformulare überhaupt geladen, wenn sich mehrere Unterformulare im gleichen Hauptformular befinden? Dies wollen wir experimentell prüfen, indem wir ein neues, leeres Hauptformular erstellen und diesmal zuerst das Unterformular **sfmBestellungen** in den Entwurf des Formulars ziehen und erst dann **sfmKunden**.

Den Code brauchen Sie nicht erneut einzugeben, da er sich ja in dem Unterformular **sfmKunden** befindet. Wenn Sie das Formular nun öffnen, erscheint keine Fehlermeldung und das Unterformular **sfmBestellungen** zeigt nur die Bestellungen an, die zum aktuell im Unterformular **sfmKunden** ausgewählten Datensatz gehören.

Verfeinerung des Codes

Wir wollen uns allerdings nicht auf solch willkürliche Faktoren wie die Reihenfolge des Hinzufügens von Steuerelementen verlassen. Außerdem ist es ja auch kein erklärtes Ziel, gleich beim Anzeigen des Formulars alle Daten gleich auf den im ersten Formular ausgewählten Datensatz zu konzentrieren.

Schließlich ist es ohnehin kein guter Programmierstil, ein Unterformular von einem anderen Unterformular über den Umweg des Hauptformulars zu referenzieren. Und wenn wir die hier verwendete Prozedur auch für die verknüpften

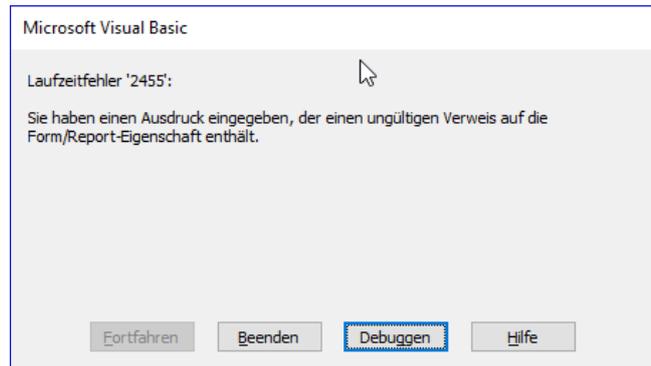


Bild 6: Fehlermeldung beim Öffnen des Formulars

Tabellen in den übrigen Unterformularen nutzen wollen, verteilt sich der Code schnell über mehrere Formulare und wird unübersichtlich.

Daher wollen wir sämtlichen Code gleich ins Hauptformular überführen, von wo wir die Unterformulare referenzieren und die Ereignisprozeduren für das Ereignis **Beim Anzeigen** der verschiedenen Unterformulare implementieren wollen (s. Listing 1). Also löschen wir zunächst die Ereignisprozedur **Form_Current** aus dem Unterformular **sfmKunden**.

Danach legen Sie eine Variable für das Unterformular **sfmKunden** im Klassenmodul des Hauptformulars **frmVerknuepfteTabellen** an:

```
Dim WithEvents frm_sfmKunden As Form
```

Damit diese gefüllt wird, implementieren wir das Ereignis **Beim Laden** des Hauptformulars wie folgt:

```
Private Sub Form_Load()  
    Set frm_sfmKunden = Me!sfmKunden.Form  
    frm_sfmKunden.OnCurrent = "[Event Procedure]"  
End Sub
```

Nun müssen wir nur noch das Ereignis **Beim Anzeigen** des Unterformulars implementieren. Dazu wählen Sie im Klassenmodul des Hauptformulars aus dem linken Kombinationsfeld den Eintrag **frm_sfmKunden** aus und aus dem

```
Dim WithEvents frm_sfmBestellungen As Form
Dim WithEvents frm_sfmKunden As Form
Dim WithEvents frm_sfmBestelldetails As Form
Dim frm_sfmArtikel As Form

Private Sub Form_Load()
    Set frm_sfmKunden = Me!sfmKunden.Form
    frm_sfmKunden.OnCurrent = "[Event Procedure]"
    Set frm_sfmBestellungen = Me!sfmBestellungen.Form
    frm_sfmBestellungen.OnCurrent = "[Event Procedure]"
    Set frm_sfmBestelldetails = Me!sfmBestelldetails.Form
    frm_sfmBestelldetails.OnCurrent = "[Event Procedure]"
    Set frm_sfmArtikel = Me!sfmArtikel.Form
End Sub

Private Sub frm_sfmKunden_Current()
    With frm_sfmBestellungen
        .Filter = "KundeID = " & frm_sfmKunden.KundeID
        .FilterOn = True
    End With
End Sub

Private Sub frm_sfmBestellungen_Current()
    With frm_sfmBestelldetails
        .Filter = "BestellungID = " & frm_sfmBestellungen.BestellungID
        .FilterOn = True
    End With
End Sub

Private Sub frm_sfmBestelldetails_Current()
    With frm_sfmArtikel
        .Filter = "ArtikelID = " & frm_sfmBestelldetails.ArtikelID
        .FilterOn = True
    End With
End Sub
```

Listing 1: Code, um die Unterformulare nacheinander zu filtern

rechten Kombinationsfeld den Eintrag **OnCurrent**. Dies legt die neue Ereignisprozedur an, die Sie wie folgt ergänzen:

```
Private Sub frm_sfmKunden_Current()
    With frm_sfmBestellungen
        .Filter = "KundeID = " & frm_sfmKunden.KundeID
        .FilterOn = True
    End With
End Sub
```

bereits vorgestellten Code-Zeilen.

Wir deklarieren also für alle vier Unterformulare jeweils eine **Form**-Variable, von denen allerdings nur die ersten drei mit dem Schlüsselwort **WithEvents** ausgestattet werden müssen – das letzte Unterformular mit den Einträgen der Tabelle **tblArtikel** soll ja beim Anklicken keine weiteren Datensätze mehr filtern.

Das war es schon – wir haben den kompletten Code im Hauptformular und können so steuern, dass das Unterformular **sfmBestellungen** nur die Bestellungen anzeigt, die dem im Unterformular **sfmKunden** ausgewählten Datensatz zugeordnet sind.

Eine Voraussetzung ist dafür zu erfüllen, damit Sie die Ereignisprozeduren der untergeordneten Formulare implementieren können: Diese müssen jeweils ein eigenes Klassenmodul enthalten.

Dies stellen Sie am schnellsten über den Wert **Ja** für die Eigenschaft **Enthält Modul** des jeweiligen Formulars ein.

Verknüpfung aller Unterformulare

Damit alle Unterformulare jeweils die Daten des folgenden Unterformulars filtern, benötigen wir für jedes Unterformular die

Die Prozedur **Form_Load** des Hauptformulars setzt dann die Verweise für die vier **Form**-Variablen auf die entsprechenden **Form**-Elemente in den Unterformular-Steuer-elementen. Für die ersten drei legt sie außerdem fest, dass in diesem Klassenmodul auch nach Implementierungen des Ereignisses **OnCurrent** gesucht werden soll.

Schließlich fehlen noch die Implementierungen der Ereignisse, die jeweils ähnlich aufgebaut sind, aber sich auf die verschiedenen Unterformulare beziehen. **frm_sfmKunden_Current** stellt den Filter von **frm_sfmBestellungen**

auf den aktuellen Kunden ein, **frm_sfmBestellungen_Current** den Filter von **frm_sfmBestelldetails** und **frm_sfmBestelldetails_Current** den von **frm_sfmArtikel**.

Benennung der Form-Variablen

Warum haben wir die Formular-Variablen nicht einfach genauso benannt wie die Unterformulare? Wir haben es zunächst ausprobiert, aber Access kommt anscheinend nicht klar, wenn es einerseits ein Unterformular-Steuer-element und andererseits eine mit **WithEvents** deklarierte Variable mit dem gleichen Namen gibt.

The screenshot shows the 'frmVerknuepfteTabellen' application with three subforms displayed, each filtered to show data for the selected customer 'Ana Trujillo Emparedados y helados'.

Kunden:

KundeID	Kunden-Code	Firma	Mari
1	ALFKI	Alfreds Futterkiste	Mari
2	ANATR	Ana Trujillo Emparedados y helados	Ana
3	ANTON	Antonio Moreno Taquería	Anto
4	AROUT	Around the Horn	Thor
5	BERGS	Berglunds snabbköp	Chris
6	BLAUS	Blauer See Delikatessen	Hani
7	BLOPP	Blondel père et fils	Fréd

Bestellungen:

Bestell-Nr	KundeID	Angestellte(r)	Bestelldatum	Lieferdatum
10308	Ana Trujillo Emp	King, Robert	18.Sep.1996	16.Okt.1996
10625	Ana Trujillo Emp	Leverling, Janet	08.Aug.1997	05.Sep.1997
10759	Ana Trujillo Emp	Leverling, Janet	28.Nov.1997	26.Dez.1997
10926	Ana Trujillo Emp	Peacock, Margaret	04.Mrz.1998	01.Apr.1998
*	(Neu)			

Bestelldetails:

Bestelldetail	Bestell-Nr	Artikel	Einzelpreis
162	10308	Gudbrandsdalsost	14,40 €
163	10308	Outback Lager	6,00 €
*	(Neu)		0,00 €

Artikel:

ArtikelID	Artikelname	Lieferant
69	Gudbrandsdalsost	Norske Meierier
*	(Neu)	

Bild 7: Gefilterte Unterformulare

Wir haben bei diesem Versuch reproduzierbar einen Absturz von Access verursacht, als wir versucht haben, das Ereignis **Beim Anzeigen** für das Element **sfm_Bestellungen** anzulegen. Aus diesem Grund haben wir allen **Form**-Variablen noch das Präfix **frm_** vorangestellt.

Test des Formulars

Nach dem Wechsel in die Formularansicht funktioniert das Formular wie gewünscht. Wenn Sie einen der Einträge des obersten Unterformulars mit den Daten der Tabelle **tblKunden** anklicken, zeigt das nächste Unterformular die dazu gehörende Bestellung, das nächste die zur Bestellung gehörenden Bestelldetails und das letzte den Artikel zum jeweiligen Bestelldetail an (s. Bild 7).

Flexible Gestaltung

Nun ist es allerdings etwas umständlich, für jede neue Konstellation ein neues dieser Formulare samt Unterformularen erstellen zu müssen. Also machen wir das Formular etwas flexibler, indem wir dem Benutzer die Auswahl der Tabellen und der Verknüpfungsfelder ermöglichen.

Kombinationsfeld rauf und runter

Kombinationsfelder sind wirklich praktisch, wenn es um die Auswahl von Lookup-Daten geht. Aber immerhin muss man immer noch auf die Schaltfläche zum Aufklappen der Liste klicken, um einen der Einträge auszuwählen. Wir haben uns eine noch schnellere Variante einfallen lassen, bei der Sie einfach nur mit der Maus in das Kombinationsfeld klicken und diese dann nach oben oder unten bewegen müssen, um zwischen den Einträgen zu navigieren. Dieser Beitrag zeigt, wie Sie ein Kombinationsfeld mit dieser Funktion ausstatten.

Beispieldatenbank

Als Beispiel verwenden wir eine Datenbank mit einer einfachen Kundentabelle sowie einem Formular, das die enthaltenen Datensätze per Kombinationsfeld zur Auswahl anbietet (siehe Bild 1). Das Kombinationsfeld heißt **cboKunden** und verwendet die folgende Abfrage als Datensatzherkunft:

```
SELECT tblKunden.KundeID, tblKunden.Firma
FROM tblKunden ORDER BY tblKunden.Firma;
```

Damit das Primärschlüsselfeld nicht eingeblendet wird, sondern nur das Feld **Firma**, stellen Sie die Eigenschaft **Spaltenanzahl** auf den Wert **2** und die Eigenschaft **Spaltenbreiten** auf den Wert **0cm** ein. Sie können das Kombinationsfeld nun wie gewohnt aufklappen und einen Eintrag auswählen.

Rauf und runter

Wenn der Benutzer das Kombinationsfeld öffnet, findet er dieses in der Regel leer vor (siehe Bild 2). Nun soll der Benutzer mit der Maus auf das Kombinationsfeld klicken und bei gedrückter Maustaste den Mauszeiger nach unten bewegen können, um den ersten Eintrag der Datensatzherkunft auszuwählen. Bewegt er den Mauszeiger bei gedrückter Maustaste weiter nach unten, wird der nächste Eintrag ausgewählt und so weiter. Bewegt er hingegen den Mauszeiger bei gedrückter Maustaste nach oben, wird wieder der vorherige Eintrag ausgewählt. Auf diese Weise kann der Benutzer durch alle vorhandenen Einträge navigieren.

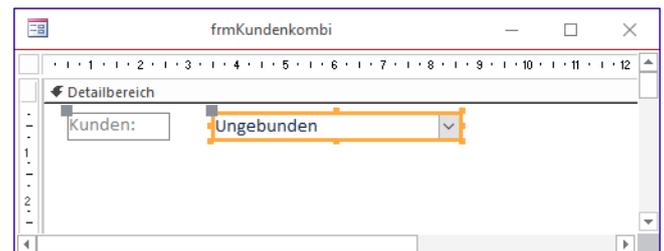


Bild 1: Beispielformular mit dem Kombinationsfeld

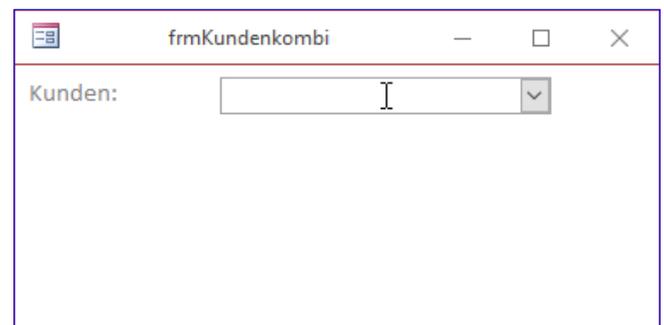


Bild 2: Kombinationsfeld ohne Auswahl

Benötigte Ereignisse

Bevor wir die neue Funktion programmieren, müssen wir uns überlegen, welche Ereignisse wir hier verwenden können. Ganz sicher werden wir das Ereignis **Bei Mausbewegung** des Kombinationsfeldes benötigen, denn dieses wird ja immer ausgelöst, wenn sich die Maus ein kleines Stückchen bewegt hat.

Gegebenenfalls benötigen wir auch noch die beiden Ereignisse **Bei Maustaste ab** und **Bei Maustaste auf** des Kombinationsfeldes, um Vorbereitungen und abschließende Arbeiten für den eigentlichen Vorgang des Ziehens mit der Maus durchzuführen.

Das Ereignis Bei Mausbewegung

Dieses für unsere Lösung existenzielle Ereignis wird alle paar Millisekunden ausgelöst. Es liefert dann mit den folgenden Parametern wichtige Werte:

- **Button:** Gibt an, welche Taste der Maus beim Bewegen der Maus gedrückt ist. Es gibt die für uns interessanten Werte **0** (keine Taste), **1** (linke Taste) und **2** (rechte Taste).
- **Shift:** Gibt an, ob der Benutzer beim Bewegen der Maus eine der Tasten **Umschalt**, **Strg** oder **Alt** gedrückt hält.
- **X:** Liefert die aktuelle X-Position des Mauszeigers.
- **Y:** Liefert die aktuelle Y-Position des Mauszeigers.

Die Positionen **X** und **Y** werden in 15er-Schritten und in der Größe Twips geliefert. Jeder Bildschirmpunkt hat eine bestimmte Anzahl Twips, wobei diese Anzahl von verschiedenen Faktoren abhängen kann, in der Regel aber 15 beträgt.

Das Ereignis Bei Maustaste ab

Dieses Ereignis wird beim Herunterdrücken der Maustaste ausgelöst. Es liefert die gleichen Parameter wie das Ereignis **Bei Mausbewegung**.

Programmierung der Funktion

Grundsätzlich sieht die Vorgehensweise wie folgt aus:

Wir definieren eine Ereignisprozedur für das Ereignis **Bei**

```
Private Sub cboKunden_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    lngPosVorher = Y
    intIndex = Me!cboKunden.ListIndex
    intCount = Me!cboKunden.ListCount
    If intIndex = -1 Then
        intIndex = 0
    End If
End Sub
```

Listing 1: Aktionen, wenn der Benutzer die Maustaste herunterdrückt

Mausbewegung. In dieser prüfen wir, ob sich die vertikale Position des Mauszeigers verändert hat und in welchem Umfang. Zusätzlich legen wir fest, nach welcher in der Y-Richtung zurückgelegten Strecke der nächste oder der vorherige Eintrag des Listenfeldes angezeigt werden soll.

Wir könnten die kleinste Einheit wählen, also 15 Twips, aber dann würden wir sehr schnell durch die Einträge rutschen und hätten kaum die Möglichkeit, gezielt einen Eintrag auszuwählen. Also stellen wir die zurückzulegende Strecke etwas größer ein, zum Beispiel auf 150 Twips. Dies legen wir in einer Konstanten namens **cIntervall** fest, die wir im Kopf des Klassenmoduls des Formulars wie folgt deklarieren:

```
Const cIntervall As Integer = 150
```

Um zu prüfen, ob der Mauszeiger seit dem letzten Ändern des ausgewählten Eintrags die in **cIntervall** angegebene Strecke zurückgelegt hat, müssen wir die Position beim Ändern des Eintrags speichern. Dazu verwenden wir eine weitere Variable, die wir ebenfalls modulweit deklarieren und die wie folgt aussieht:

```
Dim lngPosVorher As Long
```

Wir wollen immer nur bis zum letzten Eintrag scrollen, nicht darüber hinaus. Daher deklarieren wir eine Variable, mit der wir die Anzahl der enthaltenen Einträge speichern:

```
Dim intCount As Integer
```

Schließlich müssen wir uns auch noch merken, auf welcher Position im Kombinationsfeld wir uns gerade befinden, um von dort auf die vorherige oder nächste Position zu springen. Dazu legen wir noch eine Variable namens **intIndex** fest:

Schnellsuche mit Verzögerung

Eine Schnellsuche ist eine tolle Sache: Sie geben Buchstabe für Buchstabe ein und erhalten direkt nach der Eingabe das passende Suchergebnis. Dummerweise kann es je nach Datenherkunft und Backend auch einmal etwas länger dauern beziehungsweise performancemindernd sein, wenn Sie allzu oft neue Abfragen absenden. Doch es gibt einen feinen Kompromiss: Eine Schnellsuche, die nicht direkt nach der Eingabe eines Buchstabens das neue Suchergebnis präsentiert, sondern die erst nach einem gewissen Zeitraum ohne neue Eingabe neue Daten lädt.

Beispieldatenbank

Unser Versuchsaufbau für diese kleine Lösung sieht im Entwurf wie in Bild 1 aus. Wir haben dort zwei Textfelder angelegt, von denen das eine **txtSchnellsuche** heißt und die übliche Schnellsuche abbilden soll. Darunter finden Sie das Textfeld **txtSchnellsucheMitVerzoegerung**, das unsere angepasste Variante liefert.

Das Listenfeld **IstKunden** darunter verwendet als Datensatzherkunft die folgende Abfrage:

```
SELECT tb1Kunden.KundeID, tb1Kunden.Firma
FROM tb1Kunden
ORDER BY tb1Kunden.Firma;
```

Die Eigenschaft **Spaltenanzahl** stellen wir auf den Wert **2** und die Eigenschaft **Spaltenbreiten** auf den Wert **Ocm** ein, damit nur die Spalte mit dem Feld **Firma** angezeigt wird.

Herkömmliche Schnellsuche

Die einfache Schnellsuche, bei der das Suchergebnis mit der Eingabe jedes einzelnen Zeichens aktualisiert wird, sieht in der Formularansicht etwa wie in Bild 2 aus.

Um dies zu realisieren, hinterlegen Sie für die Ereignisseigenschaft **Bei Änderung** des Textfeldes **txtSuchbegriff** die Prozedur aus Listing 1.

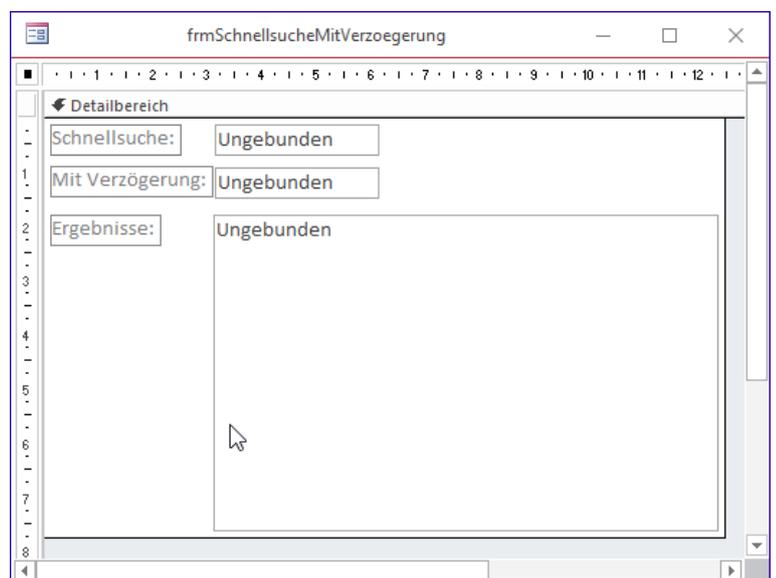


Bild 1: Formular mit Textfeldern für die Schnellsuche und die Schnellsuche mit Verzögerung

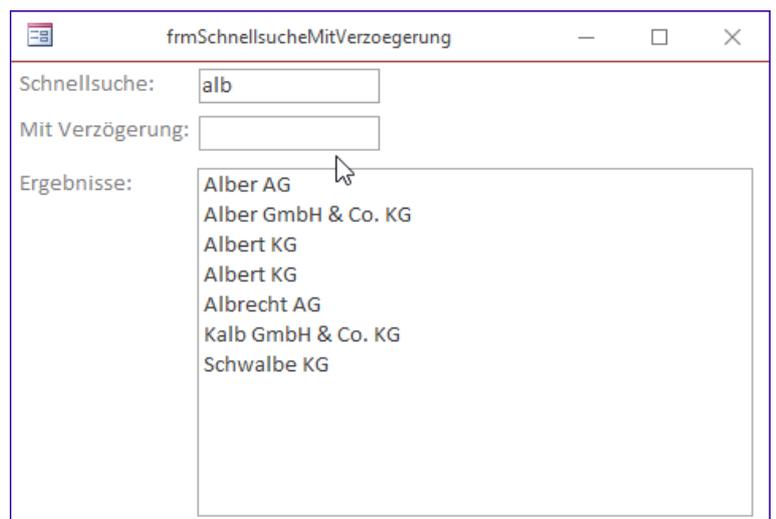


Bild 2: Einfache Schnellsuche im Einsatz

Ribbon für das Ticketsystem

Wir haben in Access im Unternehmen bereits einige Techniken rund um das Ribbon vorgestellt, aber noch sehr wenige unserer Lösungen tatsächlich mit der Ribbon-Technologie ausgestattet. Das wollen wir am Beispiel des Ticketsystems, das wir in einer eigenen Beitragsreihe behandeln, nachholen. Dort fügen wir nicht nur einige Ribbon-Schaltflächen zur Steuerung der Anwendung ein, sondern legen auch einen kleinen Backstage-Bereich zur Einstellung von Anwendungsoptionen an.

Neben den üblichen Schaltflächen, mit denen wir die Formulare und Funktionen der Anwendung aufrufen wollen, soll das Ticketsystem auch einen Backstagebereich erhalten, über den Sie Anwendungseinstellungen einsehen und ändern können. In diesem Fall ist in der Anwendung das Verzeichnis in Outlook festzulegen, in das der Benutzer die E-Mails ziehen soll, die vom Ticketsystem berücksichtigt werden sollen. Diesen Pfad speichern wir in der Datenbank in der Tabelle **tblOptionen**, die dafür das Feld **Verzeichnis** vorhält.

Der Backstagebereich soll wie in Bild 1 aussehen. Im Gegensatz zur Ribbonleiste selbst ist der Backstagebereich ein wenig komplizierter aufgebaut, aber das soll uns nicht aufhalten. Also schauen wir uns den Code an, der zur Definition unserer Optionen im Backstagebereich notwendig ist.

Der erste Abschnitt der XML-Definition enthält das Element **customUI**, in welchem wir die Callback-Funktionen für das Laden des Ribbons (**onLoad**) und für das Einlesen von Bilddateien (**loadImage**) festgelegt haben (s. Listing 1). Danach folgt die Definition der Ribbon-Leiste (**ribbon**), die wir uns weiter unten genauer ansehen. Schließlich legen wir das **backstage**-Element an, welches die Definition des Backstage-Bereichs enthält – oder in unserem Fall die Erweiterung des Backstage-Bereichs. Wie Sie der Abbildung entnehmen können, sind unter der aktuellen Definition noch alle eingebauten

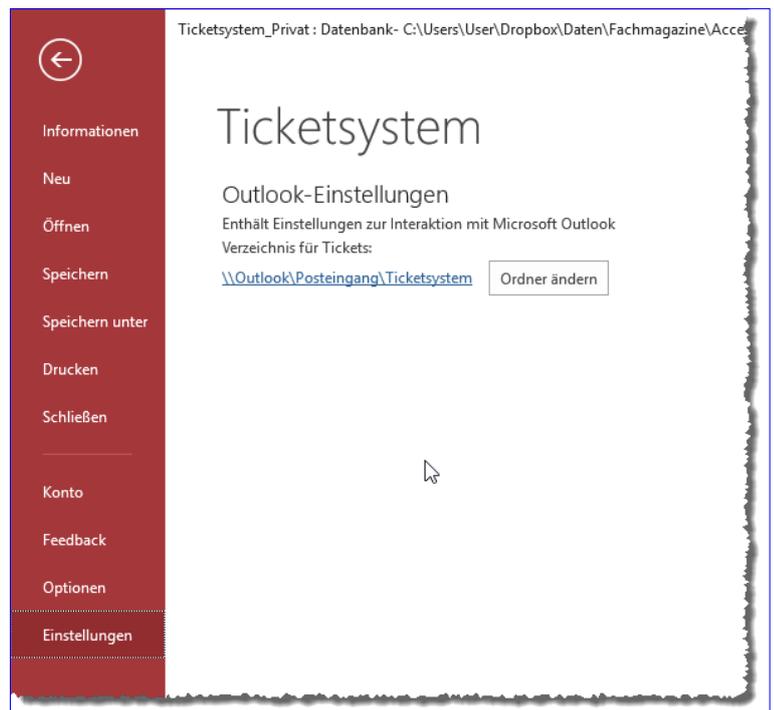


Bild 1: Benutzerdefinierter Backstage-Bereich

Elemente vorhanden, was zur Entwicklungszeit durchaus noch sinnvoll sein kann. Später werden wir diese Elemente weitgehend ausblenden.

Vorab noch kurz zu den beiden durch die Callbacks **onLoad** und **loadImage** ausgelösten Prozeduren. **onLoad** löst die folgende Prozedur aus, die wir, wie die meisten anderen mit dem Ribbon in Zusammenhang stehenden Prozeduren, im Modul **mdlRibbon** untergebracht haben:

```
Sub onLoad_Main(ribbon As IRibbonUI)
    Set objRibbon_Main = ribbon
End Sub
```

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_Main" loadImage="loadImage">
  <ribbon>
  ...
</ribbon>
<backstage>
  <tab id="tabEinstellungenBackstage" label="Einstellungen" title="Ticketsystem" firstColumnMaxWidth="600">
    <firstColumn>
      <group id="grpOutlook" label="Outlook-Einstellungen" style="normal" helperText="Enthält Einstellungen
zur Interaktion mit Microsoft Outlook">
        <primaryItem/>
        <topItems>
          <labelControl id="lblVerzeichnisFuerTickets" label="Verzeichnis für Tickets:"/>
          <layoutContainer id="xxx33091" align="left">
            <hyperlink id="hypVerzeichnisFuerTickets" getLabel="getLabel" onAction="onAction"/>
            <button id="btnOrdnerAendern" onAction="onAction" label="Ordner ändern"/>
          </layoutContainer>
        </topItems>
        <bottomItems/>
      </group>
    </firstColumn>
  </tab>
</backstage>
</customUI>
```

Listing 1: Definition des Backstage-Bereichs

Die hier verwendete Variable deklarieren wir wie folgt im gleichen Modul:

```
Public objRibbon_Main As IRibbonUI
```

Diese Variable müssen wir mit einem Verweis auf die geladene Ribbon-Definition füllen, damit wir ihre Methoden **Invalidate** und **InvalidateControl** auslösen können. Diese sorgen dafür, dass die Inhalte, Bilder und weiteren Eigenschaften neu geladen werden, sollten sich die Anforderungen geändert haben. Die Callback-Eigenschaft **loadImage** ruft die Prozedur aus Listing 2 auf. Diese soll zunächst prüfen, ob die Tabelle **MSysResources**, welche die Bilder enthält, die im Ribbon angezeigt werden sollen, in der Datenbank vorhanden ist.

Die Prozedur **loadImage** wird für jedes Bild, das in ein Steuerelement des Ribbons geladen werden soll, je einmal

aufgerufen. Dabei wird mit dem Parameter **image** jeweils der Name des zu verwendenden Bildes übergeben. Dieser ist immer in der Definition des jeweiligen Objekts für das Attribut **image** enthalten, also beispielsweise:

```
<button id="btnOffeneMails" image="mail" label="Offene
Mails" onAction="onAction" size="large"/>
```

Damit wird dann die Funktion **PicFromSharedResource_Ribbon** aufgerufen, die sich, wie einige andere für die Ermittlung von Bildern aus der Tabelle **MSysResources**, im Modul **mdlRibbonImages** befindet (siehe Beispieldatenbank). Doch zurück zur Backstage-Definition. Diese wird durch je ein öffnendes und schließendes **backstage-**Element eingeschlossen. Damit unser Eintrag unten unter den übrigen eingebauten Elementen angezeigt wird, fügen wir unterhalb des **backstage-**Elements ein **tab-**Element hinzu. Diesem geben wir eine Beschriftung, einen Titel

```
Public Sub loadImage(Control, ByRef image)
    Dim lngID As Long
    On Error Resume Next
    lngID = Nz(DLookup("ID", "MSysResources", "Name = '" & Control & "'"), 0)
    If Err.Number = 3078 Then
        MsgBox "Die Tabelle 'MSysResources' mit den Images für die Anzeige im Ribbon fehlt." & vbCrLf _
            & "Fügen Sie die Images mit dem Ribbon-Admin hinzu", vbOKOnly + vbExclamation, "Tabelle MSysResources fehlt"
        Exit Sub
    End If
    On Error GoTo 0
    If lngID = 0 Then
        MsgBox "Das Image '" & Control & "' ist nicht in der Tabelle MSysResources vorhanden." & vbCrLf _
            & "Fügen Sie dieses über den Kontextmenüeintrag 'Benutzerdefiniertes Image hinzufügen' " & vbCrLf _
            & "des image-Attributs des entsprechenden Ribbon-Steuerelements hinzu."
    Else
        Set image = PicFromSharedResource_Ribbon(CStr(Control))
    End If
End Sub
```

Listing 2: Prozedur zum Laden der Ribbon-Bilder

und eine Breite mit. Der Titel ist groß und breit im Bereich rechts neben der vertikalen Tableiste zu erkennen:

```
<tab id="tabEinstellungenBackstage" label="Einstellungen"
title="Ticketssystem" firstColumnMaxWidth="600">
```

Wir benötigen nur eine der zwei möglichen Spalten, also legen wir unter dem **tab**-Element nur ein **firstColumn**-Element an, das die übrigen Elemente enthält. Zum Beispiel das **group**-Element, dem wir ein leeres **primaryItem**-Element, ein mit einigen Elementen gefülltes **topItems**-Element und ein wiederum leeres **bottomItems**-Element hinzufügen. Das **group**-Element enthält auch einige Attribute zum Beispiel eine Beschriftung (**Outlook-Einstellungen**) und einen **helperText**, der unterhalb der Beschriftung erscheint:

```
<group id="grpOutlook" label="Outlook-Einstellungen"
style="normal" helperText="Enthält Einstellungen zur In-
teraktion mit Microsoft Outlook">
```

Unter **topItems** fügen wir einige Elemente hinzu – als Erstes ein **labelControl**, das eine Beschriftung enthält:

```
<labelControl id="lblVerzeichnisFuerTickets"
label="Verzeichnis für Tickets:"/>
```

Darunter wollen wir dann zwei Elemente nebeneinander anordnen, weshalb wir einen **layoutContainer** mit links-zentrierter Ausrichtung nutzen:

```
<layoutContainer id="xxx33091" align="left">
...
</layoutContainer>
```

Das erste Element ist ein **hyperlink**-Element, das über die Callback-Funktion **getLabel** mit einer Beschriftung gefüllt wird und deren **onAction**-Callback die VBA-Prozedur **onAction** auslöst. Dieses Element ist logischerweise wie ein Hyperlink ausgezeichnet, also in blauer und unterstrichener Schrift:

```
<hyperlink id="hypVerzeichnisFuerTickets"
getLabel="getLabel" onAction="onAction"/>
```

Rechts daneben landet ein Button mit der Beschriftung **Ordner ändern**.

Er ist wie folgt definiert und soll die Auswahl eines neuen Ordners ermöglichen:

```
<button
id="btnOrdnerAendern"
onAction="onAction"
label="Ordner ändern"/>
```

Logik hinter dem Backstage-Bereich

Die Steuerelemente des Backstage-Bereichs enthalten ein paar Callback-Attribute, die wir im VBA-Modul **mdlRibbons** um entsprechende Prozeduren anreichern.

Die erste sorgt dafür, dass bei einem Klick auf die Schaltfläche **Ordner ändern** der Outlook-Dialog zum Auswählen eines der Outlook-Ordner angezeigt wird (s. Bild 2). Dies löst die Prozedur **onAction** aus und übergibt dieser per Parameter einen Verweis auf das auslösende Steuerelement. Dessen Name ermitteln wir mit der Eigenschaft **ID** des Parameters **Control** und führen nach Steuerelement einen der Zweige einer **Select Case**-Bedingung aus:

```
Sub onAction(Control As IRibbonControl)
    Dim objOutlook As Outlook.Application
    Dim objNamespace As Outlook.NameSpace
    Dim objFolder As Outlook.Folder
    Dim strOrdner As String
    Select Case Control.ID
        Case "btnOrdnerAendern"
            Call OutlookOrdnerAuswaehlen
            ...
    End Select
End Sub
```

In unserem Fall handelt es sich um den Zweig mit dem Wert **btnOrdnerAendern**, der lediglich den Aufruf einer weiteren Prozedur namens **OutlookOrdnerAuswaehlen** enthält.

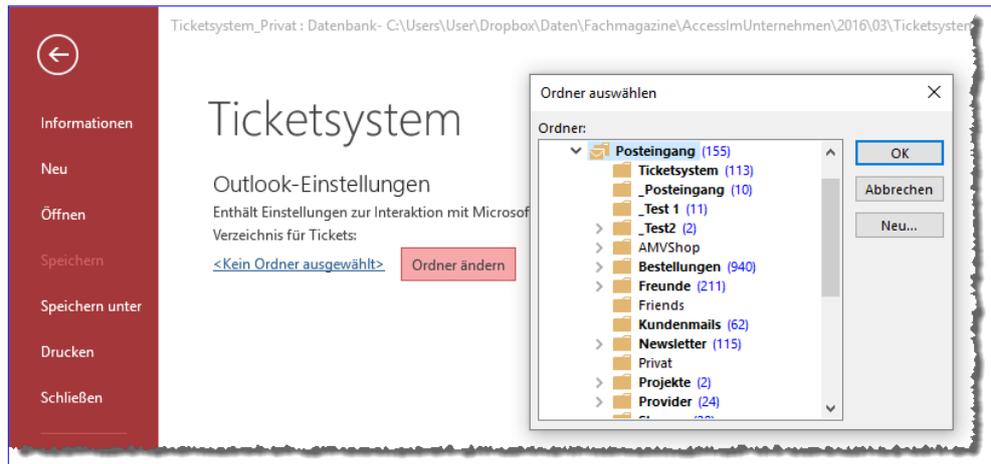


Bild 2: Aufruf des **Ordner auswählen**-Dialogs von Outlook

Outlook-Ordner auswählen

Die Prozedur **OutlookOrdnerAuswaehlen** sieht wie in Listing 3 aus. Sie ermittelt zunächst den Namen des aktuell aktiven Fensters, und zwar mit einer Hilfsfunktion namens **GetActiveWindowTitle**.

Diese nutzt zwei API-Funktionen, nämlich **GetForegroundWindow** und **GetWindowText**, die im Kopf des Moduls **mdlAPI** deklariert werden (s. Listing 4). Die Funktion **GetActiveWindowTitle** ermittelt zunächst mit der Funktion **GetForegroundWindow** ein Handle auf das aktive Fenster. Dann liest sie mit **GetWindowText** den Titel des Fensters ein und gibt diesen als Rückgabewert der Funktion zurück. Nachdem der Fenstertitel des aktiven Fensters so in der Variablen **strFenster** der Prozedur **OutlookOrdnerAuswaehlen** gelandet ist, ruft diese eine weitere Funktion namens **GetFolderName** auf. Diese ist scheinbar recht einfach gestrickt, denn sie ruft lediglich die Methode **PickFolder** eines Objekts namens **GetMAPI** auf:

```
Public Function GetFolderName() As String
    Dim objFolder As Outlook.Folder
    Set objFolder = GetMAPI.PickFolder
    If Not objFolder Is Nothing Then
        GetFolderName = objFolder.FolderPath
    End If
End Function
```

```
Public Sub OutlookOrdnerAuswählen()
    Dim strOrdner As String
    Dim db As DAO.Database
    Dim lngCount As Long
    Dim strFenster As String
    strFenster = GetActiveWindowTitle
    strOrdner = GetFolderName
    Set db = CurrentDb
    db.Execute "UPDATE tblOptionen SET Verzeichnis = '" & strOrdner & "'", dbFailOnError
    lngCount = db.RecordsAffected
    If lngCount = 0 Then
        db.Execute "INSERT INTO tblOptionen(Verzeichnis) VALUES('" & strOrdner & "'", dbFailOnError
    End If
    objRibbon_Main.InvalidateControl "hypVerzeichnisFuerTickets"
    AppActivate strFenster
End Sub
```

Listing 3: Die Prozedur zum Auswählen eines neuen Outlook-Ordners

Allerdings ist dies kein Objekt, sondern wiederum eine Funktion, die ein **MAPI-Folder**-Objekt zurückliefert. Diese Funktion wollen wir im Rahmen dieses Beitrags jedoch nicht erläutern – bei Interesse schauen Sie sich einfach den Quellcode im Modul **mdlOutlook** der Beispieldatenbank an. Der Benutzer wählt nun im Dialog **Ordner auswählen** den Ordner aus, der als Quellordner für die im Ticketsystem zu verarbeitenden E-Mails dient, die der Benutzer zu diesem Zweck aus dem Posteingang dort hineinzieht. Nachdem das Verzeichnis, zum Beispiel **\\Outlook\Posteingang\Ticketsystem**, in der Variablen

strOrdner gelandet ist, erstellt die Prozedur **OutlookOrdnerAuswählen** ein neues **Database**-Objekt auf Basis der aktuellen Datenbank und ruft dessen **Execute**-Methode auf.

Als Parameter übergibt sie dieser eine **UPDATE**-Abfrage, welche den Wert aus **strOrdner** in das Feld **Verzeichnis** der Tabelle **tblOptionen** einträgt. Wenn dies erfolgreich ist, liefert **db.RecordsAffected** einen Wert ungleich **0**. Ist der Wert dennoch **0**, was auf einen Fehler hinweist, geht die Prozedur davon aus, dass noch kein Datensatz in der Ta-

belle vorhanden ist, um die Option zu speichern. Dann erfolgt der Aufruf einer weiteren Aktionsabfrage, diesmal des Typs **INSERT INTO**. Diese legt dann einen neuen Datensatz in der Tabelle **tblOptionen** an und trägt direkt das Verzeichnis aus **strOrdner** dort ein.

Schließlich folgt eine wichtige Anweisung:

```
objRibbon_Main.InvalidateControl
"hypVerzeichnisFuerTickets"
```

```
Private Declare Function GetForegroundWindow Lib "user32" () As Long
Private Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" _
    (ByVal hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long

Public Function GetActiveWindowTitle() As String
    Dim nHwnd As Long
    Dim sTitle As String
    Dim nResult As Long
    nHwnd = GetForegroundWindow()
    sTitle = Space$(255)
    nResult = GetWindowText(nHwnd, sTitle, Len(sTitle))
    GetActiveWindowTitle = Left$(sTitle, nResult)
End Function
```

Listing 4: Funktion zum Ermitteln des Titels des aktuellen Windows-Fensters

Lösung zum Formular-Add-In umbauen

Im Beitrag »Verknüpfte Daten suchen« haben wir eine Lösung beschrieben, die man zwar prima in eine Datenbank integrieren und dann dort nutzen kann. Allerdings ist es doch aufwändig, jedes Mal erst die benötigten Objekte zu importieren – und außerdem steigert das Verteilen des gleichen Codes auf viele verschiedene Datenbanken nicht unbedingt die Wartbarkeit. Da die genannte Lösung nicht für den Benutzer, sondern eher für den Entwickler gedacht ist, wollen wir diese in ein Add-In umwandeln. Dieser Beitrag zeigt die Vorgehensweise und auch die Fallstricke.

Die umzuwandelnde Datenbank enthält zwei Formulare, von denen eines ein Unterformular ist, sowie eine Tabelle. Beim Aufrufen des Add-Ins über den entsprechenden Ribbon-Eintrag soll das Hauptformular der Anwendung geöffnet werden. Wir gehen nun Schritt für Schritt vor, um die Fallstricke Stück für Stück zu entschärfen.

Tabelle USysRegInfo anlegen

Aufmerksame Leser unseres Magazins wissen, dass eine Tabelle namens **USysRegInfo** für ein Add-In unerlässlich ist. Diese speichert die Informationen, die beim Hinzufügen des Add-Ins zur Liste der Add-Ins der Anwendung in die Registry geschrieben werden, damit Access dieses Add-In in der Add-In-Liste anzeigt.

Dort steht dann beispielsweise, wo die Add-In-Datenbank gespeichert ist und welche Funktion aufgerufen werden

soll, wenn der Benutzer das Add-In startet. In unserem Fall sieht die Tabelle wie in Bild 1 aus. Die erste Zeile gibt an, dass beim Starten des Add-Ins eine Funktion namens **Autostart** aufgerufen werden soll.

Die zweite enthält das Verzeichnis, in dem das Add-In gespeichert sein wird. **IACCDIR** wird beim Registrieren automatisch durch das Add-In-Verzeichnis ersetzt.

Startfunktion definieren

Neben der Tabelle müssen wir also eine Funktion definieren, welche beim Start des Add-Ins aufgerufen wird, und dort die durchzuführenden Schritte eintragen. In unserem Fall wollen wir nur das Formular **frmTabellen** aufrufen.

Die Funktion legen wir in einem neuen Standardmodul namens **mdlAddIn** an:

Subkey	Type	ValName	Value
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\VerknuepfteDatenSuchen	1 Expression		=Autostart()
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\VerknuepfteDatenSuchen	1 Library		ACCDIR\VerknuepfteDatenSuchen.acdda
HKEY_CURRENT_ACCESS_PROFILE\Menu Add-Ins\VerknuepfteDatenSuchen	1 Description		Dieses Tool hilft Ihnen, sich die Daten
*	0		

Bild 1: Die Tabelle **USysRegInfo**

```
Function Autostart()  
    DoCmd.OpenForm "frmTabellen"  
End Function
```

Anwendungseigenschaften anpassen

Damit das Add-In im Add-In-Manager eine gute Figur abgibt, stellen Sie ein paar Eigenschaften für die Anwendung ein. Dazu gehören der **Titel**, die **Firma** und der **Kommentar** (s. Bild 2).

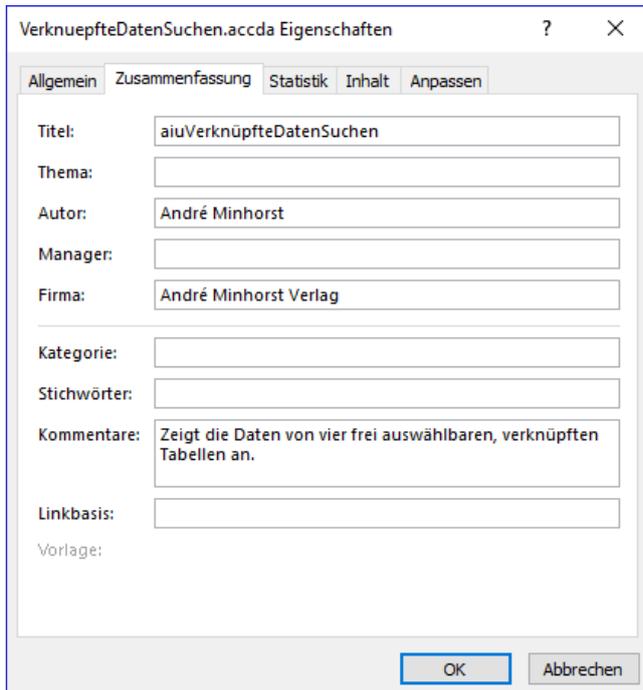


Bild 2: Ändern der Eigenschaften der Access-Anwendung

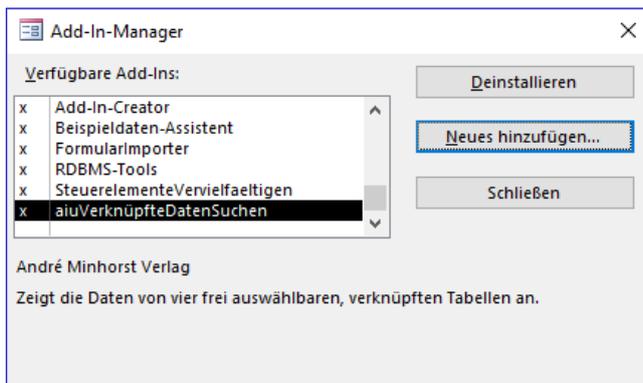


Bild 3: Add-In-Manager von Access

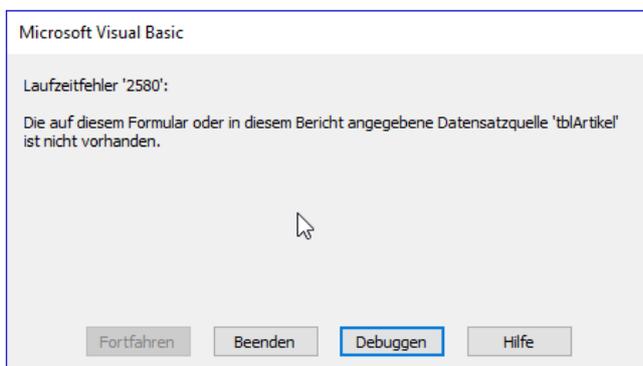


Bild 4: Fehler beim Versuch, das neue Add-In zu starten

Datenbanktyp ändern

Damit die Add-In-Datenbank als solche erkannt wird, ändern Sie die Dateierweiterung der Datenbank von **.accdb** auf **.accda**.

Add-In registrieren

Nun können Sie das Add-In bereits registrieren. Dazu schließen Sie die Anwendung und öffnen eine beliebige andere Datenbankanwendung in Access. Wählen Sie dann den Ribbon-Eintrag **Datenbanktools|Add-Ins|Add-Ins|Add-In-Manager** aus. Klicken Sie auf die Schaltfläche **Neues hinzufügen...**, um einen **Datei öffnen**-Dialog zu öffnen und die **.accda**-Datenbank auszuwählen. Nach der Auswahl unserer Datei wird diese wie in Bild 3 im Add-In-Manager angezeigt. Hier finden Sie dann auch die Änderungen in den Eigenschaften der Anwendung wieder.

Erster Start

Der erste Startversuch durch Anklicken des neuen Eintrags in der Add-In-Liste im Ribbon führt dann gleich zum ersten Fehler (s. Bild 4). Gewohnheitsmäßig werden Sie nun auf die Schaltfläche **Debuggen** klicken und sich den Fehler ansehen – hierzu folgt später eine Warnung!

Schauen wir zunächst auf den Fehler: Access kann die Tabelle **tblArtikel**, die als Datenherkunft des Formulars angegeben werden soll, nicht finden. Das hat normalerweise den Grund, dass keine solche Tabelle in der Datenbank enthalten ist.

In diesem Fall haben wir das Add-In aber zum Experimentieren von einer Datenbank aus geöffnet, die wir mit genau den gleichen Tabellen ausgestattet haben, die wir auch schon beim Erstellen des Add-Ins zum Testen verwendet haben. Grundsätzlich ist die Tabelle also vorhanden.

Allerdings greift das Add-In standardmäßig auf die Tabellen in der Add-In-Datenbank zu – aber dort befindet sich ja nur noch die Tabelle **tblKonfigurationen**! Die Tabelle **tblArtikel** jedoch befindet sich in der Host-Datenbank und kann somit nicht so einfach als Datenherkunft eines

Formulars in der Add-In-Datenbank verwendet werden. Wie wir dies lösen, schauen wir uns gleich an. Bevor wir uns ans Werk machen, jedoch noch die versprochene Warnung.

Warnung: Code im Add-In ändern

Wie erwähnt, möchte man beim Auftauchen von Fehlern in Add-Ins genau wie bei normalen Datenbanken gleich in den VBA-Editor eintauchen und den Fehler beheben. Wenn Sie allerdings Elemente einer Add-In-Datenbank ändern, während diese von einer anderen Access-Datenbank als Add-In geöffnet wurde, werden diese Änderungen nicht gespeichert! Sie können zwar temporär Änderungen durchführen und auch testen, aber Sie sollten, wenn Sie dies vorhaben, beispielsweise Änderungen am Code in die Zwischenablage kopieren, die Datenbank schließen, die Add-In-Datenbank öffnen und dort die Änderungen reproduzieren.

Erst danach können Sie die Änderungen durch erneuten Aufruf des Add-Ins in der Host-Datenbank testen. Das ist leider etwas umständlich, aber nicht zu ändern.

Hierbei gibt es aber noch einen Fallstrick: Durch das Hinzufügen der Datenbank zur Add-In-Liste wurde diese in den Add-In-Ordner kopiert. Sie müssen also auch noch aufpassen, dass Sie die Änderungen an der richtigen Datenbank durchführen. Es gibt zwei Möglichkeiten:

- Sie ändern direkt die Add-In-Datenbank im Add-In-Verzeichnis.
- Sie ändern die Originaldatenbank und kopieren diese nach dem Ändern jeweils über die Version im Add-In-Verzeichnis.

Wenn Sie Änderungen an Stellen vornehmen, welche die Funktion als Add-In an sich betreffen – also beispielsweise Einstellungen der Anwendungseigenschaften oder in der Tabelle **USysRegInfo** –, müssen Sie das Add-In auch neu registrieren.

Zugriff auf Datenherkünfte von Tabellen, die sich nicht im Add-In befinden

Die Zeile, die den obigen Fehler ausgelöst hat, befindet sich in der folgenden Prozedur:

```
Private Sub UnterformulareAktualisieren()  
    ...  
    Me!sfmTabelle4.Form.RecordSource = Nz(Me!cboTabellen4)  
    ...  
End Sub
```

Gibt es eine Alternative zum Zuweisen des Namens der Tabelle zur Eigenschaft **RecordSource** des Unterformulars? Wir könnten versuchen, ein Recordset auf Basis dieser Tabelle zu erstellen und dieses der Eigenschaft **Recordset** des Unterformulars zuweisen. Dazu benötigen wir zunächst eine **Database**-Variable und jeweils eine **Recordset**-Variable für die vier Unterformulare. Die **Database**-Variable füllen wir wie üblich mit der Funktion **CurrentDb**.

Den Recordset-Variablen weisen wir dann mit der **OpenRecordset**-Methode das Recordset auf Basis der im entsprechenden Kombinationfeld festgelegten Tabelle zu. Schließlich bekommt die **Recordset**-Eigenschaft der vier Unterformulare jeweils das entsprechende Recordset zugewiesen – hier die gekürzte Fassung der geänderten Prozedur:

```
Private Sub UnterformulareAktualisieren()  
    Dim db As DAO.Database  
    Dim rst1 As DAO.Recordset  
    Dim rst2 As DAO.Recordset  
    Dim rst3 As DAO.Recordset  
    Dim rst4 As DAO.Recordset  
    Set db = CurrentDb  
    If Not IsNull(Me.cboTabellen4) Then  
        Set rst4 = db.OpenRecordset(Me!cboTabellen4, _  
            dbOpenDynaset)  
        Me!cboFremdschluesse14.RowSource = _  
            Nz(Me!cboTabellen4)
```

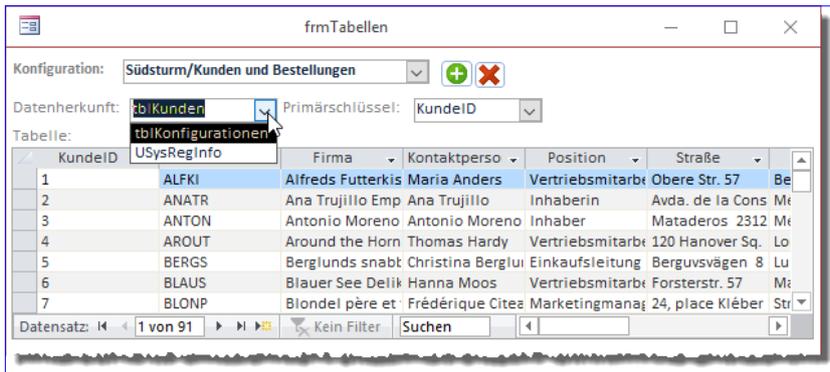


Bild 5: Das Add-In zeigt seine eigenen Tabellen im Kombinationsfeld an.

```

Set Me!sfmTabelle4.Form.Recordset = rst4
UnterformularEinstellen Me!sfmTabelle4.Form
End If

...

End Sub

```

Mit dieser Änderung öffnet sich das Formular zumindest schon einmal fehlerfrei.

Aber warum gelingt dies nun so reibungslos? Greifen wir mit den Recordsets auf Basis des mit **CurrentDb** gelieferten **Database**-Objekts nicht auch auf die Tabellen der Add-In-Datenbank zu? Genau das geschieht nicht:

```

Private Sub Form_Load()
    Dim strBezeichnung As String
    Dim db As DAO.Database
    Dim lngKonfigurationID As Long
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT MSysObjects.Name FROM MSysObjects WHERE (((MSysObjects.Name) Not Like " _
        "'MSys*' And (MSysObjects.Name) Not Like '~*' And (MSysObjects.Name) Not Like 'f_*') AND ((MSysObjects.Type) " _
        "'In (1,4,5,6))) ORDER BY MSysObjects.Name;", dbOpenDynaset)
    Set Me!cboTabelle1.Recordset = rst
    Set Me!cboTabelle2.Recordset = rst
    Set Me!cboTabelle3.Recordset = rst
    Set Me!cboTabelle4.Recordset = rst
    ...
End Sub

```

Listing 1: Füllen der Kombinationsfelder zur Auswahl der Tabellen

CurrentDb bezieht sich immer auf die Host-Datenbank. Wenn Sie per VBA/DAO auf eine Tabelle der Add-In-Datenbank zugreifen wollen, müssen Sie statt **CurrentDb** die Funktion **CodeDb** nutzen. Diese liefert einen Verweis auf das **Database**-Objekt der Add-In-Datenbank.

Weitere Recordsets zuweisen

Allerdings ist noch nicht alles Gold, was glänzt: Das Kombinationsfeld **cboTabelle1** zeigt, wie die übrigen Kombinationsfelder zur Auswahl der Tabellen, nur die Tabellen der Add-In-Datenbank an (s. Bild 5). Also müssen wir wohl auch hier nochmal Hand anlegen und die **RowSource**-Eigenschaft durch entsprechende Recordsets ersetzen.

Bisher hatten wir dort den folgenden Ausdruck als Wert der Eigenschaft **Datensatzherkunft** verwendet:

```

SELECT MSysObjects.Name FROM MSysObjects WHERE (((MSysObjects.Name) Not Like 'MSys*' And (MSysObjects.Name) Not Like '~*' And (MSysObjects.Name) Not Like 'f_*') AND ((MSysObjects.Type) In (1,4,5,6))) ORDER BY MSysObjects.Name;

```

Stücklisten, Teil II

Was lange währt, wird endlich gut: Hier kommt der zweite Teil zu der vor einigen Ausgaben begonnenen Reihe zum Thema Stücklisten. Nachdem das TreeView-Steuer-element zur Anzeige und zum Bearbeiten von Baugruppen und Teilen steht, erweitern wir die Lösung nun um Kontextmenü-Einträge, mit denen Sie Baugruppen und Elemente hinzufügen, bearbeiten und entfernen können.

Kontextmenüs für die Stückliste

Im ersten Teil der Beitragsreihe haben wir bereits ein erstes Kontextmenü mit Untermenüs zur Anwendung hinzugefügt. Dieses wird angezeigt, wenn der Benutzer mit der rechten Maustaste auf den Root-Eintrag des TreeViews mit der Beschriftung **Produkte** klickt. Es erscheint dann ein Kontextmenü mit den beiden Einträgen **Baugruppen** und **Einzelteile**, die wiederum Untermenüs mit den vorhandenen Elementen anzeigen (s. Bild 1).

Nach dem Anklicken einer der Baugruppen beziehungsweise eines der Einzelteile wird dieses direkt unterhalb des Elements **Produkte** angelegt.

Es fehlen allerdings noch einige weitere Kontextmenüs – beispielsweise, um Baugruppen weitere Baugruppen oder Elemente hinzuzufügen, um Baugruppen und Teile zu entfernen oder auch um Baugruppen oder Teile zu markieren, zu kopieren und an anderen Stellen wieder einzufügen.

Produkte, Baugruppen und Einzelteile

Zur Erinnerung an den ersten Teil der Beitragsreihe: Wir unterscheiden prinzipiell zwischen Baugruppen (Tabelle **tblBaugruppen**) und Einzelteilen (**tblEinzelteile**) sowie zwischen Produkten. Letztere sind Baugruppen oder Einzelteile, die im Datensatz ihrer Tabelle mit dem Wert **True** im Feld **IstProdukt** gekennzeichnet sind.

Produkte werden immer direkt unterhalb des Root-Elements **Produkte** angezeigt, alle anderen dort, wo sie zugeordnet wurden. Es kann natürlich auch Baugruppen oder Einzelteile geben, die nur in der Tabelle verfügbar

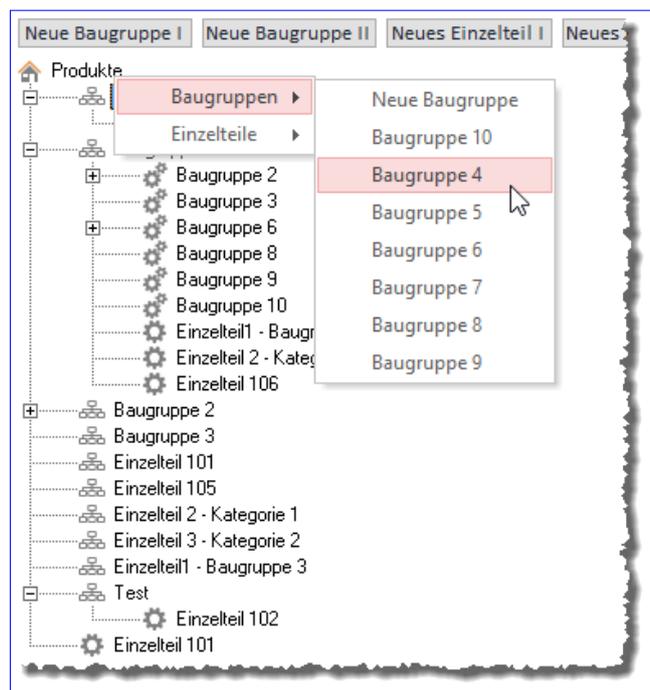


Bild 1: Kontextmenü zum Anlegen von Produkten

sind, aber noch keinem Produkt oder keiner Baugruppe untergeordnet sind und dementsprechend nicht im Tree-View erscheinen.

Baugruppen oder Teile per Kontextmenü entfernen

Die wichtigste Prozedur beim Anzeigen von Kontextmenüs ist die, welche beim Herunterdrücken der rechten Maustaste ausgelöst wird. In unserem Fall sieht diese wie in Listing 1 aus.

Achtung: Möglicherweise kommt Ihnen diese Prozedur bekannt vor, da wir diese im ersten Teil schon einmal

```
Private Sub objTreeView_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, _
    ByVal x As stdole.OLE_XPOS_PIXELS, ByVal y As stdole.OLE_YPOS_PIXELS)
    Dim objNode As MSComctlLib.Node
    Dim strTyp As String
    Dim lngID As Long
    Dim strKey As String
    Dim strKeyItem As String
    Select Case Button
        Case acRightButton
            Set objNode = objTreeView.HitTest(x, y)
            If Not objNode Is Nothing Then
                strKey = objNode.Key
                strKeyItem = Mid(objNode.Key, InStrRev(objNode.Key, "|") + 1)
                strTyp = Left(strKeyItem, 1)
                lngID = Mid(strKeyItem, 2)
                Select Case strTyp
                    Case "r"
                        KontextmenueRoot strKey
                    Case "b"
                        KontextmenueBaugruppe lngID, strKey
                    Case "e"
                        KontextmenueEinzelteil lngID, strKey
                End Select
            End If
        End Select
    End Sub
```

Listing 1: Aufrufen eines Kontextmenü-Eintrags

menüs aufruft. Bevor dies geschieht, ermittelt sie mit der **HitTest**-Methode des **TreeView**-Elements auf welchem Element die rechte Maustaste betätigt wurde. Das Ergebnis landet in der **MSComctlLib.Node**-Variablen **objNode**. Sollte diese anschließend nicht den Wert **Nothing** enthalten, hat der Benutzer tatsächlich ein Element angeklickt.

In diesem Fall ermitteln wir aus der **Key**-Variablen des Elements, der beispielsweise den Wert **r0**, **r0lb18** oder **r0lb18le50** enthalten kann, den Buchstaben des hintersten Elements, hier also **r**, **b** oder **e**, und speichern diesen in der Variablen **strTyp**. Die Zahl

vorgestellt haben. Bei der Weiterentwicklung haben wir jedoch ein paar kleine Optimierungen vorgenommen, welche das Anlegen und Entfernen von Elementen wesentlich erleichtern. Dazu gehört unter anderem, dass wir bei den Aufrufen der Prozeduren **KontextmenueEinzelteil** und **KontextmenueBaugruppe** nun immer auch den **Key**-Wert des aktuellen Elements mitgeben – mehr dazu weiter unten.

Die Prozedur prüft anhand des Parameters **Button**, welche Maustaste der Benutzer heruntergedrückt hat. Handelt es sich um die rechte Maustaste (**acRightButton**), soll das Kontextmenü eingeblendet werden. Unsere Prozedur dient dabei als eine Art Verteiler, die prüfen soll, auf welcher Elementart die rechte Maustaste betätigt wurde, und dann eine entsprechende Prozedur zum Anzeigen des Kontext-

hinter diesem Element, die dem Primärschlüsselwert des entsprechenden Eintrags einer der Tabellen **tblBaugruppen** oder **tblEinzelteile** entspricht, landet hingegen in der Variablen **lngID**. Der Inhalt von **strTyp** wird in der folgenden **Select Case**-Bedingung geprüft. Abhängig vom Wert rufen wir dann eine der Prozeduren **KontextmenueRoot** (**r**, Root-Element), **KontextmenueEinzelteil** (**p**, Produkt-Element), **KontextmenueBaugruppe** (**b**, Bauteil-Element) oder **KontextmenueEinzelteil** (**e**, Einzelteil-Element) auf. Dieser übergeben wir den Primärschlüssel des aktuellen Elements sowie den **Key**-Wert (zum Beispiel **r0**, **r0lb18** oder **r0lb18le50**).

Schauen wir uns erst das Root-Kontextmenü an. Die entsprechende Prozedur heißt **KontextmenueRoot** und sieht wie in Listing 2 aus. Die Prozedur löscht zunächst

```
Private Sub KontextmenueRoot(Optional strKey As String)
    Dim cbr As CommandBar
    On Error Resume Next
    CommandBars("TreeView_Root").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("TreeView_Root", msoBarPopup, False, True)
    With cbr
        KontextmenueBaugruppen cbr, 0, "r0"
        KontextmenueEinzelteile cbr, 0, "r0"
        .ShowPopup
    End With
End Sub
```

Listing 2: Kontextmenü für das Root-Element einblenden

ein eventuell bereits vorhandenes Kontextmenü namens **TreeView_Root** – vorsichtshalber bei deaktivierter Fehlerbehandlung, da der Versuch, ein nicht vorhandenes Kontextmenü zu löschen, einen Fehler auslösen würde. Danach legen wir dieses direkt neu an, und zwar mit der dafür vorgesehenen Methode **Add**. Diese erwartet als Parameter den Namen des Menüs, die Art (da die **Add**-Methode früher auch für das Anlegen von Menüleisten und Symbolleisten verantwortlich war - hier **msoBarPopup** für Kontextmenü) sowie die Angabe, ob es sich um ein temporäres Menü handelt, das mit dem Schließen der Anwendung wieder entsorgt werden kann.

Danach folgt der Aufruf zweier weiterer Routinen, nämlich **KontextmenueBaugruppen** und **KontextmenueEinzelteile**:

```
KontextmenueBaugruppen cbr, 0, "r0"
KontextmenueEinzelteile cbr, 0, "r0"
```

Nach dem Anlegen der Kontextmenü-Elemente für die Baugruppen und die Einzelteile wird das Menü mit der **Popup**-Methode des **CommandBar**-Objekts eingeblendet.

Menüeinträge für die Baugruppen

Die Prozedur **KontextmenueBaugruppen** (s. Listing 3) erwartet drei Parameter – einen Verweis auf das Kontextmenü (**cbr**), die ID des Elements, von dem aus das Kon-

textmenü aufgerufen wurde, sowie den Wert der **Key**-Eigenschaft dieses Elements.

Die Prozedur legt mit der Variablen **cbp** ein Steuerelement des Typs **msoControlPopup** an, also ein Untermenü, und weist diesem den Text **Baugruppen** hinzu. Das Untermenü soll einen Eintrag mit der Bezeichnung **Neue Baugruppe** sowie eine Auflistung der bisher verfügbaren Baugruppen erhalten.

Also legen wir zunächst eine einzelne Schaltfläche an (Typ **msoControlButton**), speichern diese in der Variablen **cbc** und weisen den Text **Neue Baugruppe** sowie für die **OnAction**-Eigenschaft den Wert **=NeueBaugruppe(0)** hinzu. Der Wert **0** für den einzigen Parameter kennzeichnet, dass die Baugruppe in der obersten Ebene, also als Produkt angelegt werden soll – dazu später mehr.

Danach ermittelt die Prozedur die anzuzeigenden Baugruppen. Wenn die Funktion vom Root-Element aus aufgerufen wurde, sollen alle Elemente erscheinen, die noch nicht über das Feld **IstProdukte** als Produkt gekennzeichnet sind und ohnehin schon als Produkt-Elemente angezeigt werden.

Wurde das Kontextmenü von einer Baugruppe aus aufgerufen, erzeugt die Prozedur mit Hilfe der Funktion **AusschliessendeBaugruppen** eine Liste der Baugruppen, die nicht angelegt werden können, da dies einen Zirkelbezug erzeugen würde. Mehr dazu weiter unten. Dies ergibt eine Liste wie **1, 2, 3**, die dann innerhalb der folgenden **If...Then**-Bedingung zu einer **WHERE**-Bedingung hinzugefügt wird. Diese sieht anschließend etwa wie folgt aus:

```
WHERE BaugruppeID NOT IN (1, 2, 3)
```

Danach erstellen wir ein Recordset auf Basis der Tabelle **tblBaugruppen**, in der wir alle Baugruppen erfassen, die

dem aktuellen Element untergeordnet werden können, ohne einen Zirkelbezug auszulösen. Falls der Prozedur mit dem Parameter **strKey** ein Wert wie etwa **r0lb1lb2** übermittelt wurde, soll dieser auch in der Funktion als Parameter enthalten sein, die durch einen der nun anzulegenden Kontextmenübefehle aufgerufen wird. Dazu wandeln wir die Zeichenkette wie **r0lb1lb2** in die Zeichenkette

, **'r0lb1lb2'** um, damit wir diese in der folgenden Schleife leicht als Parameter zu der Funktion hinzufügen können.

In einer **Do While**-Schleife durchlaufen wir nun alle Datensätze dieses Recordsets und legen für jeden Datensatz einen neuen Eintrag im Kontextmenü an. Die Einträge erhalten jeweils den Namen der Baugruppe als Beschriftung

```
Public Sub KontextmenueBaugruppen(cbr As CommandBar, Optional lngBaugruppeParentID As Long, Optional ByVal strKey As String)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim cbp As CommandBarPopup
    Dim cbc As CommandBarButton
    Dim strIN As String
    Set db = CurrentDb
    Set cbp = cbr.Controls.Add(msoControlPopup)
    cbp.Caption = "Baugruppen"
    Set cbc = cbp.Controls.Add(msoControlButton)
    With cbc
        .Caption = "Neue Baugruppe"
        .OnAction = "=NeueBaugruppe(0)"
    End With
    If lngBaugruppeParentID = 0 Then
        strWHERE = " WHERE IstProdukt = FALSE"
    Else
        strWHERE = AuszuschliessendeBaugruppen(lngBaugruppeParentID)
        If Len(strWHERE) > 0 Then
            strWHERE = " WHERE BaugruppeID NOT IN (" & strIN & ")"
        End If
    End If
    Set rst = db.OpenRecordset("SELECT BaugruppeID, Baugruppe FROM tblBaugruppen " & strWHERE _
        & " ORDER BY Baugruppe", dbOpenDynaset)
    If Not Len(strKey) = 0 Then
        strKey = ", '" & strKey & "'"
    End If
    Do While Not rst.EOF
        Set cbc = cbp.Controls.Add(msoControlButton)
        With cbc
            .Caption = rst!Baugruppe
            .OnAction = "=BaugruppeHinzufuegen(" & rst!BaugruppeID & ", " & lngBaugruppeParentID & strKey & ")"
        End With
        rst.MoveNext
    Loop
End Sub
```

Listing 3: Untermenü für die Baugruppen hinzufügen

und einen Ausdruck für die Eigenschaft **OnAction**, der die Funktion **BaugruppeHinzufuegen** mit dem Primärschlüsselwert des aktuellen Datensatzes als Parameter enthält – also beispielsweise folgenden Wert, wenn **strKey** leer war:

```
=BaugruppeHinzufuegen(123, 0)
```

Enthielt **strKey** hingegen einen Wert, lautet der Funktionsaufruf wie folgt:

```
=BaugruppeHinzufuegen(123, 0, 'r0|b1|b2')
```

Das Ergebnis sieht dann etwa wie in Bild 2 aus.

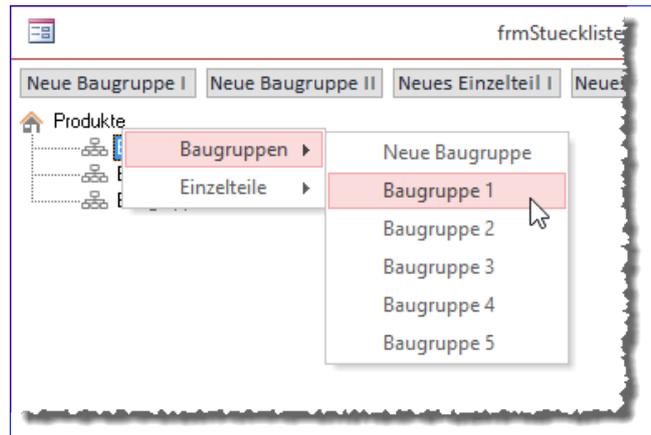


Bild 2: Anzeigen der Baugruppen im Kontextmenü für das Root-Element

```
Public Sub KontextmenueEinzelteile(cbr As CommandBar, Optional lngBaugruppeParentID As Long, Optional ByVal _
    strKey As String)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim cbp As CommandBarPopup
    Dim cbc As CommandBarButton
    Set db = CurrentDb
    Set cbp = cbr.Controls.Add(msoControlPopup)
    cbp.Caption = "Einzelteile"
    Set cbc = cbp.Controls.Add(msoControlButton)
    With cbc
        .Caption = "Neues Einzelteil"
        .OnAction = "=NeuesEinzelteil(0)"
    End With
    Set rst = db.OpenRecordset("SELECT EinzelteilID, Einzelteil FROM tblEinzelteile ORDER BY Einzelteil", _
        dbOpenDynaset)
    If Not Len(strKey) = 0 Then
        strKey = ", '" & strKey & "'"
    End If
    Do While Not rst.EOF
        Set cbc = cbp.Controls.Add(msoControlButton)
        With cbc
            .Caption = rst!Einzelteil
            .OnAction = "=EinzelteilHinzufuegen(" & rst!EinzelteilID & ", " & lngBaugruppeParentID & strKey & ")"
        End With
        rst.MoveNext
    Loop
End Sub
```

Listing 4: Untermenü für die Einzelteile hinzufügen

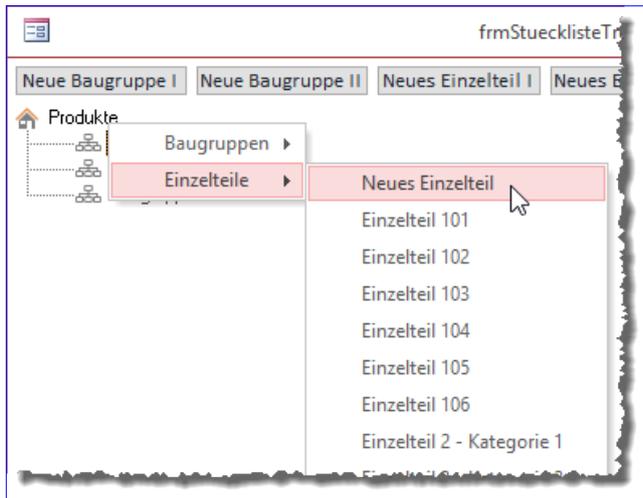


Bild 3: Kontextmenü zum Hinzufügen eines Einzelteils zum Root-Element

Menüeinträge für die Einzelteile

Die zweite von der Prozedur **KontextmenueRoot** aufgerufene Prozedur heißt **KontextmenueEinzelteile** und sieht wie in Listing 4 aus. Diese Prozedur erwartet die gleichen drei Parameter wie die Prozedur **KontextmenueBaugruppen**.

Sie erstellt ein Untermenü mit der Beschriftung **Einzelteile** und speichert dieses in der Variablen **cbp**. Diesem fügt sie nun eine einzelne Schaltfläche hinzu, welche den Text **Neues Einzelteil** und die Funktion **=NeuesEinzelteil(0)** besitzt. Danach erstellt die Prozedur ein Recordset auf Basis der Tabelle **tblEinzelteile** und durchläuft diese Datensätze genau wie bei den Bauteilen.

Dabei erstellt sie für jeden Eintrag eine neue Schaltfläche im Kontextuntermenü und weist diesem als Bezeichnung den Wert des Feldes **Einzelteil** zu. Auch der Wert der Eigenschaft **OnAction** mit der Funktion, die beim Betätigen dieses Kontextmenü-Eintrags ausgelöst werden soll, ist prinzipiell mit den von den Baugruppen bekannten Funktionen identisch – allein der Name der Funktion heißt **EinzelteilHinzufuegen** und nicht **BaugruppeHinzufuegen**. Ein Beispiel für einen solchen Funktionsaufruf sieht so aus:

```
=EinzelteilHinzufuegen(65, 47, 'r0|b46|b47')
```

Das resultierende Kontextmenü sieht wie in Bild 3 aus.

Kontextmenü für eine Baugruppe

Eine Baugruppe kann weitere Baugruppen und Einzelteile enthalten. Das Kontextmenü soll also auch die beiden Einträge zum Hinzufügen von Baugruppen und Einzelteilen enthalten. Außerdem benötigen wir einen Eintrag, mit dem wir die aktuelle Baugruppe aus dem Produkt beziehungsweise der übergeordneten Baugruppe entfernen können.

Es wird schon hier offensichtlich: Wir benötigen Teile des zuvor definierten Kontextmenüs nochmal. Da wir komplett gleiche Code-Segmente nicht mehrfach in verschiedene Prozeduren schreiben wollen, um die Wartbarkeit zu erhöhen, haben wir die mehrfach verwendeten Teile einfach in eigene Routinen ausgegliedert – also beispielsweise in die Prozeduren **KontextmenueBaugruppen** und **KontextmenueEinzelteile**, die wir dann von den Prozeduren, in denen diese eigentlich zum Einsatz kommen, aufrufen.

Kontextmenü für Baugruppen

Soeben haben wir uns angesehen, wie das Kontextmenü für das Root-Element gefüllt wird. Das Kontextmenü für eine Baugruppe sieht nicht wesentlich anders aus. Es soll allerdings einen Eintrag zum Entfernen einer Baugruppe enthalten.

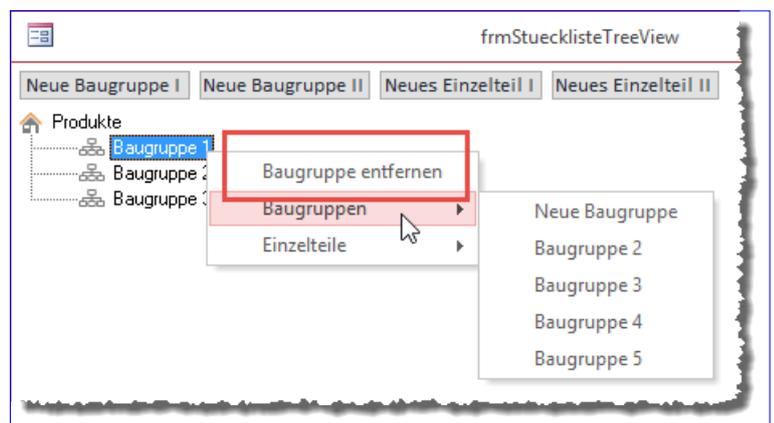


Bild 4: Kontextmenü zum Entfernen einer Baugruppe

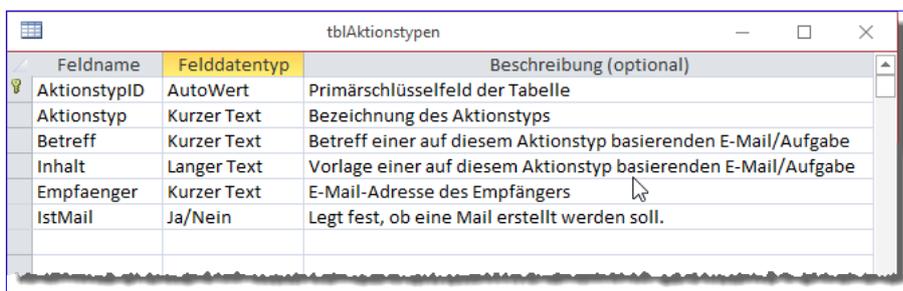
Ticketsystem, Teil IV

In der vorherigen Folge dieser Beitragsreihe haben wir begonnen, das Übersichtsformular für die angelegten Tickets zu entwickeln und ein Detailformular zu erstellen, mit welchem der Ablauf eines Tickets eingesehen werden kann – und das auch zur Abarbeitung der Tickets dienen soll. Im vorliegenden Teil wollen wir diese Formulare und die dafür notwendigen Tabellen weiterentwickeln und die Lösung endlich einsatzbereit machen.

Das Ergebnis der Vorüberlegungen für den Aufbau des Formulars zur detaillierten Anzeige eines Tickets mit den damit verbundenen Aktionen war auch eine Erweiterung des Datenmodells.

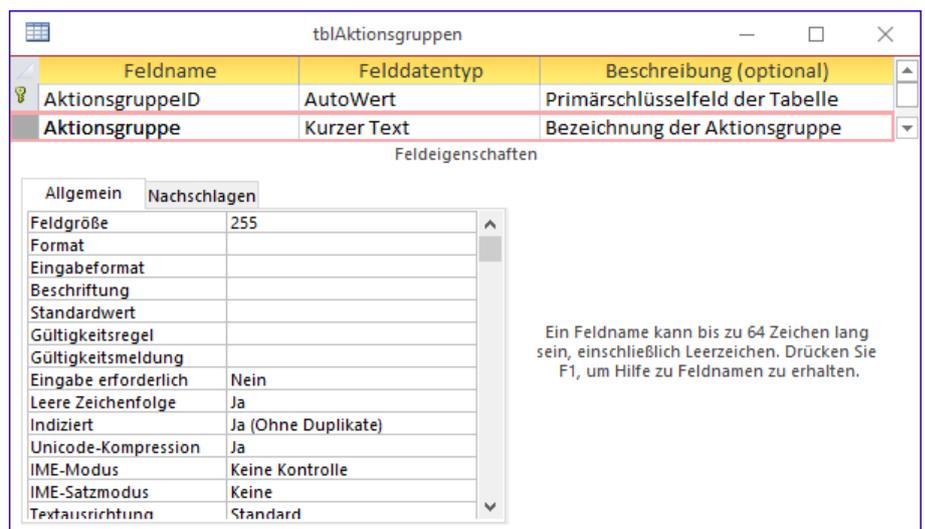
Dabei steht im Vordergrund, wie wir mit einem Ticket umgehen wollen. Im vorherigen Teil haben wir schon angerissen, dass wir als Reaktion auf ein Ticket eine Gruppe von Aktionen vorsehen, wobei die Aktionen selbst auf bestimmten Aktionstypen basieren und für jeden Fall verschiedene Kombinationen von Aktionstypen in einer Tabelle namens **tblAktionsgruppen** zusammengefasst werden.

Noch nicht klar war im vorherigen Teil der Beitragsreihe, welche Felder wir in den verschiedenen Tabellen speichern und wie wir mit diesen arbeiten. Grundsätzlich soll der Ablauf so aussehen:



Feldname	Felddatentyp	Beschreibung (optional)
AktionstypID	AutoWert	Primärschlüsselfeld der Tabelle
Aktionstyp	Kurzer Text	Bezeichnung des Aktionstyps
Betreff	Kurzer Text	Betreff einer auf diesem Aktionstyp basierenden E-Mail/Aufgabe
Inhalt	Langer Text	Vorlage einer auf diesem Aktionstyp basierenden E-Mail/Aufgabe
Empfänger	Kurzer Text	E-Mail-Adresse des Empfängers
IstMail	Ja/Nein	Legt fest, ob eine Mail erstellt werden soll.

Bild 2: Entwurf der Tabelle **tblAktionstypen**



Feldname	Felddatentyp	Beschreibung (optional)
AktionsgruppeID	AutoWert	Primärschlüsselfeld der Tabelle
Aktionsgruppe	Kurzer Text	Bezeichnung der Aktionsgruppe

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 1: Entwurf der Tabelle **tblAktionsgruppen**

- Wir erhalten ein Ticket.
- Wir ermitteln die Gruppe von Aktionen, die als Reaktion auf das Ticket durchgeführt werden sollen. Die Gruppe ist in der Tabelle **tblAktionsgruppen** gespeichert und enthält den Namen der Aktionsgruppe (s. Bild 1).

- Die Tabelle **tblAktionsgruppen** soll mit einem oder mehreren Einträgen einer Tabelle namens **tblAktionstypen** verknüpft sein, die neben der Bezeichnung des Aktionstyps noch einen Betreff und einen Inhalt sowie einen Empfänger für die zu erstellende E-Mail oder Aufgabe enthält (s. Bild 2).

- Die Aktionstypen sind den Aktionsgruppen über die Verknüpfungstabelle **tblAktionsgruppenAktionstypen** zugeordnet (s. Bild 3).
- Die Daten dieser drei Tabellen sehen etwa wie in Bild 4 aus, wobei eine Gruppe zwei Aktionstypen enthält und eine weitere nur einen Aktionstyp.
- Diese Tabellen sind allerdings nur die Grundlage der eigentlichen Bearbeitung des Tickets. Wenn man eine der Aktionsgruppen auswählt, soll nämlich für jeden zugeordneten Aktionstyp eine Mail mit den Daten der drei Felder

Feldname	Felddatentyp	Beschreibung (optional)
AktionstypID	AutoWert	Primärschlüsselfeld der Tabelle
AktionsgruppeID	Zahl	Fremdschlüsselfeld zur Auswahl der Aktionsgruppe
AktionstypID	Zahl	Fremdschlüsselfeld zur Auswahl des Aktionstyps

Indexname	Feldname	Sortierreihenfolge
PrimaryKey	AktionstypID	Aufsteigend
UniqueKey	AktionsgruppeID	Aufsteigend
UniqueKey	AktionstypID	Aufsteigend

Bild 3: Entwurf der Tabelle **tblAktionsgruppenAktionstypen**

Betreff, Inhalt und **Empfänger** erstellt und ein Eintrag zur Tabelle **tblAktionen** hinzugefügt werden. Dabei

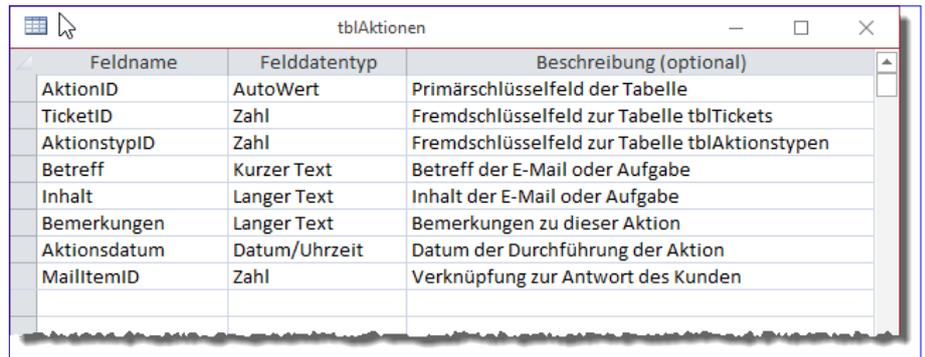
AktionsgruppeID	Aktionsgruppe
1	Neue Rechnung
2	Fehlende Lieferung

AktionstypID	Aktionstyp	MailBetreff	Mailinhalt	Empfänger
1	Mitteilung neue Rechnung an Kunde	[TicketID] Ihre Anfrage bezüglich einer neuen Rechnung	Hallo [Anrede] [Nachname], wir haben Ihre Anfrage erhalten. Sie	[Email]
2	Mitteilung neue Rechnung an Buchhaltung	Bitte um neue Rechnung	Hallo zusammen, bitte stellen Sie eine neue Rechnung entsprechend der	buchhaltung@minhorst.com
3	Mitteilung Versand fehlender Lieferung	[TicketID] Ihre Anfrage bezüglich einer fehlenden Lieferung	Hallo [Anrede] [Nachname], wir bitten vielmals um Entschuldigung	[Email]

Bild 4: Beispieldatensätze der Tabellen **tblAktionstypen**, **tblAktionsgruppen** und **tblAktionsgruppenAktionstypen**

werden gegebenenfalls noch Platzhalter mit kundenspezifischen Daten gefüllt.

- Außerdem soll ein Feld namens **Aktionsdatum** in der Tabelle **tblAktionen** gefüllt werden, in die der Versand der E-Mail oder das Datum der Erledigung der Aufgabe eingetragen wird.



Feldname	Felddatentyp	Beschreibung (optional)
AktionID	AutoWert	Primärschlüsselfeld der Tabelle
TicketID	Zahl	Fremdschlüsselfeld zur Tabelle tblTickets
AktionstypID	Zahl	Fremdschlüsselfeld zur Tabelle tblAktionstypen
Betreff	Kurzer Text	Betreff der E-Mail oder Aufgabe
Inhalt	Langer Text	Inhalt der E-Mail oder Aufgabe
Bemerkungen	Langer Text	Bemerkungen zu dieser Aktion
Aktionsdatum	Datum/Uhrzeit	Datum der Durchführung der Aktion
MailItemID	Zahl	Verknüpfung zur Antwort des Kunden

Bild 5: Entwurf der Tabelle **tblAktionen**

- Erst wenn alle Aktionen zu einem Ticket ein Aktionsdatum enthalten, wird dieses als erledigt markiert.

Beim Erstellen der E-Mails sollen noch die Daten des jeweiligen Kunden herangezogen und in die Platzhalter der Texte für **Betreff**, **Inhalt** und **Empfänger** eingetragen werden. Dann werden die E-Mails angezeigt, damit der Benutzer diese gegebenenfalls noch anpassen kann. Manchmal möchte man vielleicht den Text ändern, den Kunden nicht siezen, sondern duzen oder auch eine Anlage zur E-Mail hinzufügen. Wichtig ist hier, dass erst der tatsächliche Text der E-Mail nach dem Absenden in der Tabelle **tblAktionen** gespeichert wird.

Die Tabelle **tblAktionen** ist wie in Bild 5 aufgebaut.

Einen Überblick über die neu hinzugekommenen Tabellen und ihre Verbindung zur Tabelle **tblTickets** finden Sie in Bild 6. Hier können Sie auch gut erkennen, dass die Tabel-

le **tblAktionstypen** als Verknüpfungstabelle zur Herstellung einer m:n-Beziehung zwischen den Tabellen **tblAktionstypen** und **tblAktionstypen** dient.

Aktionstyp-Sonderfall Kundenantwort

Den Aktionstyp wollen wir mit jeder Aktion speichern, um schnell erkennen zu können, um welchen Aktionstyp es sich handelt. Nun kann es auch sein, dass der Kunde auf eine Mail erneut antwortet. Solche Aktionen sollen auch in der Tabelle **tblAktionen** gespeichert werden. Nur: Welchem Aktionstyp ordnen wir dies zu? Wir legen dazu einen einzigen Aktionstyp an, der sich von den übrigen Aktionstypen unterscheidet. Sein Wert im Feld **Aktionstyp** lautet schlicht **Kundenantwort**, die übrigen Felder bleiben leer.

Bearbeiten von Aktionstypen und Aktionsgruppen

Damit der Benutzer die Aktionstypen und Aktionsgruppen anlegen und bearbeiten kann, legen wir einige Formulare

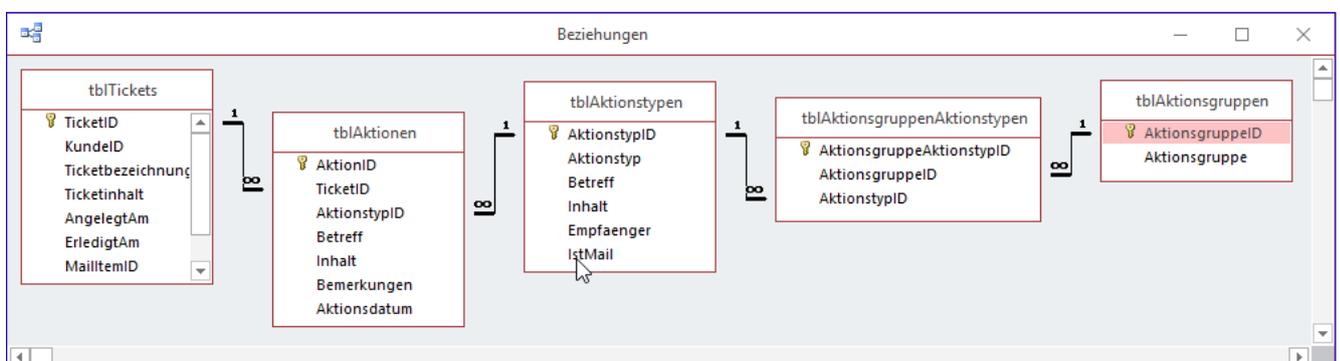


Bild 6: Datenmodell der vorgestellten Tabellen

Bild 7: Eingabe der Daten eines Aktionstyps

Bild 8: Zusammenfassen der Aktionstypen zu einer Aktionsgruppe

an. Das erste heißt **frmAktionstyp** und liefert die Detailansicht eines Aktionstyps zur Bearbeitung (s. Bild 7).

Das Formular **frmAktionsgruppen** zeigt jeweils eine Gruppe und die dazugehörigen Aktionstypen in einem Unterformular an. Über dieses Formular kann man den Namen der Aktionsgruppe festlegen sowie die einzelnen Aktionstypen hinzufügen (s. Bild 8). Mit einem Doppelklick auf eine der Einträge der zugeordneten Aktionstypen wird dieser im Formular **frmAktionstyp** angezeigt. Die Aktionstypen können durch einfache Auswahl aus dem Kombinationsfeld zur Aktionsgruppe hinzugefügt werden.

Erstellen des Formulars frmAktionstyp

Das Formular **frmAktionstyp** sieht im Entwurf wie in Bild 9 aus und verwendet die folgende Abfrage als Datenherkunft:

```
SELECT AktionstypID, Aktionstyp, Betreff, Inhalt,
Empfaenger
FROM tblAktionstypen
WHERE Not Aktionstyp="Kundenantwort"
ORDER BY Aktionstyp;
```

Das entspricht der Tabelle **tblAktionstypen**, die Abfrage fügt jedoch noch eine Sortierung nach dem Inhalt des Feldes **Aktionstyp** hinzu. Außerdem soll der Eintrag mit dem Wert **Kundenantwort** im Feld **Aktionstyp** ausgeschlossen werden, da dieser nicht durch den Benutzer bearbeitet werden darf.

Ein Blättern in den Datensätzen ist nicht vorgesehen, daher stellen wir die Eigenschaften **Navigationschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf **Nein** ein. Oben im Formular finden Sie ein Kombinationsfeld namens **cboAuswahl**, welches die folgende Abfrage als Datensatzherkunft nutzt:

```
SELECT AktionstypID, Aktionstyp
FROM tblAktionstypen
WHERE Not Aktionstyp="Kundenantwort"
ORDER BY Aktionstyp;
```

Bild 9: Entwurf des Formulars frmAktionstyp

```

Private Sub cmdLoeschen_Click()
    If MsgBox("Datensatz wirklich löschen?", vbYesNo, "Datensatz löschen") = vbYes Then
        On Error Resume Next
        DoCmd.SetWarnings False
        RunCommand acCmdDeleteRecord
        DoCmd.SetWarnings True
        Select Case Err.Number
            Case 3200
                MsgBox "Der Eintrag kann nicht gelöscht werden, da er bereits mit einer Aktionsgruppe verknüpft ist."
            Case 0
                Me!cboAuswahl.Requery
                Me.Recordset.MoveFirst
                Me!cboAuswahl = Me!AktionstypID
            Case Else
                MsgBox "Fehler " & Err.Number & vbCrLf & Err.Description
        End Select
        On Error GoTo 0
    End If
End Sub

```

Listing 1: Ereignisprozedur zum Löschen eines Aktionstyps über die Schaltfläche **cmdLoeschen**

Damit das erste Feld mit dem Primärschlüsselfeld ausgeblendet und nur als gebundene Spalte genutzt wird, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein. Auch hier schließen wir den Eintrag **Kundenantwort** für das Feld **Aktionstyp** aus.

Außerdem soll das Kombinationsfeld beim Öffnen des Formulars immer gleich den Wert der Datensatzherkunft anzeigen, der auch im Formular angezeigt wird. Deshalb hinterlegen wir für das Ereignis **Beim Laden** des Formulars die folgende Ereignisprozedur:

```

Private Sub Form_Load()
    Me!cboAuswahl = Me!AktionstypID
End Sub

```

Nach dem Auswählen eines der Einträge des Kombinationsfeldes soll der entsprechende Eintrag auch im Formular angezeigt werden. Dazu fügen wir eine weitere Ereignisprozedur hinzu, die diesmal durch das Ereignis **Nach Aktualisierung** des Kombinationsfeldes ausgelöst wird:

```

Private Sub cboAuswahl_AfterUpdate()
    Me.Recordset.FindFirst "AktionstypID = " & Me!cboAuswahl
End Sub

```

Aktionstyp löschen

Unten im Formular befinden sich zwei Schaltflächen namens **cmdLoeschen** und **cmdNeu**. Mit der Schaltfläche **cmdLoeschen** können Sie logischerweise den aktuellen Eintrag löschen. Die entsprechende Ereignisprozedur finden Sie in Listing 1. Vor dem Löschen fragt die Prozedur per Meldungsfenster ab, ob der Datensatz tatsächlich gelöscht werden soll.

Falls ja, erfolgt dies per **RunCommand accmdDeleteRecord**. Für diese Anweisung haben wir die Fehlerbehandlung deaktiviert, da es sein kann, dass der aktuell angezeigte Aktionstyp bereits über die Tabelle **tblAktionsgruppenAktionstypen** mit einer der Aktionsgruppen verknüpft ist. Wegen der referenziellen Integrität für die Verknüpfung zwischen diesen beiden Tabellen löst der Versuch, einen verknüpften Datensatz zu löschen, den Fehler mit der Nummer **3200** aus. Diesen fangen wir ab

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    Select Case DataErr
        Case 3022
            MsgBox "Es ist bereits ein Aktionstyp mit der Bezeichnung '" & Me.Aktionstyp & "' vorhanden."
            Me!Aktionstyp.SetFocus
            Response = acDataErrContinue
        Case Else
            MsgBox "Fehler " & DataErr & vbCrLf & AccessError(DataErr)
    End Select
End Sub
```

Listing 2: Diese Prozedur wird beispielsweise ausgelöst, wenn der Benutzer einen bereits vorhandenen Aktionstyp eingibt.

und geben in diesem Fall eine entsprechende Meldung aus.

Außerdem haben wir hier die Anzahl von Warnungen deaktiviert, die je nach Einstellung angezeigt werden können. Sollte beim Löschen kein Fehler auftreten, aktualisiert die Prozedur die Datensatzherkunft des Kombinationsfeldes, springt zum ersten Datensatz des Formulars und stellt auch das Kombinationsfeld auf diesen Datensatz ein.

Neuen Aktionstyp anlegen

Die Schaltfläche **cmdNeu** löst die folgende Ereignisprozedur aus:

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

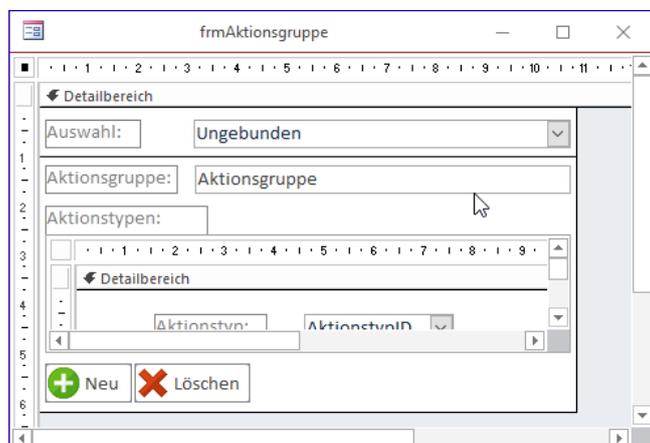


Bild 10: Entwurf des Formulars **frmAktionsgruppe**

Damit wird der Datensatzzeiger auf einen neuen Datensatz verschoben.

Kombinationsfeld aktualisieren

Wenn der Benutzer den aktuell angezeigten Datensatz auf eine andere Weise als durch Auswahl über das Kombinationsfeld **cboAuswahl** festlegt, also beispielsweise durch Neuanlegen eines Datensatzes, muss das Kombinationsfeld natürlich aktualisiert werden. Dies erledigt die folgende Prozedur, die durch das Ereignis **Nach Aktualisierung** des Formulars ausgelöst wird:

```
Private Sub Form_AfterUpdate()
    Me!cboAuswahl.Requery
    Me!cboAuswahl = Me!AktionstypID
End Sub
```

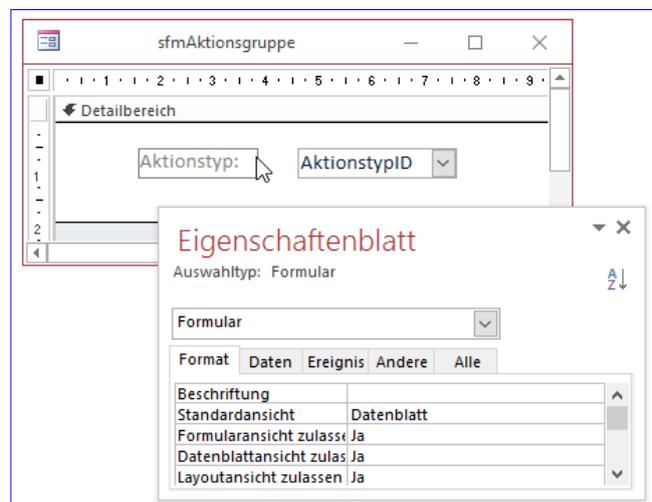


Bild 11: Entwurf des Unterformulars **sfmAktionsgruppe**



Bild 12: Eigenschaften des Unterformular-Steuerelements **sfmAktionsgruppe**

Einen möglichen Fehler haben wir noch außer Acht gelassen: Es kann geschehen, dass der Benutzer beim Anlegen eines neuen Aktionstyps einen Wert für das eindeutig indizierte Feld **Aktionstyp** eingibt, der bereits in einem anderen Datensatz vorhanden ist. In diesem Fall löst dies den Fehler **3022** aus, allerdings können Sie diesen nur im Ereignis **Bei Fehler** des Formulars abfangen. Die dafür benötigte Ereignisprozedur haben wir in Listing 2 abgebildet. Wir prüfen dort in einer **Select Case**-Bedingung den Wert des Parameters **DataErr** mit der Fehlernummer. Im Falle des Wertes **3022** kann es sich nur um das Feld **Aktionstyp** handeln, da wir nur dafür einen eindeutigen Index festgelegt haben. Es erscheint eine entsprechende Meldung und der Fokus wird auf dieses Feld verschoben.

Erstellen des Formulars **frmAktionsgruppe**

Das Formular **frmAktionsgruppen** enthält genau wie das Formular **frmAktionstyp** ein Kombinationsfeld zur Auswahl des aktuellen Datensatzes (s. Bild 10). Die Datenquellen für das Formular sowie das Kombinationsfeld sollen wieder nach dem gleichen Feld sortiert sein. Die Abfrage für die Datenherkunft des Formulars lautet:

```
SELECT AktionsgruppeID, Aktionsgruppe
FROM tblAktionsgruppen
ORDER BY Aktionsgruppe;
```

Das Kombinationsfeld verwendet die gleiche Datenherkunft, da die Tabelle **tblAktionsgruppen** ja ohnehin nur das Primärschlüsselfeld sowie das anzuzeigende Feld **Aktionsgruppe** enthält.

Das Unterformular **sfmAktionsgruppe** verwendet die Tabelle **tblAktionsgruppenAktionstypen** als Datenherkunft. Es zeigt jedoch nur das Nachschlagefeld **AktionstypID** dieser Tabelle an. Seine Eigenschaft **Standardansicht** haben wir auf **Datenblatt** eingestellt (s. Bild 11). Die Datensatzherkunft des Nachschlagefeldes ändern wir auf folgende SQL-Abfrage, damit der Eintrag **Kundenantwort** nicht zur Auswahl angeboten wird:

```
SELECT AktionstypID, Aktionstyp
FROM tblAktionstypen
WHERE Not Aktionstyp="Kundenantwort";
```

Wenn Sie das Unterformular erstellt haben und dieses aus dem Navigationsbereich in den Detailbereich des Hauptformulars ziehen, das bereits mit der Tabelle **tblAktionsgruppen** verknüpft ist, sollten die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** automatisch mit

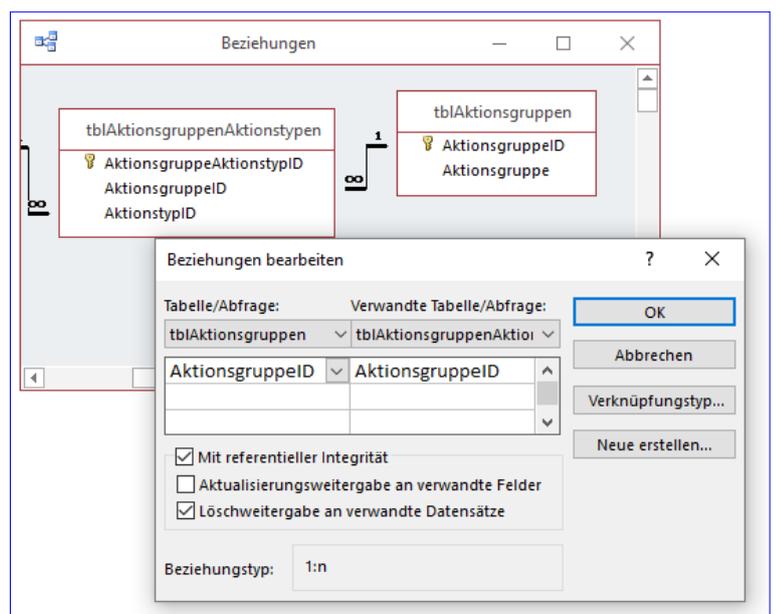


Bild 13: Einstellen der Beziehungseigenschaften

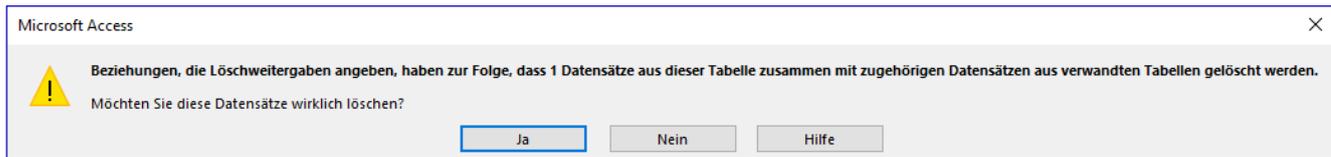


Bild 14: Meldung beim Löschen verknüpfter Datensätze

dem Wert **AktionsgruppeID** gefüllt worden sein (s. Bild 12).

Aktionsgruppe auswählen

Nach der Auswahl eines der Einträge des Kombinationsfeldes **cboAuswahl** wird diese Ereignisprozedur ausgelöst und zeigt den gewählten Datensatz an:

```
Private Sub cboAuswahl_AfterUpdate()
    Me.Recordset.FindFirst "AktionsgruppeID = " & Me!cboAuswahl
End Sub
```

Löschen einer Aktionsgruppe

Wenn der Benutzer eine Aktionsgruppe löschen will, kann er das mit der folgenden Ereignisprozedur erledigen, die durch einen Klick auf die Schaltfläche **cmdLoeschen** ausgelöst wird:

```
Private Sub cmdLoeschen_Click()
    If MsgBox("Eintrag wirklich löschen?", vbYesNo, &
        "Eintrag löschen") = vbYes Then
        DoCmd.SetWarnings False
        RunCommand acCmdDeleteRecord
        DoCmd.SetWarnings True
        Me.Recordset.MoveFirst
        Me!cboAuswahl.Requery
        Me!cboAuswahl = Me!AktionsgruppeID
    End If
End Sub
```

Auch hier tritt allerdings ein Fehler auf, wenn Sie einen Datensatz löschen, der bereits von der Verknüpfungstabelle **tblAktionsgruppenAktionstypen** referenziert wird. In diesem Fall wollen wir dies allerdings nicht verhin-

dern, sondern im Gegenteil die referenzierten Einträge der Tabelle **tblAktionsgruppenAktionstypen** ebenfalls löschen. Dazu nehmen wir eine kleine Änderung an den Beziehungseigenschaften zwischen den beiden Tabellen vor, und zwar im Beziehungen-Fenster (s. Bild 13). Hier aktivieren wir die Option **Löschweitergabe an verwandte Datensätze**. Dadurch werden die verknüpften Datensätze der Tabelle **tblAktionsgruppenAktionstypen** automatisch mitgelöscht.

Damit beim Löschen nicht die lange Meldung aus Bild 14 erscheint, deaktivieren wir vor dem Löschen mit **DoCmd.SetWarnings False** die Warnhinweise. Damit verhindern wir auch, dass noch der Fehler mit der Nummer **2501** folgt, wenn der Benutzer dann verunsichert auf **Nein** klickt. Außerdem soll das Formular nach dem Löschen eines Datensatzes den ersten enthaltenen Datensatz anzeigen und das Kombinationsfeld **cboAuswahl** ebenfalls auf diesen Eintrag einstellen.

Neue Aktionsgruppe anlegen

Genau wie beim Formular **frmAktionstypen** legen Sie mit der Methode **GoToRecord** des **DoCmd**-Objekts über die Schaltfläche **cmdNeu** einen neuen Datensatz an:

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

Nach der Aktualisierung soll immer auch das Kombinationsfeld aktualisiert werden.

```
Private Sub Form_AfterUpdate()
    Me!cboAuswahl = Me!AktionsgruppeID
End Sub
```