HAUFE.
Ausgabe 01/2016

ACCESS

IM UNTERNEHMEN

KONTAKTE VERWALTEN

Zeigen Sie Ihre Kontakte wie unter Outlook als Visitenkarten übersichtlich in Zeilen und Spalten an und erstellen, löschen und bearbeiten Sie diese (ab S. 2).

In diesem Heft:

PROJEKTZEIT-AUSWERTUNG

Werten Sie die mit Outlook erfassten Projektzeiten mithilfe von Access-Berichten aus.

SEITE 35

KANBAN MIT OUTLOOK UND ACCESS

Bringen Sie Übersicht in Ihre aktuell bearbeiteten Projekte und steigen Sie so Ihre Effizienz.

SEITE 52

TIPPS UND TRICKS RUND UM DAS RIBBON

Die Ribbon-Programmierung birgt viele Geheimnisse. Wir liefern praxisnahe Lösungen!

SEITE 65

Projekte im Griff!

Für die Verwaltung von Projekten, Aufgaben und Tätigkeiten gibt es genügend Software. Ob man seine Projekte mit der Unterstützung solcher Hilfsprogramme jedoch effizienter erledigt, hängt maßgeblich davon ab, ob man die Programme auch konsequent nutzt. Je komplizierter dies wird, desto schlechter stehen die Chancen, dass sich die Durchführung tatsächlich verbessert. Wir zeigen eine Alternative zu den gängigen Softwarepaketen auf, die allein Outlook und Access nutzt.

Bereits in der vorherigen Ausgabe haben wir im Beitrag **Projektzeiterfassung mit Outlook und Access** eine Lösung aufgezeigt, mit der Sie Ihre Arbeitszeiten ganz einfach per Drag and Drop in den Terminkalender von Outlook erfassen können, wobei die Daten im Hintergrund in einer Access-Datenbank gespeichert werden.

Dies bohren wir in der vorliegenden Ausgabe nochmals auf – und zwar gleich in zwei Beiträgen. Der erste heißt **Projektzeitauswertung** und zeigt die Access-Seite unserer Projektzeiterfassung (ab S. 35). Während die Daten ja komplett über Outlook erfasst werden, wollen wir die Fähigkeiten von Access nutzen, um diese Daten auszuwerten. Dazu stellt der Beitrag zunächst einen Bericht vor, der die Daten nach Projekten, Aufgaben und Tätigkeiten gruppiert darstellt. Dieser Bericht kann sowohl nach dem Projekt als auch nach einem bestimmten Zeitraum gefiltert werden. Somit erhalten Sie leicht eine Grundlage, um beispielsweise die geleisteten Stunden für ein Projekt zu ermitteln und in Rechnung zu stellen. Zusätzlich stellt der Beitrag ein Formular vor, mit dem Sie die gewünschten Filterkriterien zusammenstellen und den Bericht öffnen können.

Der zweite Beitrag zu diesem Thema heißt **Kanban mit Outlook und Access** erweitert die Art der Eingabe in Outlook (ab S. 52). Dabei nutzt es eine Eigenschaft von Kanban, eines Vorgehensmodells zur Softwareentwicklung. Dort gibt es für die einzelnen Projekte verschiedene Phasen wie **ToDo**, **Entwicklung**, **Testen**, **Veröffentlichen**, **Erledigt**. Um den Überblick zu behalten und nicht zu viele Projekte gleichzeitig zu bearbeiten, soll die Anzahl der Projekte je Phase beschränkt sein – beispielsweise sollte man nicht mehr als zwei Projekte gleichzeitig entwickeln oder testen, da das ständige Hin- und Herspringen sonst zu viel Aufwand kosten würde.

Die Kanban-Erweiterung für die Projektzeiterfassung mit Outlook reichert die Aufgabenliste um Kategorien an, die den einzelnen Projektphasen entsprechen. Neue Aufgaben landen jeweils in einer als Standard festgelegten, ersten Kategorie und werden dann, wenn diese eine Phase durchlaufen haben, per Drag and Drop in die nächste Kategorie/Phase gezogen. Dies erhöht den Überblick über die aktuell bearbeiteten Aufgaben enorm und sorgt für eine effizientere Erledigung.

In weiteren Beiträgen kümmern wir uns beispielsweise darum, Kontaktdaten von Kunden im Stil der Visitenkarten von Outlook in einem Access-Formular darzustellen (**Kontakte verwalten**, ab S. 2). Der Clou ist, dass wir die Visitenkarten nicht nur untereinander, wie in Access-Formularen üblich, sondern in Zeilen und Spalten angeordnet darstellen.

Der Beitrag **Datenblattmarkierung füllen** zeigt ab S. 12, wie Sie größere Bereiche von Unterformularen in der Datenblattansicht mit dem gleichen Wert füllen können. So lässt sich zum Beispiel ein Bereich markieren und etwa mit einem Erledigt-Datum füllen. Normalerweise können Sie nur ein Feld gleichzeitig füllen oder Sie müssen einen Bereich kopieren, um diesen in einen anderen Bereich einzufügen.

Unter dem Titel **Berichtsbereiche, Gruppierungen und Sortierungen** steigen wir ab S. 29 in die Grundlagen der Berichtsgestaltung ein.

Der Beitrag **Rund um das Ribbon** zeigt einige Tipps und Tricks zur Darstellung von Ribbons, die im Kontext mit Formularen eingeblendet oder aktiviert werden sollen (ab S. 65).

Schließlich liefern die beiden Beiträge **Der ACLib-FilterForm-Wizard** (ab S. 24) und **SQL-Text für Filterbedingungen mit Klasse** (ab S. 46) Know-how zum komfortablen Filtern von Formularen.

Und nun: Viel Spaß beim Lesen!

Ihr Michael Forster

Kontakte verwalten

Im Beitrag »Mehrere Datensätze pro Spalte in Formularen« haben wir gezeigt, wie Sie mehrere Unterformulare in einem Formular mit jeweils einem Datensatz füllen, sodass der Eindruck entsteht, die Datensätze seien innerhalb des Formulars in Spalten und Zeilen arrangiert. Im vorliegenden Beitrag zeigen wir ein Beispiel für den praktischen Einsatz dieser Lösung. Dabei wollen wir die Daten einer Kundendatenbank wie in der Kontakte-Ansicht von Outlook als Visitenkarten darstellen – mit der zusätzlichen Option, einen Kontakt direkt von der Übersicht aus löschen zu können.

Im Beitrag **Mehrere Datensätze pro Spalte in Formularen** (**www.access-im-unternehmen.de/1010**) haben wir ja einigen Code in den Klassenmodulen des Haupt- und Unterformulars untergebracht, der teilweise auch noch an die verwendete Datenherkunft (die Tabelle **tblKunden**) angepasst war.

Wenn Sie die dort vorgestellten Techniken in weiteren Formularen nutzen wollen, sollten Sie nicht immer den kompletten Code übertragen und anpassen müssen.

Deshalb haben wir den Code in zwei Klassenmodule ausgelagert. Diese müssen Sie nur noch für die gewünschte Konfiguration konfigurieren und initialisieren, was mit einem einzigen Befehl möglich ist. Damit müssen Sie sich weniger mit der Programmierung auseinandersetzen und können sich etwas mehr auf das Design konzentrieren.

Um eine schöne Kontaktübersicht wie in Outlook zu erstellen, legen Sie dazu zunächst ein Hauptformular an, dem Sie die Steuerelemente zum Blättern in den Kontakten hinzufügen. Diese benennen Sie beispielsweise **cmdNachOben**, **cmdNachUnten** und **cmdNeu**. Außerdem benötigen Sie noch ein Bezeichnungsfeld zur Anzeige der aktuellen Position, das in diesem Fall **lblEintraege** heißen soll.

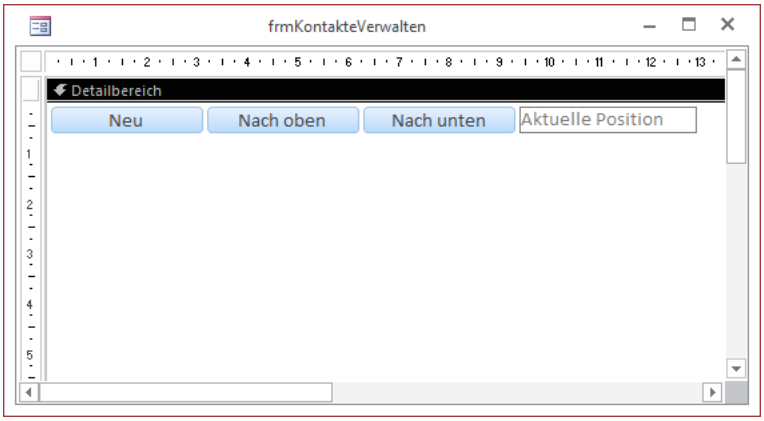


Bild 1: Hauptformular für die Anzeige der Kontakte

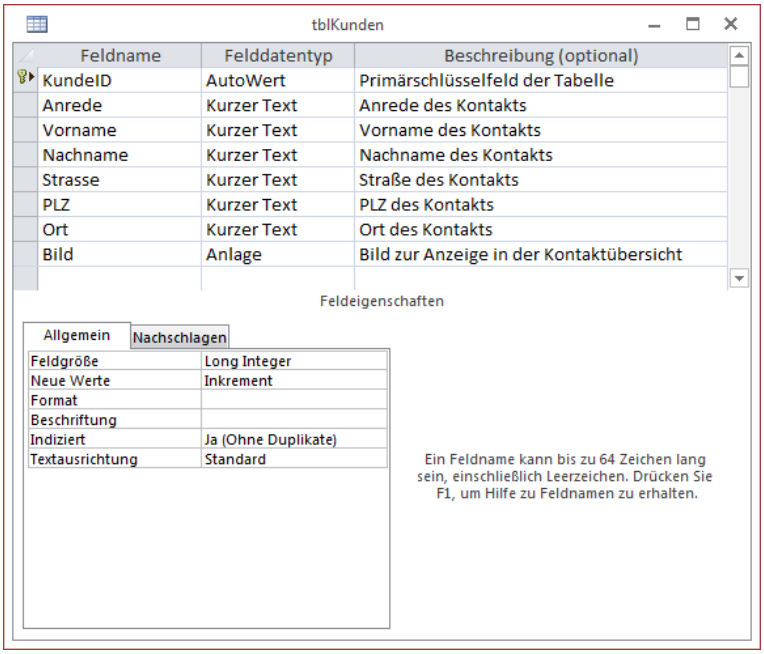


Bild 2: Entwurf der Tabelle mit den anzuzeigenden Kontaktdaten

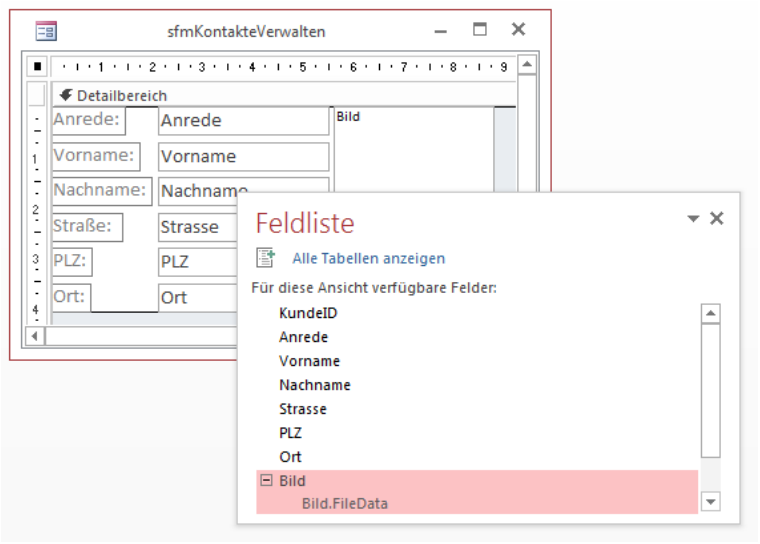


Bild 3: Hinzufügen der Felder zum Unterformular **sfmKontakteVerwalten**

Das Hauptformular sieht demnach zunächst wie in Bild 1 aus und erhält den Namen **frmKontakteVerwalten**. Um die Optik kümmern wir uns später, wir wollen zunächst die Funktionalität sicherstellen.

Bevor wir uns um das Unterformular kümmern, stellen Sie noch die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Bildlaufleisten** und **Trennlinien** auf den Wert **Nein** ein. Diese Elemente benötigen wir nicht, da das Hauptformular ja komplett ungebunden ist.

Unterformular erstellen

Im zweiten Schritt legen Sie ein Unterformular namens **sfmKontakteVerwalten** an, dem Sie – nur zum Hinzufügen der entsprechenden Steuerelemente – die zu verwendende Datenherkunft zuweisen. In diesem Fall verwenden wir eine Tabelle namens **tblKunden**, die im Entwurf wie in Bild 2 aussieht.

Fügen Sie dann die Felder der Tabelle hinzu und ordnen Sie diese wie in Bild 3 an. Da nur ein Datensatz je Unterformular angezeigt werden soll, benötigen wir die Elemente zum Navigieren innerhalb der Datensätze des Unterformulars nicht. Stellen Sie daher die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Bildlaufleisten** und **Trennlinien** auf den Wert **Nein** ein. Außerdem legen Sie den Wert der Eigenschaft **Zyklus** auf **Aktueller Datensatz** fest, damit der Benutzer den Datensatz beim Navigieren mit der **Tab**-Taste nicht wechseln kann.

Außerdem fügen Sie noch eine Schaltfläche namens **cmdLoeschen** hinzu, die Sie, genau wie die Schaltflächen im Hauptformular, nicht mit einer Ereignisprozedur zu versehen brauchen. Das Unterformular sieht dann in der Formularansicht wie in Bild 4 aus.

Wichtig ist hier, dass Sie nur wenig Platz zwischen den Steuerelementen und dem Seitenrand lassen. Nur wenn die Rahmenlinien der enthaltenen Steuerelemente angezeigt werden sollen, lassen Sie einen Abstand von **1** zwischen den Steuerelementen und den Seitenrändern des Detailbereichs des Unterformulars. Abstände fügen wir später durch die Positionierung der Unterformulare im Hauptformular hinzu.

Unterformulare-Steuerelemente einfügen

Nun müssen Sie nur noch Unterformular-Steuerelemente in der gewünschten Anzahl zum Hauptformular hinzufügen. Die Betonung liegt auf Unterformular-**Steuerelemente**, denn wir wollen diese Steuerelemente erst zur Laufzeit mit dem Verweis auf das Unterformular selbst füllen. Um

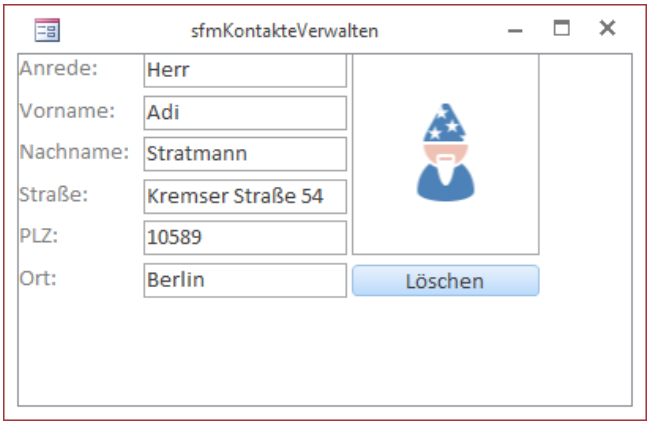


Bild 4: Das Unterformular in der Formularansicht

die Größe optimal einzustellen, sollten Sie jedoch fürs Erste einfach das komplette Unterformular zum Hauptformular hinzufügen. Dies erledigen Sie am einfachsten, indem Sie das Hauptformular in der Entwurfsansicht öffnen und das Unterformular aus dem Navigationsbereich an die gewünschte Stelle im Entwurf ziehen.

Bild 5: Erster Test des Unterformulars im Hauptformular

Hier entfernen Sie zunächst das Bezeichnungsfeld des Unterformular-Steuerelements. Danach wechseln Sie in die Formularansicht des Hauptformulars, um zu prüfen, ob alle Elemente des Unterformulars wie gewünscht angezeigt werden. Die Ansicht aus Bild 5 macht uns darauf aufmerksam, dass die ständige Wiederholung der Bezeichnungsfelder für jeden Datensatz wohl wenig Sinn macht und eine Menge Platz benötigt – also gehen wir nochmals in den Entwurf des Unterformulars und entfernen die Bezeichnungsfelder.

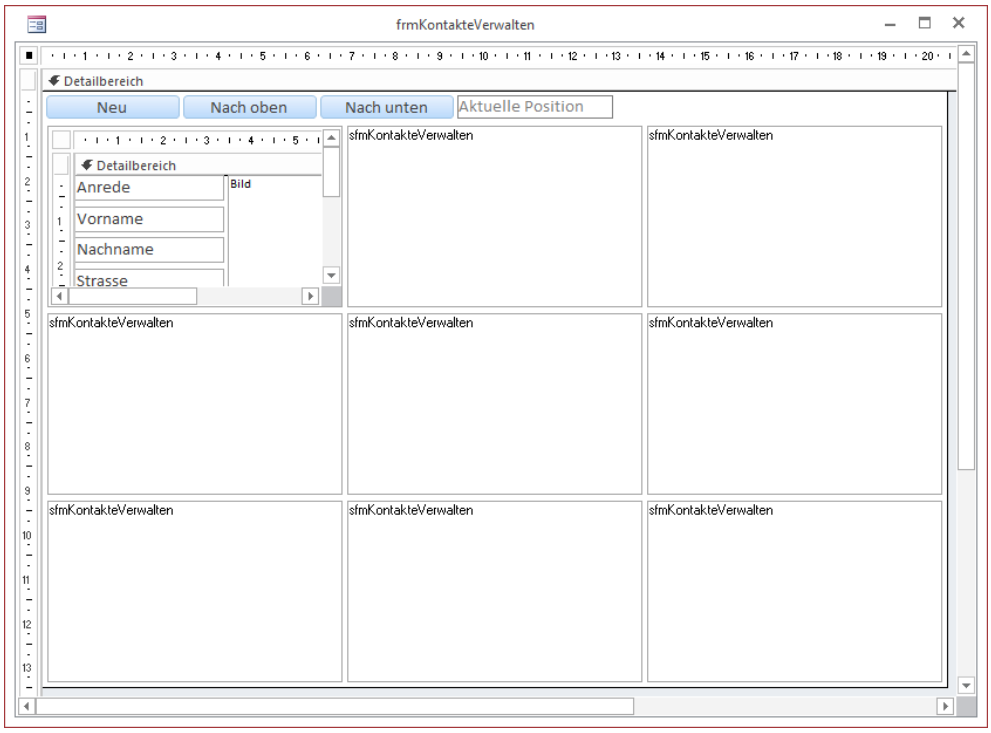


Bild 6: Fertig angeordnete Unterformulare

Nachdem dies erledigt ist, ziehen wir das Unterformular erneut in das Hauptformular und entfernen wieder das Bezeichnungsfeld des Unterformular-Steuerelements. Nun passt alles!

Sie können nun das Unterformular mit **Strg + C** kopieren und **Strg + V** so oft in das Formular einfügen, wie Sie es wünschen. Ordnen Sie die Unterformulare dabei so wie in Bild 6 an und lassen Sie etwas Platz dazwischen.

Stellen Sie die Eigenschaft **Rahmenart** des Unterformular-Steuerelements nach Wunsch auf **Transparent** ein, um die Rahmen um jeden Kontakt zu entfernen.

Wenn Sie nun in die Formularansicht wechseln, erhalten Sie die Ansicht aus Bild 7. Alle Unterformulare zeigen den gleichen Datensatz an – und zwar den ersten, den die Datenherkunft liefert.

Das ist auch logisch, denn wir haben dem Formular ja noch keinerlei Logik hinzugefügt. Bevor wir dies tun, ist auch noch ein weiterer Schritt nötig: Damit die Programmlogik die Unterformulare mit den gewünschten Daten füllen kann, müssen diese noch entsprechende Namen erhalten.

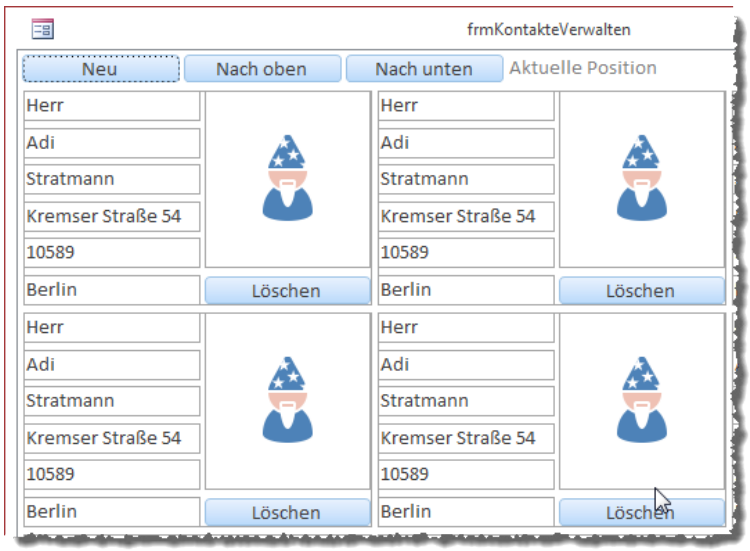


Bild 7: Alle Unterformulare zeigen den gleichen Datensatz an.

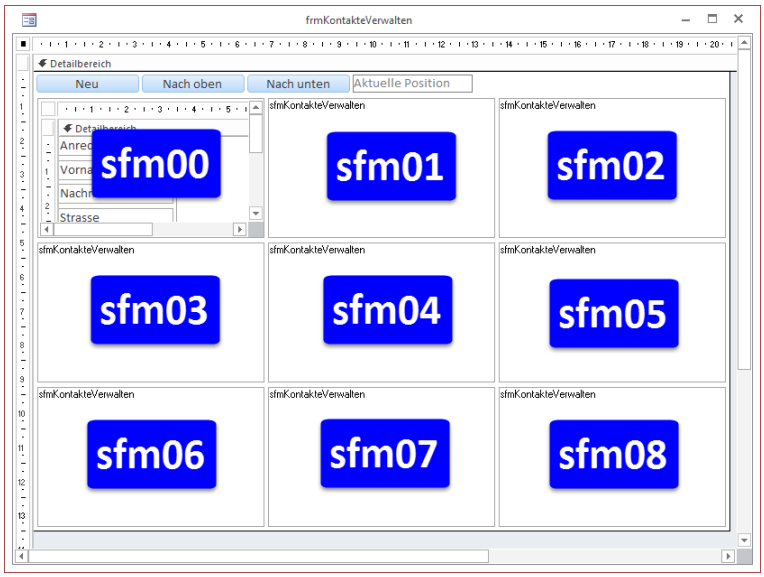


Bild 8: Namen der Unterformular-Steuerelemente

Daher benennen Sie diese von links nach rechts und dann von oben nach unten mit Namen nach dem Schema **sfmXX**, wobei **XX** bei **00** beginnt und dann jeweils um eins erhöht wird. Bild 8 zeigt die Namen der Unterformular-Steuerelemente für eine Konfiguration mit drei mal drei Unterformularen.

Unterformulare füllen

Damit kommen wir zum Einsatz der beiden Klassen **clsDatenNebeneinander** und **clsDatenNebeneinanderUnterformular**. Fügen Sie dem Formular **frmKontakteVerwalten** eine Prozedur hinzu, die durch das Ereignis **Beim Laden** ausgelöst wird. Diese ergänzen Sie um eine der Anweisungen aus Listing 1.

Hier deklarieren wir zunächst eine Objektvariable des Typs **clsDatenNebeneinander** im Kopf des Klassenmoduls. Dies bewirkt, dass das Objekt auch nach dem Beenden der **Form_Load**-Prozedur, welche die Variable füllt, noch vorhanden ist. Die **Form_Load**-Prozedur erstellt ein neues Objekt auf Basis der Klasse **clsDatenNebeneinander** und speichert den Verweis darauf in der Variablen **objDatenNebeneinander**. Nun fehlt nur noch ein Schritt, nämlich das Übergeben der Daten für die Funktion dieser Klasse. Dies erfolgt mit der Methode **Initialisieren**, welche die folgenden Parameter erwartet:

```
Dim objDatenNebeneinander As clsDatenNebeneinander

Private Sub Form_Load()
    Set objDatenNebeneinander = New clsDatenNebeneinander
    With objDatenNebeneinander
        .Initialisieren Me, Me!cmdNachUnten, Me!cmdNachOben, Me!cmdNeu, Me!lblEintraege, "sfmKontakteVerwalten", _
            "tblKunden", "KundeID", "cmdLoeschen"
    End With
End Sub
```

Listing 1: Initialisieren der Unterformulare und der Funktionen zum Blättern, Anlegen und Löschen

- **frm**: Verweis auf das aufrufende Formular, einfach per **Me** zu übergeben

- **cmdNachUnten**: Verweis auf die Schaltfläche, mit der eine Seite nach unten geblättert werden soll (im Beispiel **cmdNachUnten**)

- **cmdNachOben**: Verweis auf die Schaltfläche, mit der eine Seite nach oben geblättert werden soll (hier **cmdNachOben**)

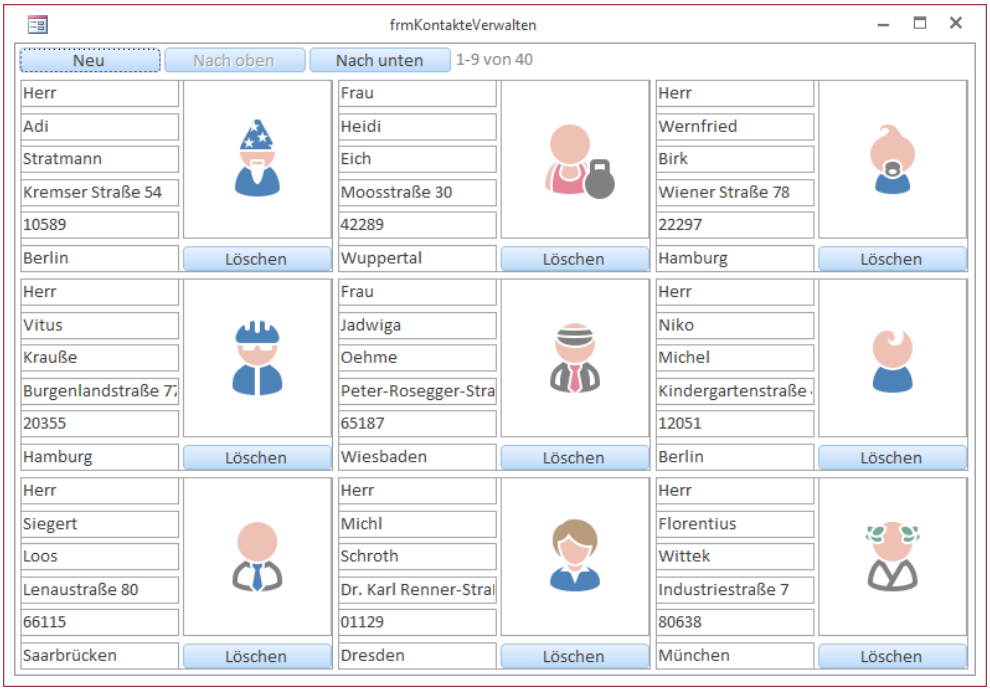


Bild 9: Formular mit gefüllten Unterformularen

- **cmdNeu**: Schaltfläche zum Anlegen eines neuen Datensatzes (hier **cmdNeu**)

- **lblEintraege**: Bezeichnungsfeld, das die aktuelle Position anzeigen soll

- **strSubform**: Name des Unterformulars, das in den Unterformular-Steuerelementen angezeigt werden soll, hier **sfmKontakteVerwalten**

- **strRecordsource**: Datenherkunft für die Unterformulare, hier **tblKunden**

- **strPrimaerschluessel**: Name des Primärschlüsselfeldes der Datenherkunft, hier **KundeID**

- **strLoeschen**: Name der Löschen-Schaltfläche im Unterformular, hier **cmdLoeschen**

Öffnen Sie das Formular nun in der Formularansicht, erhalten Sie die gewünschten Daten wie in Bild 9. Das Vor- und Zurückblättern gelingt ebenso wie das Anlegen neuer und das Löschen vorhandener Datensätze.

Technische Hintergründe

Es stellt sich nun zum Beispiel noch die Frage, warum der Aufruf der Methode **Initialisieren** keine Angabe der Anzahl der zu füllenden Unterformulare enthält.

Der Grund ist einfach: Wenn Sie die Unterformular-Steuerelemente nach den Vorgaben benennen, also mit **sfm00**, **sfm01** und so weiter, durchläuft die Klasse diese einfach und füllt alle vorhandenen Unterformulare.

Erst, wenn es das nächste Unterformular-Steuerelement in der Reihenfolge nicht mehr findet, geht es davon aus, dass es alle Unterformulare gefüllt hat.

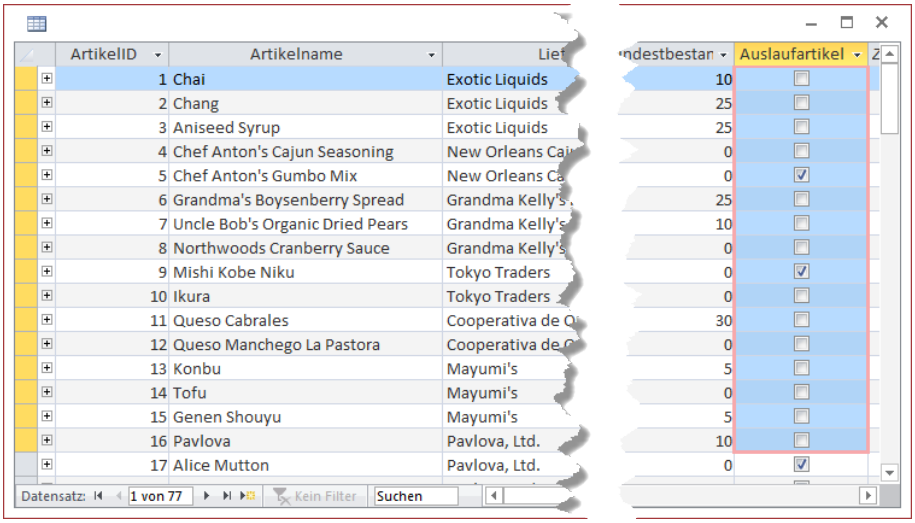
Gegenüber der Basisversion, die wir im Beitrag **Mehrere Datensätze pro Spalte in Formularen** (**www.access-im-unternehmen.de/1010**) vorgestellt haben, mussten wir natürlich einige Elemente umbauen – immerhin sollte ja die komplette Funktionalität, die dort in Haupt- und Unterformular steckte, nun in zwei Klassen ausgelagert werden, die nur noch per **Initialisieren**-Methode aufgerufen werden sollen.

Datenblattmarkierung füllen

In der Datenblattansicht lassen sich ja bereits eine Menge Dinge erledigen – Daten einfügen, löschen, bearbeiten, kopieren, ausschneiden ... Sie können sogar komplette Bereiche kopieren und in andere Bereiche einfügen, sofern diese Bereiche zueinander kompatibel sind, und Access ist hier recht tolerant. Was aber fehlt, ist die Markierung eines Zielbereichs, dessen Felder dann alle mit dem gleichen Wert gefüllt werden. Wenn Sie also etwa für alle Datensätze ein Ja/Nein-Feld anhaken möchten, müssen Sie dies immer noch manuell erledigen. Dieser Beitrag zeigt eine passende Lösung für Datenblätter in Formularen und Unterformularen.

Wenn Sie einen Bereich eines Formulars oder Unterformulars in der Datenblattansicht mit den gleichen Werten füllen wollten, waren Sie bislang auf zwei Möglichkeiten angewiesen:

- das Absetzen einer entsprechenden UPDATE-SQL-Aktionsabfrage, was aber den Nachteil hat, dass dies selbst die Fähigkeiten von Powerusern mitunter überschreitet, oder
- das manuelle Füllen der Felder mit dem gewünschten Wert.



ArtikelID	Artikelname	Lief	ndestbestan	Auslaufartikel
1	Chai	Exotic Liquids	10	☐
2	Chang	Exotic Liquids	25	☐
3	Aniseed Syrup	Exotic Liquids	25	☐
4	Chef Anton's Cajun Seasoning	New Orleans Cai	0	☐
5	Chef Anton's Gumbo Mix	New Orleans Ca	0	☑
6	Grandma's Boysenberry Spread	Grandma Kelly's	25	☐
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's	10	☐
8	Northwoods Cranberry Sauce	Grandma Kelly's	0	☐
9	Mishi Kobe Niku	Tokyo Traders	0	☑
10	Ikura	Tokyo Traders	0	☐
11	Queso Cabrales	Cooperativa de Q	30	☐
12	Queso Manchego La Pastora	Cooperativa de Q	0	☐
13	Konbu	Mayumi's	5	☐
14	Tofu	Mayumi's	0	☐
15	Genen Shouyu	Mayumi's	5	☐
16	Pavlova	Pavlova, Ltd.	10	☐
17	Alice Mutton	Pavlova, Ltd.	0	☑

Datensatz: 1 von 77 | Kein Filter | Suchen

Bild 1: Das wäre schön: Die markierten Felder alle mit einem Schlag beispielsweise aus- oder abwählen

Letzteres geschieht für eine bessere Effizienz am besten so, dass man eine gewisse Menge Felder mit dem Wert füllt (sagen wir zehn untereinander liegende Felder), dann diesen Bereich kopiert und in eine Markierung unter den gefüllten Feldern mit einer Spalte Breite und zehn Zeilen Höhe wieder einfügt. Wenn man viele Felder füllen muss, kann man nochmal größere Bereiche markieren und kopieren und dann in entsprechend größere Bereiche einfügen.

Der Nachteil: Man müsste immer ungefähr die gewünschte Menge Felder ermitteln, damit alle kopierten Werte dort hineinpassen. Der Vorteil: Access ist beim Einfügen nicht pingelig, die Größe von Quell- und Zielfeldern muss nicht unbedingt übereinstimmen. Das heißt, wenn Sie beispielsweise zehn Felder in einen Bereich von weniger als zehn Feldern einfügen, dann werden nur die markierten Felder gefüllt.

Wenn der Zielbereich größer ist als der Quellbereich, dann werden ebenfalls nur die entsprechend dem kopierten Bereich gefüllt.

Befriedigend ist das alles nicht: Fehler können so leicht auftreten, sei es durch das Auslassen eines zu füllenden Feldes oder durch Überschreiben von Bereichen, die gar nicht geändert werden sollen. Es wäre also am einfach-

sten, wenn man etwa die in Bild 1 markierten Felder mit einem Klick etwa aktivieren oder deaktivieren oder mit einem Wert füllen könnte.

Die Lösung

Ein viel besserer Ansatz wäre etwa ein Kontextmenü-Eintrag, den man für eine Markierung in der Datenblattansicht auswählen kann und der dann eine **InputBox** öffnet, der man den einzufügenden Wert übergibt. Nach Betätigen der **OK**-Schaltfläche soll dieser Wert in alle markierten Zellen eingefügt werden. Dabei soll es keine Rolle spielen, wie groß die Markierung ist – es sollen also sowohl mehrere Spalten als auch mehrere Zeilen markierbar sein, auch in Kombination miteinander. Noch schöner wäre es, wenn ein in der Zwischenablage befindlicher Wert gleich als Standardwert in der **InputBox** angezeigt würde.

Dies gilt es nun zu realisieren – machen wir uns ans Werk!

Einsatz der fertigen Lösung

Die fertige Lösung wird in Form einer Klasse kommen, die alle notwendigen Funktionen liefert. Die Klasse heißt **clsDatasheetInsert** und sollte in dem Hauptformular, welches das in der Datenblattansicht erscheinende Unterformular enthält, mit der Objektvariablen **objDatasheetInsert** deklariert werden:

```
Dim objDatasheetInsert As clsDatasheetInsert
```

In die Prozedur, die durch das Ereignis **Beim Laden** des Hauptformulars ausgelöst wird, fügen wir die folgenden Anweisungen ein:

```
Private Sub Form_Load()
    Set objDatasheetInsert = New clsDatasheetInsert
    With objDatasheetInsert
        Set .Form = Me!sfmArtikel.Form
    End With
End Sub
```

Die erste Anweisung instanziert das neue Objekt und speichert einen Verweis darauf in der Variablen **objDatasheetInsert**. Die Anweisung innerhalb des **With**-Konstrukts weist der Eigenschaft **Form** des Objekts einen Verweis auf das Unterformular in der Datenblattansicht zu, das mit den Funktionen zum Füllen mehrerer Felder gleichzeitig versehen werden soll.

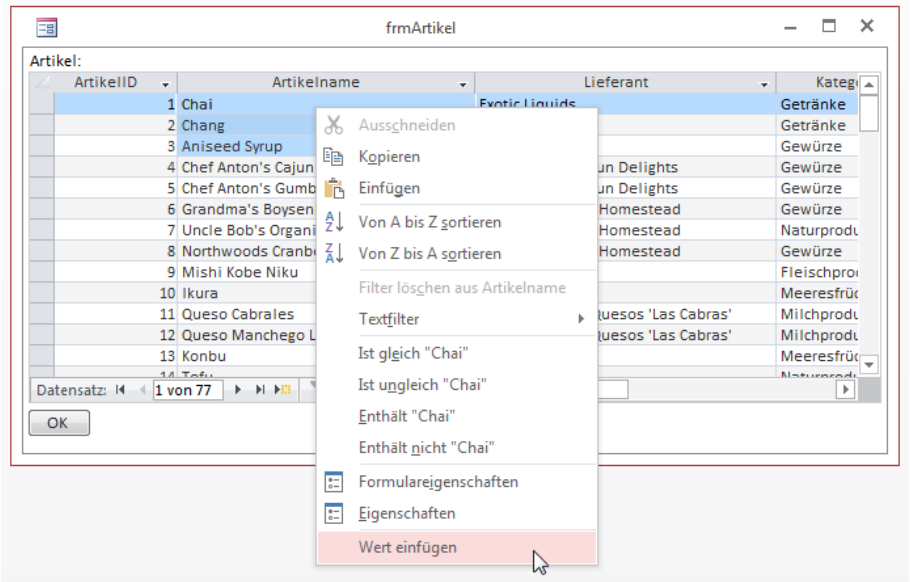


Bild 2: Einfügen von Werten per Kontextmenü

Danach können Sie dann wie in Bild 2 die Zielfelder markieren, mit der rechten Maustaste das Kontextmenü aufrufen und dann den Eintrag **Wert einfügen** auswählen, um die gewünschten Daten in die markierten Felder einzufügen.

Kontextmenü-Eintrag hinzufügen

Für diese Funktion benötigen wir zunächst einmal einen Kontextmenü-Eintrag – und zwar nicht in irgendeinem Kontextmenü, sondern genau in dem, das beim

Rechtsklick auf eines der Felder oder auf mehrere markierte Felder erscheint.

Für einen groben Überblick, um welches Kontextmenü es sich handeln könnte, schauen wir uns die Liste aller **Commandbar**-Objekte an, die den Typ **msoBarTypePopup** aufweisen. Das erledigt die folgende Prozedur, welche die Namen der Kontextmenüs im Direktbereich ausgibt (für diese und weitere Prozeduren benötigen Sie einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** – s. Bild 3):

```
Public Sub CommandbarsAusgeben()
    Dim cbr As Office.CommandBar
    For Each cbr In CommandBars
        If cbr.Type = msoBarTypePopup Then
            Debug.Print cbr.Name
        End If
    Next cbr
End Sub
```

Hier erscheinen gleich zu Beginn die folgenden beiden Einträge:

- **Form Datasheet**
- **Form Datasheet Cell**

Nun müssen wir nur noch herausfinden, welches der beiden Kontextmenüs beim Anklicken eines Feldes im Datenblatt erscheint. Dazu legen wir einfach in allen betroffenen Kontextmenüs einen neuen Eintrag an, der den Namen des Kontextmenüs trägt:

```
Public Sub CommandbarsTesten()
    Dim cbr As Office.CommandBar
    Dim cbb As Office.CommandBarButton
    Set cbr = CommandBars.Item("Form Datasheet Cell")
    Set cbb = cbr.Controls.Add(msoControlButton, , , , True)
    cbb.Caption = "Form Datasheet Cell"
    Set cbr = CommandBars.Item("Form Datasheet")
```

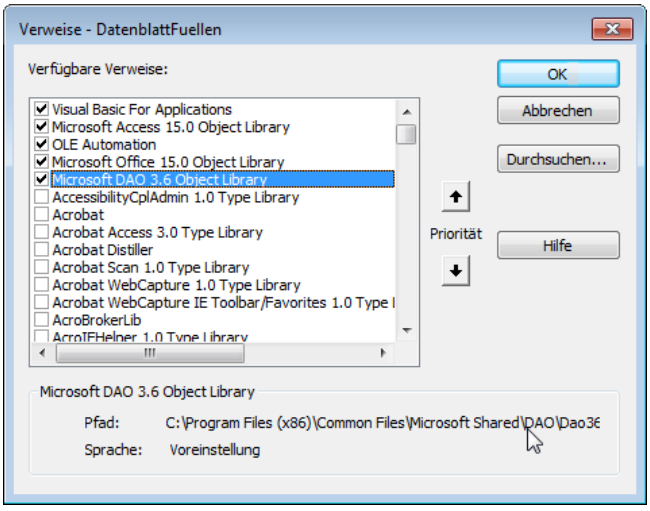


Bild 3: Verweis auf die Office-Bibliothek

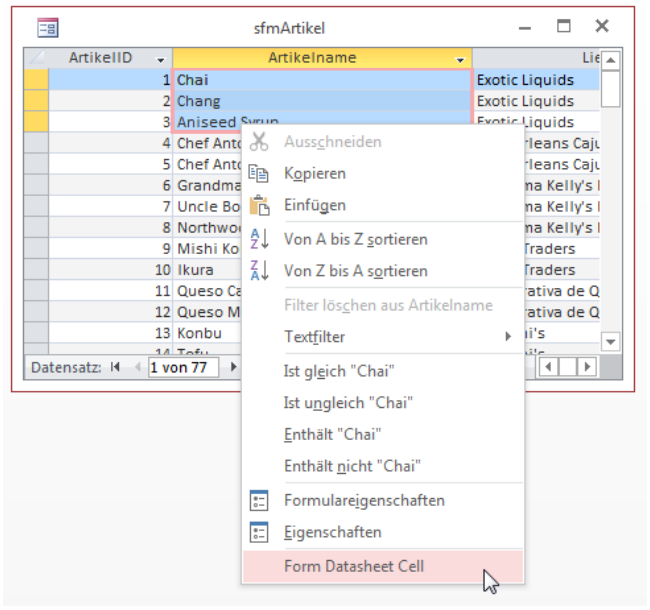


Bild 4: Ermitteln des Kontextmenüs für das Datenblatt

```
    Set cbb = cbr.Controls.Add(msoControlButton, , , , True)
    cbb.Caption = "Form Datasheet"
End Sub
```

Die **Set cbr ...**-Anweisung referenziert jeweils eines der betroffenen Kontextmenüs, die **Set cbb ...**-Anweisung fügt ein neues Steuerelement des Typs **msoControlButton** hinzu. Über die **Caption**-Eigenschaft legt die Prozedur die Beschriftung des jeweiligen Eintrags fest.

```
Public Property Set Form(frm As Form)
    Dim cbr As Office.CommandBar
    Dim i As Integer
    Set m_Form = frm
    Set cbr = CommandBars.Item("Form Datasheet Cell")
    For i = cbr.Controls.Count To 1 Step -1
        If cbr.Controls.Item(i).Caption = "Wert einfügen" Then
            cbr.Controls.Item(i).Delete
        End If
    Next i
    Set cbbInsert = cbr.Controls.Add(msoControlButton, , , , True)
    cbbInsert.Caption = "Wert einfügen"
End Property
```

Listing 1: Einrichten des Kontextmenüs

Danach klicken wir einfach mit der rechten Maustaste auf das gewünschte Element im Datenblatt und erhalten ganz unten die Antwort auf unsere Frage: Das Kontextmenü **Form Datasheet Cell** ist unser gesuchtes Objekt (s. Bild 4).

Werte einfügen per Klasse

Weiter oben haben wir ja bereits erläutert, dass wir die Funktionalität zum Einfügen von Daten in mehrere Felder gleichzeitig in einer Klasse kapseln möchten. Das hat den Vorteil, dass Sie den Code ganz einfach in mehreren Formularen wiederverwenden können.

Diese Klasse heißt **clsDatasheetFill** und enthält eine **Property Set**-Prozedur, die einen Verweis auf das zu unterstützende Formular erwartet. Diesen Verweis speichert die Klasse in der wie folgt im Klassenmodul deklarierten Variablen:

```
Dim m_Form As Form
```

Die Prozedur selbst finden Sie in Listing 1. Sie trägt zuerst den Verweis auf das per Parameter übergebene Formular-Objekt in die Variable **m_Form** ein. Dann sorgt sie noch für die Erweiterung des Kontextmenüs um einen Eintrag mit dem Text **Wert einfügen**. Dazu füllt sie die Variable **cbr** mit einem Verweis auf das Element **Form Datasheet Cell** der **CommandBars**-Auflistung.

Anschließend durchläuft sie alle Einträge dieses Kontextmenüs und prüft diese auf die Beschriftung **Wert einfügen**. Sind bereits Einträge mit dieser Beschriftung vorhanden, entfernt die Prozedur diese mit der **Delete**-Methode des jeweiligen Elements. Danach legt sie diesen Eintrag neu an und speichert einen Verweis darauf in der Variablen **cbbInsert**. Die folgende Anweisung stellt nur noch die Beschriftung auf **Wert einfügen** ein.

Die Variable **cbbInsert** wurde noch nicht deklariert. Da wir für den neu hinzugefügten Eintrag zum Kontextmenü auch eine Ereignisprozedur hinterlegen möchten, die beim Anklicken des Eintrags ausgelöst wird, benötigen wir noch eine entsprechende, mit dem Schlüsselwort **WithEvents** ausgestattete Variable.

Auch diese landet wieder im Kopf des Klassenmoduls:

```
Dim WithEvents cbbInsert As CommandBarButton
```

Wie legen Sie nun die benötigte Ereignisprozedur an? Dazu wählen Sie einfach im Klassenmodul im linken Kombinationsfeld den Eintrag **cbbInsert** aus, der dort nur deshalb erscheint, weil Sie die Objektvariable mit **WithEvents** deklariert haben. Nun erscheint im rechten Kombinationsfeld das Standardereignis **Click** (im Übrigen auch das einzige Ereignis dieses Objekts) und der VBA-Editor legt die passende Ereignisprozedur an (s. Bild 5).

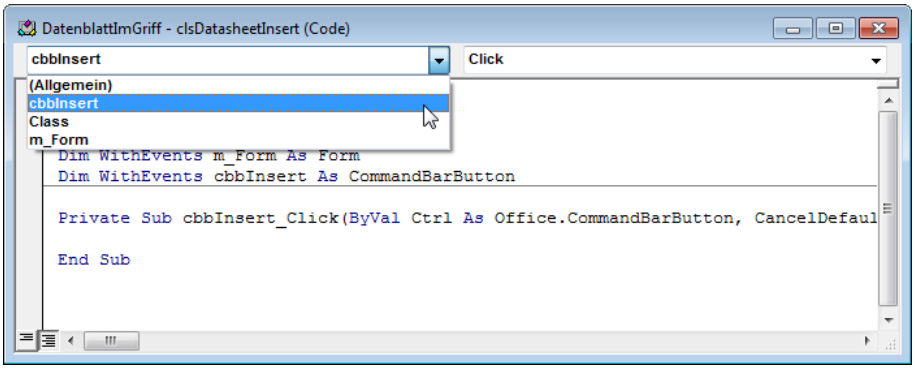


Bild 5: Anlegen einer Ereignisprozedur, die beim Anklicken eines Kontextmenü-Eintrags ausgelöst wird

Diese Prozedur ergänzen Sie nun wie in Listing 2. Die Prozedur fragt per Inputbox den einzufügenden Inhalt ab. Dabei verwendet sie als Standardwert einen Wert, den die Funktion **AusZwischenablage** liefert.

Diese finden Sie im Modul **mdlTools**. Die Funktion gibt den aktuellen Inhalt der Zwischenablage zurück. Auf diese Weise könnte der Benutzer wie gewohnt den einzufügenden Text per **Strg + C** oder mit dem Kontextmenü-Befehl **Kopieren** ermitteln.

Enthält die Variable **strInhalt** nach dem Eintragen des Ergebnisses der **InputBox**-Funktion keine leere Zeichenfolge, stellt die Prozedur die Variable **bolFill** auf den Wert **True** ein. Falls nicht, soll die Prozedur prüfen, ob der Benutzer vielleicht einfach nur die **Abbrechen**-Schaltfläche betätigt hat (was ebenfalls eine leere Zeichenkette zurückliefert) oder ob er tatsächlich eine leere Zeichenkette eingegeben hat, um die markierten Felder der Datenblattansicht zu leeren beziehungsweise mit einer leeren Zeichenkette zu füllen. Wählt der Benutzer hier die Antwort **Ja** (**vbYes**), wird **bolFill** ebenfalls auf **True** eingestellt. Hat **bolFill** im Anschluss den Wert **True**, ruft die Prozedur die Routine **Fill** mit dem einzusetzenden Wert aus der Variablen **strInhalt** als Parameter auf.

Füllen der markierten Felder

Nun folgt der interessante Teil der Lösung. Hier benötigen wir zunächst einen Ansatz, um die markierten Felder zu füllen.

Um die entsprechenden Inhalte zu ändern, gibt es zwei Wege:

```
Private Sub cbbInsert_Click(ByVal Ctrl As Office.CommandBarButton, CancelDefault As Boolean)
    Dim strInhalt As String
    Dim bolFill As Boolean
    strInhalt = InputBox("Geben Sie den einzufügenden Wert an:", "Zellen füllen", AusZwischenablage)
    If Len(strInhalt) > 0 Then
        bolFill = True
    Else
        If MsgBox("Felder leeren?", vbYesNo) = vbYes Then
            bolFill = True
        End If
    End If
    If bolFill Then
        Fill strInhalt
    End If
End Sub
```

Listing 2: Die Ereignisprozedur, die beim Auswählen des Kontextmenü-Eintrags ausgelöst wird

Der ACLib-FilterForm-Wizard

In Access-Formularen wird regelmäßig Code zum dynamischen Erzeugen eines Filterausdrucks eingebaut. Wie Sie diesen Code übersichtlich gestalten, steht im Beitrag „SQL-Text für Filterbedingungen“ (S. 46). Damit die Filter-Klassen aus diesem Beitrag genutzt werden können, ist Code im Formular erforderlich. Mit diesem Code wird die Filterung ausgelöst und der Filterausdruck erzeugt. Außerdem müssen Sie die Klassen in ihre Anwendung einfügen. Diese Aufgaben nimmt Ihnen der ACLib-FilterForm-Wizard ab.

ACLib-FilterForm-Wizard

Der ACLib-FilterForm-Wizard ist ein Access-Add-In und wird mit dem Access-Add-In-Manager von Access installiert. Über den Add-In-Menüeintrag **ACLib FilterForm Wizard** öffnen Sie den Assistenten.

Nach Auswahl des Filterformulars wird der Assistent wie in Bild 1 dargestellt. Mit den drei Dropdown-Feldern im rechten oberen Bereich besteht die Möglichkeit, die Steuerelemente zum Anwenden der Filter, zum Löschen aller Filterwerte und das Steuerelement zum Aktivieren der automatischen Filterung festzulegen.

Im mittleren Bereich **FilterControls** werden die Filterbedingungen und die dazugehörigen Steuerelemente mit den Filterwerten definiert. Im unteren Bereich sehen Sie, ob die benötigten Klassen bereits in Ihrer Anwendung sind. Sie können diese Klasse über den Assistent vorab in Ihre Anwendung einfügen, wenn Sie auf **Klassen installieren** klicken. Die Klassen werden automatisch eingefügt, sobald der Filtercode im Formular erstellt wird.

Wählen Sie in **UseFilter-Methode** die Code-Variante für die **UseFilter**-Prozedur. Die Auswahl **Filter-Variante** definiert die im Filterformular anzuwendende Klasse. Im Dropdown-Feld **SQL-Dialekt** legen Sie die benötigten SQL-Formate fest.

Mit Klick auf **Formular-Code einfügen** wird der Code zum Filtern in das ausgewählte Filterformular eingefügt.

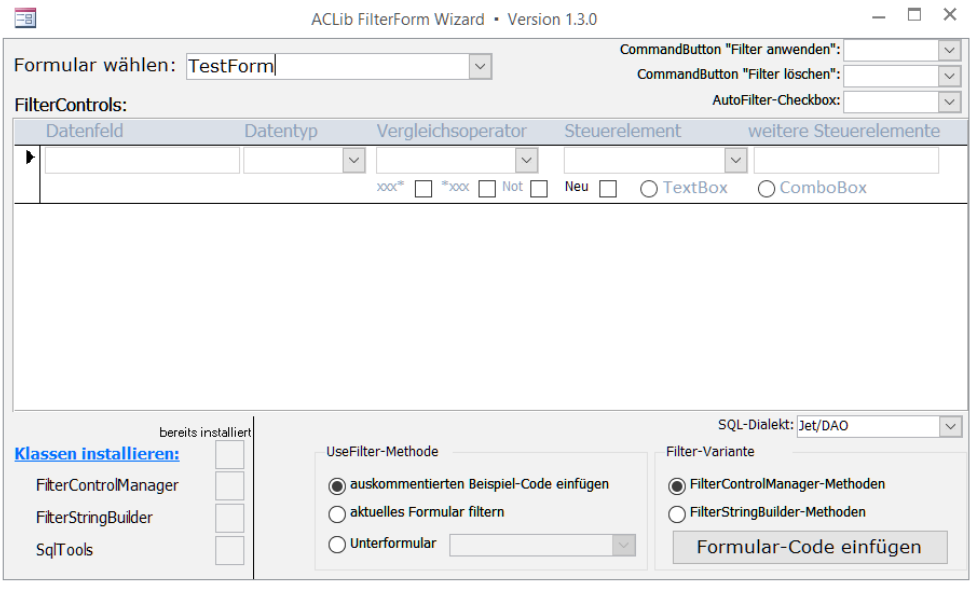


Bild 1: ACLib-FilterForm-Wizard – Ansicht nach Auswahl eines Formulars

Filterformular erstellen

Bevor Sie den FilterForm-Wizard öffnen, erstellen Sie ein Formular mit allen benötigten Steuerelementen zur Filterwerteingabe, zum Auslösen der Filter und so weiter. Falls Sie die gefilterten Daten in einem Unterformular darstellen wollen, fügen Sie ein Unterformularsteuerelement in das Formular ein. Vergessen Sie nicht, die Steuerelemente passend zu benennen.

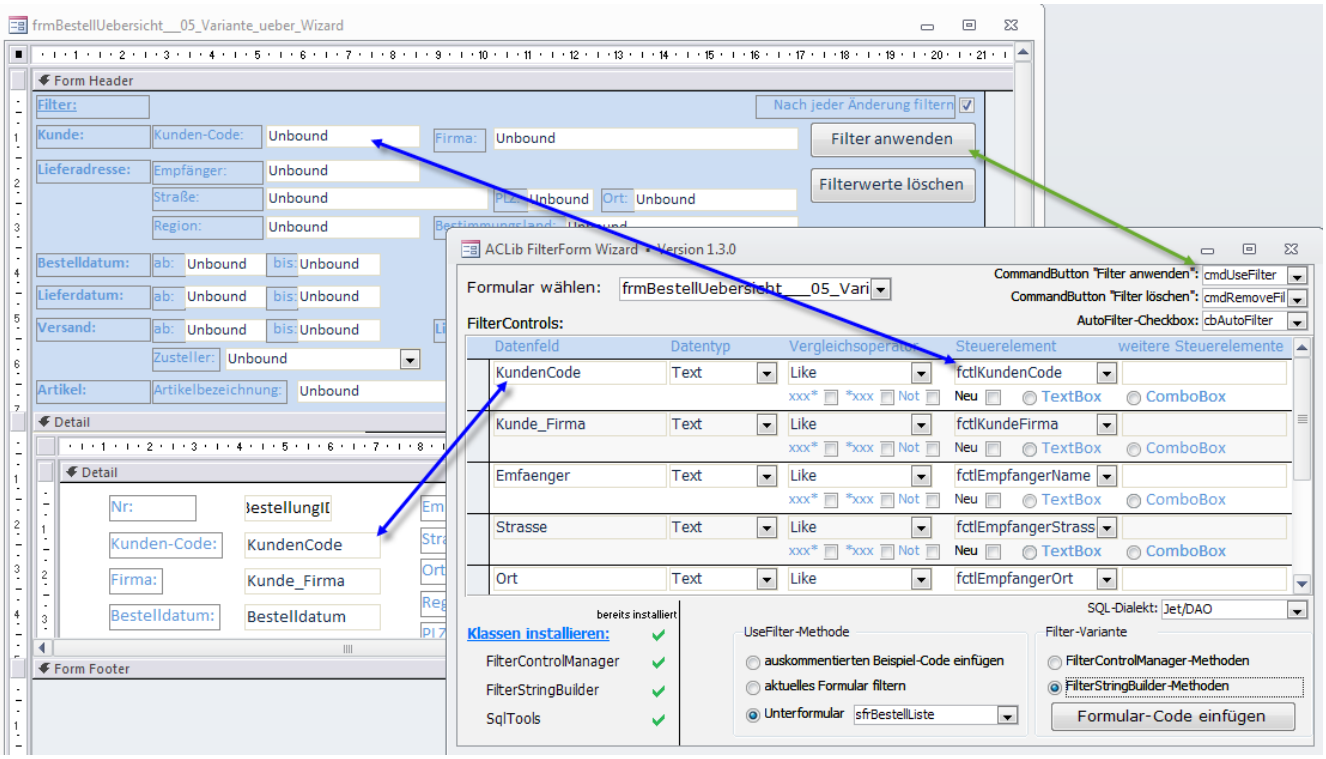


Bild 2: Filterbedingungen und Steuerelemente einstellen

Ein vorbereitetes Formular zum Ausprobieren des Assistenten ist in der Beispiel-Anwendung enthalten. Erstellen Sie eine Kopie vom Formular **frmBestellUebersicht___00_ohne_Code**. Für diesen Beitrag wurde die Kopie unter dem Namen **frmBestellUebersicht___05_Variante_ueber_Wizard** gespeichert.

Filterbedingungen definieren

Nach dieser Vorbereitung öffnen Sie den Assistenten und wählen Ihr Filterformular aus. Im Anschluss stellen Sie die Filterbedingungen und die benötigten Steuerelemente, wie in Bild 2 gezeigt, ein. Je Filterbedingung müssen Sie das zu filternde Datenfeld, den Datentyp, den gewünschten Vergleichsoperator und das Steuerelement mit dem Filterwert definieren. Die zur ersten Filterbedingung in der Abbildung passende Code-Zeile sehen Sie hier:

```
.Add "KundenCode", SQL_Text, SQL_Like, _
                                Me.fctlKundenCode.Value
```

Wenn Sie die Checkbox **xxx*** aktivieren, wird im Code statt **SQL_Like** der Ausdruck **SQL_Like + SQL_Add_WildCardSuffix** eingefügt. Damit wird bei der **Filterstring**-Erstellung der Wert am Ende mit * ergänzt.

Neben dem Auswahlfeld des Filtersteuerelements ist ein Textfeld zur Eingabe weiterer Steuerelementnamen. Dieses Textfeld benötigen Sie zum Definieren einer Filterbedingung, bei der mehr als zwei Filtersteuerelemente benötigt werden.

Im Beispielformular werden aus den Steuerelementen für die Datumsfilter **Between**-Filterausdrücke erstellt.

Aus dem Eintrag der **Between**-Filterbedingung in Bild 3 entsteht die folgende Code-Zeile:

```
.Add "Bestelldatum", SQL_Date, SQL_Between, _
    Me.fctlBestelldatumAb.Value, _
    Me.fctlBestelldatumBis.Value
```

```
Option Compare Database
Option Explicit

Private Function FilterControlValueChanged()
   If Me.cbAutoFilter.Value = True Then
      RefreshFilter
   Else
      Me.cmdUseFilter.FontBold = True
   End If
End Function

Private Sub RefreshFilter()
   UseFilter GetFilterString()
End Sub

Private Function UseFilter(ByVal NewFilterString As String)
   With Me.sfrBestellListe.Form
      .Filter = NewFilterString
      .FilterOn = (Len(.Filter) > 0)
   End With
   Me.cmdUseFilter.FontBold = False
End Function

Private Sub RemoveFilter()
   RemoveFilterValues
   UseFilter "0=1" ' Anzeige leeren
   Me.cmdUseFilter.FontBold = False
End Sub

Private Sub RemoveFilterValues()
   Dim fctl As Control
   For Each fctl In GetFilterControls()
      fctl.Value = Null
   Next
End Sub

Private Function GetFilterControls() As Collection
   Dim fctlCol As Collection
   Set fctlCol = New Collection
   'Filter-Steuerelemente anfügen:
   fctlCol.Add Me.fctlKundenCode
   fctlCol.Add Me.fctlKundeFirma
   fctlCol.Add Me.fctlEmpfangerName
   fctlCol.Add Me.fctlEmpfangerStrasse
   fctlCol.Add Me.fctlEmpfangerOrt
   Set GetFilterControls = fctlCol
End Function
```

Listing 1: Erstellter Filter-Formular-Code, Teil I

Berichtsbereiche, Gruppierungen und Sortierungen

Berichte haben verschiedene Bereiche, in die Sie die anzuzeigenden Daten einordnen können. Dazu gehört natürlich der Detailbereich, der für jeden Datensatz der Datenherkunft einmal angezeigt wird, der Berichtskopf und der Berichtsfuß sowie der Seitenkopf und der Seitenfuß. Schließlich kommen noch die entsprechenden Kopf- und Fußbereiche eventuell eingerichteter Gruppierungen hinzu. Dieser Beitrag zeigt, was Sie mit den einzelnen Bereichen anstellen können.

Wenn Sie alles aus Berichten herausholen wollen, müssen Sie die einzelnen Bereiche eines Berichts kennen und wissen, welche Ereignisse zu welcher Phase des Renderns eines Bereichts ausgelöst werden.

Berichtsbereiche

Wenn Sie einen Bericht neu erstellen, zeigt dieser drei Bereiche an – den Seitenkopf, den Detailbereich und den Seitenfuß. Seitenkopf und Seitenfuß werden auf jeder Seite angezeigt, der Detailbereich für jeden darzustellenden Datensatz je einmal.

Wenn Sie Informationen nur am Anfang und am Ende eines Berichts anzeigen wollen, fügen Sie noch die Bereiche **Berichtskopf** und **Berichtsfuß** ein – und zwar über den entsprechenden Kontextmenü-Eintrag **Berichtskopf/-fuß** einer Bereichsüberschrift im Bericht.

Fehlen noch die Gruppierungen: Damit können Sie nach einzelnen Feldern der Datenherkunft eines Berichts gruppieren – also etwa eine Liste von Artikeln nach einer Kategorie. Das ist insbesondere interessant, da Sie für jede Gruppierung auch einen Gruppenkopf und einen Gruppenfuß anlegen können. Der Gruppenkopfbereich könnte etwa den Namen der Kategorie aufführen, der Gruppenfußbereich die Anzahl der in der Kategorie enthaltenen Artikel.

Gruppierungen

In neueren Access-Versionen ab Version 2007 finden Sie den Bereich zum Einrichten von Gruppierungen und Sortierungen unten im Access-Fenster. Mit den beiden Schaltflächen **Gruppe hinzufügen** und **Sortierung hinzufügen** fügen Sie Gruppierungen und Sortierungen hinzu. Eine Konstellation mit zwei Gruppierungen sieht etwa wie in Bild 1 aus.

Alternativ zum Betätigen einer der Schaltflächen und anschließendem Auswählen des Feldes, nach dem Sie gruppieren oder sortieren möchten, können Sie das gewünschte Feld in der Entwurfsansicht des Berichts auch direkt aus der Feldliste an die gewünschte Stelle der Gruppierungs- und Sortierungsoptionen ziehen.

Der Unterschied zwischen dem Hinzufügen einer Gruppierung oder Sortierung ist, dass bei der Gruppierung gleich ein Kopfbereich mit hinzugefügt wird. Dies können Sie aber wie in Bild 2 auch für eine Sortierung nachholen und diese so in eine Gruppierung umwandeln.

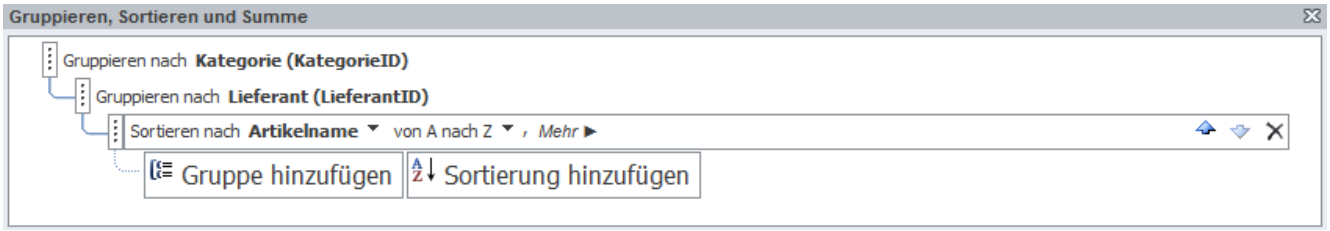


Bild 1: Dialog zum Festlegen von Sortierungen und Gruppierungen

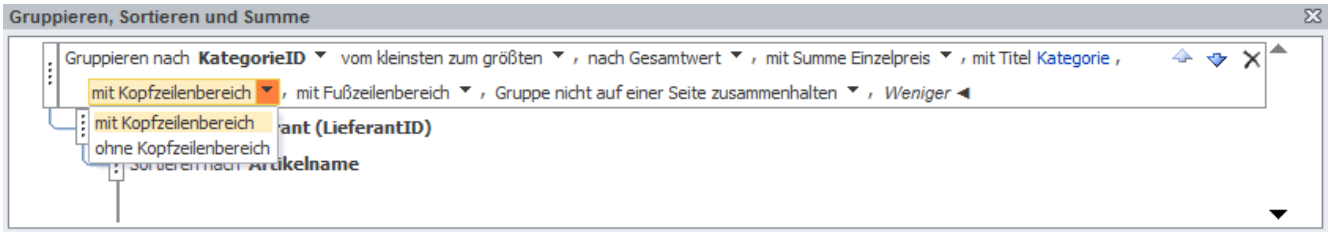


Bild 2: Festlegen der Bereiche einer Gruppierung

Der Beispielbericht sieht nun ohne Steuerelemente bereits wie in Bild 3 aus.

Die Einstellungen aus dieser Abbildung sorgen übrigens dafür, dass die zugrunde liegenden Artikel zunächst nach Kategorien und dann innerhalb der Kategorien nach Lieferanten gruppiert werden. Die Daten jeder Lieferanten-Gruppe werden dann noch einer aufsteigenden Sortierung unterzogen.

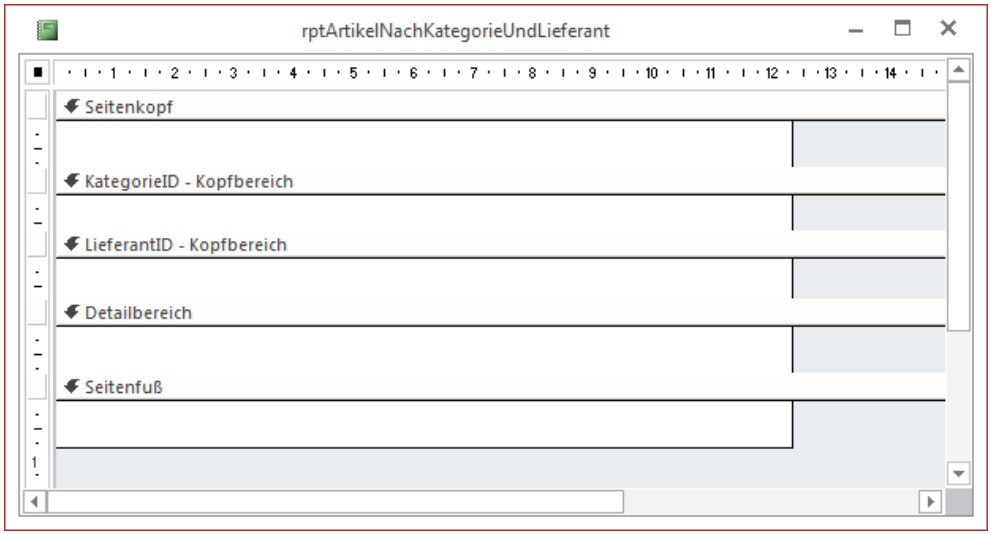


Bild 3: Berichtsbereiche inklusive einer Gruppierung

Eigenschaften von Berichten und Berichtsbereichen

Berichte und die einzelnen Bereiche verwenden einige Eigenschaften, die wichtig für die Gestaltung des Berichts sind. Dummerweise sind die Eigenschaften nicht an einem Ort aufzufinden, sondern über das Eigenschaftsfenster und den soeben vorgestellten Bereich **Gruppieren, Sortieren und Summe** verteilt. Nachfolgend schauen wir uns diese Eigenschaften im Detail an.

Kopfzeilenbereich und Fußzeilenbereich

Diese beiden Eigenschaften finden Sie im Bereich **Gruppieren, Sortieren und Summe**. Sie legen damit für Gruppierungen fest, ob diese einen Kopf- und/oder einen Fußbereich anzeigen sollen. Sie können diese Eigenschaften unter VBA mit den Eigenschaften **GroupHeader** und **GroupFooter** ansprechen. Die Eigenschaften gehören zum **GroupLevel**-Objekt. Dieses können Sie mit einer Variablen des Typs **GroupLevel** referenzieren. Dieses **GroupLevel**-Objekt liefert noch mehr Eigenschaften als nur **GroupHeader** und **GroupFooter** zurück. Die folgende Ereignisprozedur legen Sie für die Ereigniseigenschaft **Beim Laden** des Berichts fest (s. Bild 4):

```
Private Sub Report_Load()
    Dim grp As GroupLevel
    Set grp = Me.GroupLevel(0)
    Debug.Print "GroupInterval: " & grp.GroupInterval
    Debug.Print "ControlSource: " & grp.ControlSource
    Debug.Print "GroupFooter: " & grp.GroupFooter
    Debug.Print "GroupHeader: " & grp.GroupHeader
    Debug.Print "GroupOn: " & grp.GroupOn
    Debug.Print "KeepTogether: " & grp.KeepTogether
```

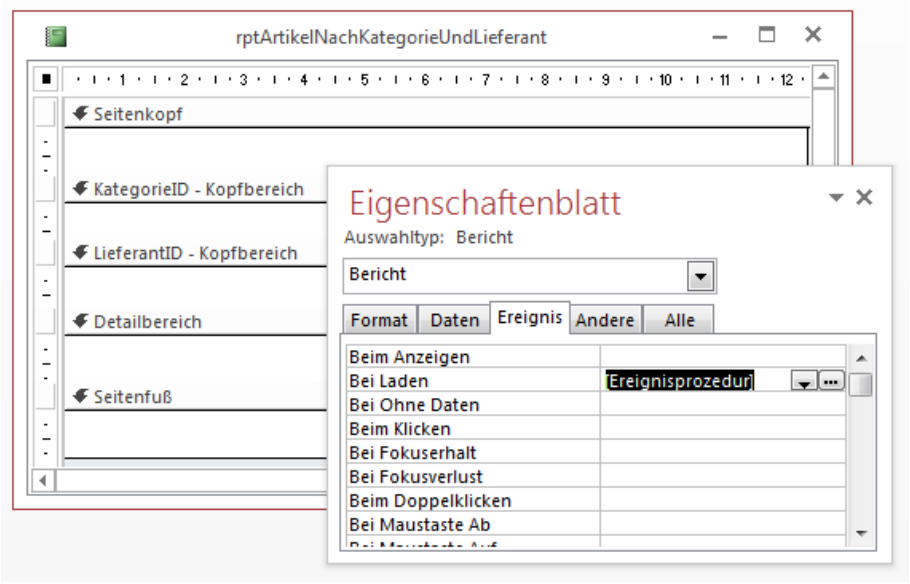


Bild 4: Erstellen einer Ereignisprozedur, die Bereichseigenschaften beim Laden des Berichts ausgibt

```
    Debug.Print "SortOrder: " & grp.SortOrder
End Sub
```

In diesem Fall lassen wir die Eigenschaften für das **GroupLevel**-Objekt mit dem Index **0** ausgeben. Dies liefert folgende Ergebnisse, die zeigen, dass es sich um den Gruppenkopf der Gruppierung nach dem Feld **KategorieID** handelt:

```
GroupInterval: 1
ControlSource: KategorieID
GroupFooter:   Falsch
GroupHeader:   Wahr
GroupOn:       0
KeepTogether:  0
SortOrder:     Falsch
```

Mit **GroupHeader** und **GroupFooter** legen Sie also fest, ob ein Kopf- und ein Fußbereich eingeblendet werden sollen.

Die Eigenschaften Gruppieren nach und Intervall

Oben haben wir schon gesehen, dass die Eigenschaft **ControlSource** der Gruppierungsebene den Namen des Feldes liefert, nach dem gruppiert werden soll. In diesem Fall soll nach jedem Wert des Feldes **KategorieID** gruppiert werden.

Dazu legen Sie in der Benutzeroberfläche für die Eigenschaft den entsprechenden Feldnamen fest.

Wir können eine Gruppe jedoch auch anders nutzen als zur Gruppierung nach allen Elementen der Gruppe. Unter den Eigenschaften des **GroupLevel**-Objekts finden wir nämlich auch noch die beiden Eigenschaften **GroupOn** und **GroupInterval**. **GroupOn** enthält hier standardmäßig den Wert **0**.

Für das folgende Beispiel entfernen Sie die bisherigen Gruppen und legen eine neue Gruppe nach dem Feld **Artikelname** an. Stellen Sie dann die Eigenschaft, die standardmäßig den Wert **nach Gesamtwert** enthält, auf den Wert **Benutzerdefiniert** ein und legen Sie unter **Zeichen** den Wert **1** fest (s. Bild 5).

Was bewirkt dies nun? Um dies zu sehen, fügen Sie dem Detailbereich ein paar der Felder der Tabelle **tblArtikel** hinzu. Den Gruppenkopfbereich der soeben erstellten Gruppe ergänzen Sie um den Ausdruck **=Links([Artikelname];1)** als Wert der Eigenschaft **Steuerelementinhalt** (s. Bild 6).

Wenn Sie nun in die Seitenvorschau wechseln, erhalten Sie die Ansicht aus Bild 7. Der Bericht gruppiert die Artikel nach dem Anfangsbuchstaben und fügt jeder Gruppe einen Gruppenkopfbereich hinzu, der den ersten Buchstaben einer jeden Gruppe ausgibt.

Auf diese Weise haben Sie mit wenigen Handgriffen eine Artikelliste erstellt, die jeweils den Anfangsbuchstaben vor einer Gruppe entsprechender Artikel ausgibt.

Projektzeitauswertung

Im Beitrag »Projektzeiterfassung mit Outlook und Access« haben wir Outlook um Funktionen erweitert, die eine einfache Erfassung von Projektzeiten und deren Speicherung in eine Access-Datenbank ermöglichen. Wenn Sie auf diese Weise einige Stunden mit Ihren Projekten erfasst haben, wollen Sie die gespeicherten Daten vermutlich für verschiedene Auswertungen nutzen – beispielsweise, um die bei einem Projekt aufgelaufenen Stunden zu ermitteln. Dieser Beitrag zeigt, wie Sie dies auf einfache Weise erledigen.

Im Beitrag **Projektzeiterfassung mit Outlook und Access** (**www.access-im-unternehmen.de/1016**) haben wir ja bereits das Datenmodell vorgestellt, dessen Tabellen unsere Projektzeiten aufnehmen sollen. Dort finden Sie eine Projekte-, eine Aufgaben- und eine Tätigkeitentabelle (s. Bild 1). Formulare zum Bearbeiten dieser Objekte benötigen wir prinzipiell nicht – das Anlegen, Bearbeiten und Löschen erfolgt vollständig über Outlook.

Was wir jedoch benötigen, ist ein Formular zur Auswahl der auszuwertenden Daten. Dieses sollte die Möglichkeit bieten, Tätigkeiten nach Projekt, Zeitraum et cetera zu filtern. Wir beginnen jedoch mit einem Bericht, der einen Überblick über alle Elemente liefern soll – also alle Projekte, Aufgaben und Tätigkeiten über den gesamten Zeitraum.

Übrigens: Wenn Sie die Projektzeiterfassung nutzen, stellt Outlook eine Verbindung zur Datei Projektzeiterfassung.accdb her. Sie können diese dann nicht mehr exklusiv öffnen, um beispielsweise neue Berichte hinzuzufügen.

Wenn Sie die Objekte der Datenbank bearbeiten möchten, sollten Sie also zuvor Outlook schließen und somit den Zugriff auf die Datenbankdatei freigeben.

Gesamtübersicht über Projekte, Aufgaben und Tätigkeiten

Diesen Bericht legen wir der Einfachheit halber zunächst in der Berichtsansicht an, die mit Access 2007 eingeführt wurde, aber wenig Beachtung findet. Erstellen Sie also einen neuen Bericht und speichern Sie diesen unter dem Namen **rptProjekteAufgabenTaetigkeiten**.

Danach erstellen Sie eine neue Abfrage namens **qryProjekteAufgabenTaetigkeiten** und fügen dieser die Tabellen **tblProjekte**, **tblAufgaben** und **tblTaetigkeiten** hinzu.

Aus der Tabelle **tblProjekte** benötigen wir die Felder **ProjektID** und **Projekt**, aus **tblAufgaben** die Felder **AufgabeID** und **Aufgabe** und aus **tblTaetigkeiten** die Felder **TaetigkeitID**, **Taetigkeit**, **Startzeit** und **Endzeit**. Außerdem ziehen Sie aus allen drei Tabellen das Feld **Ge-**

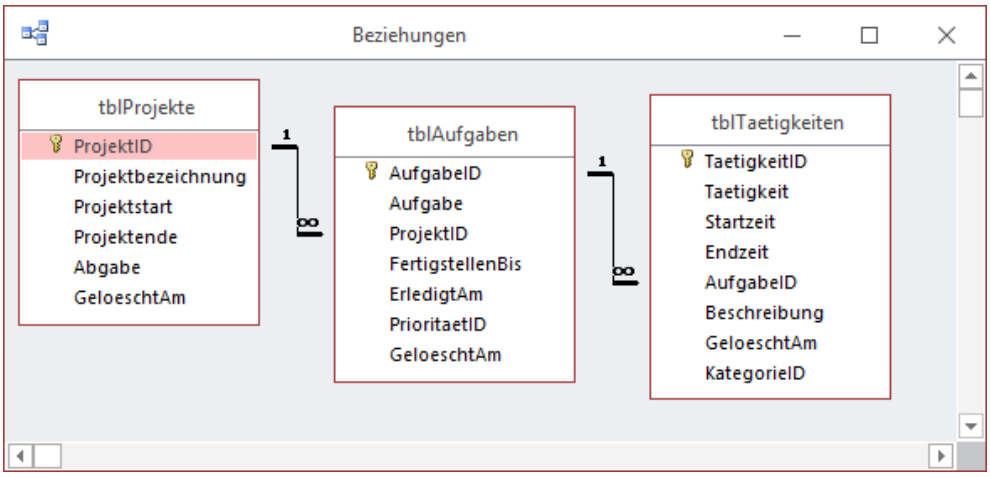


Bild 1: Datenmodell der zu berücksichtigenden Tabellen der Projektzeiterfassung

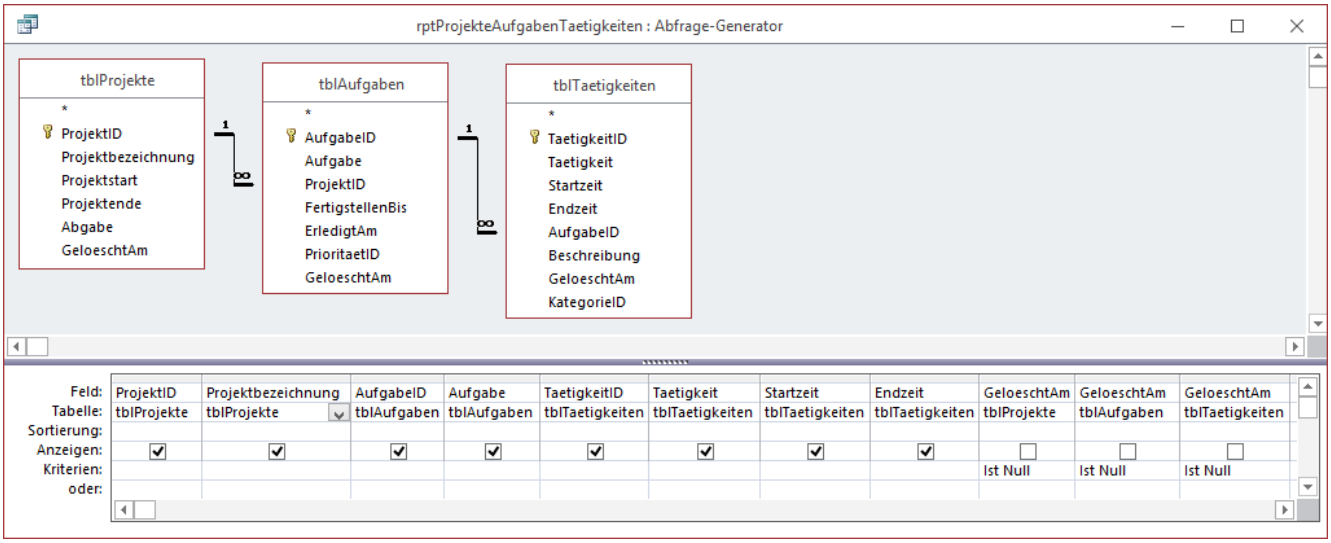


Bild 2: Datenherkunft des Berichts zur Anzeige aller Projekte, Aufgaben und Tätigkeiten

loeschtAm in das Entwurfsraster. Stellen Sie als Kriterium für diese Felder den Ausdruck **Ist Null** ein. Auf diese Weise sorgen wir dafür, dass nur solche Daten in den Bericht gelangen, die nicht über Outlook gelöscht und in den Access-Tabellen als gelöscht markiert wurden (s. Bild 2).

Damit wir ein wenig Spielmaterial für die Erstellung der Berichte haben, finden Sie in Bild 3 eine Arbeitswoche mit einigen Tätigkeiten für zwei verschiedene Projekte und diverse Aufgaben.

Die darin enthaltenen Daten schlagen sich in der Abfrage **qryProjekteAufgabenTaetigkeiten** wie in Bild 4 nieder. Diese müssen wir nun nur noch in entsprechender Form im Bericht anordnen.

Berichtsbereiche definieren

Ein Bericht, der Daten zu drei verschiedenen Tabellen anzeigen soll, müssen Sie natürlich mit entsprechenden Gruppierungen versehen. Dazu klicken Sie in der Entwurfsansicht des Berichts im Ribbon auf den Eintrag **Entwurf|Gruppierung und Summen|Gruppieren und sortieren**. Dies blendet im unteren Abschnitt des Access-Fensters den Bereich **Gruppieren, Sortieren und Summe** ein. Hier klicken Sie zunächst auf **Gruppe hinzufügen**. Wählen Sie unter **Feld auswählen** dann das Feld **ProjektID** aus. Fügen Sie nun noch eine weitere Gruppe hinzu und legen Sie dafür das Feld **AufgabeID** fest. Außerdem wollen wir die Tätigkeiten innerhalb der Gruppen noch nach dem Startdatum sortieren. Fügen Sie nun also noch eine Sortierung über die Schaltfläche **Sortierung hinzufügen** hinzu und geben Sie das Feld **Startdatum** unter **Feld auswählen** an.

Für die **ProjektID**- und **AufgabeID**-Gruppen wollen wir Gruppenköpfe anzeigen, welche die Bezeichnungen der Projekte und Gruppen anzeigen. Der Detailbereich soll die Tätigkeiten und die Start- und Endzeiten enthalten. Wir gehen hier vereinfachend davon aus, dass die Tätigkeiten zu normalen Tageszeiten durchgeführt werden und nicht über die Tagesgrenzen hinweg.

Dadurch können wir mit der Vorgabe einer entsprechenden Formatierung einen Ausdruck wie den folgenden verwenden:

```
=Format([Startzeit];"tttt"", ""tt.mm.jjjj") &
" " & Format([Startzeit];"Zeit, 24Std") & "-" &
Format([Endzeit];"Zeit, 24Std")
```

Dies liefert die Zeitspanne der Tätigkeit etwa in der Form **Montag, 1.1.2015 12:00-12:25**.

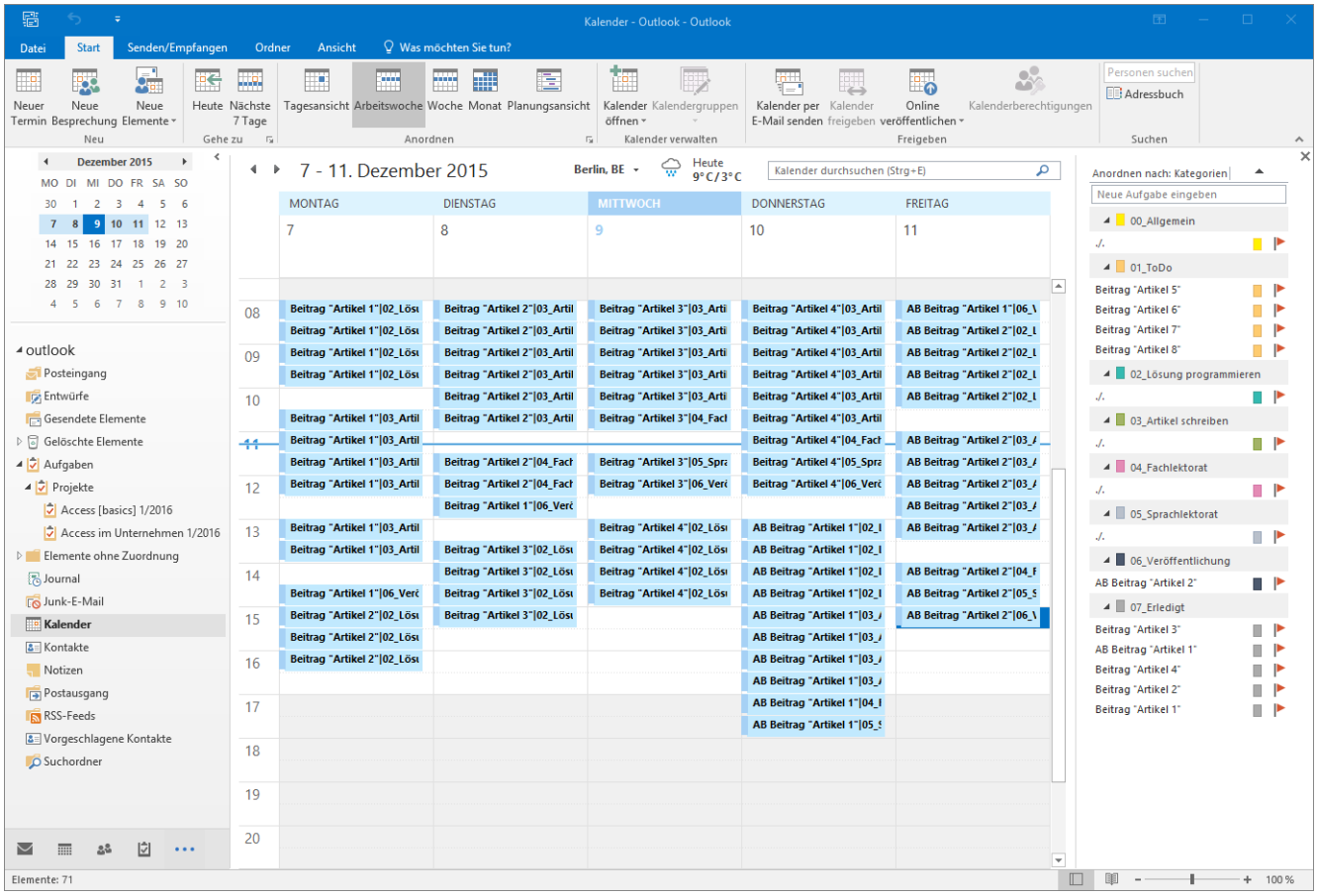
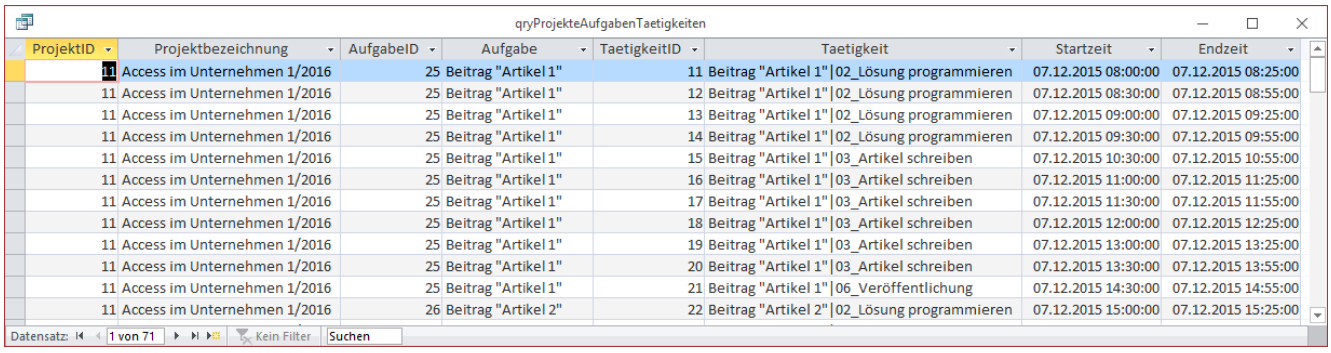


Bild 3: Diesen vollen Wochenplan wollen wir gleich anschaulich in einem Bericht darstellen.



ProjektID	Projektbezeichnung	AufgabeID	Aufgabe	TaetigkeitID	Taetigkeit	Startzeit	Endzeit
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	11	Beitrag "Artikel 1"|02_Lösung programmieren	07.12.2015 08:00:00	07.12.2015 08:25:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	12	Beitrag "Artikel 1"|02_Lösung programmieren	07.12.2015 08:30:00	07.12.2015 08:55:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	13	Beitrag "Artikel 1"|02_Lösung programmieren	07.12.2015 09:00:00	07.12.2015 09:25:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	14	Beitrag "Artikel 1"|02_Lösung programmieren	07.12.2015 09:30:00	07.12.2015 09:55:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	15	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 10:30:00	07.12.2015 10:55:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	16	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 11:00:00	07.12.2015 11:25:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	17	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 11:30:00	07.12.2015 11:55:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	18	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 12:00:00	07.12.2015 12:25:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	19	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 13:00:00	07.12.2015 13:25:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	20	Beitrag "Artikel 1"|03_Artikel schreiben	07.12.2015 13:30:00	07.12.2015 13:55:00
11	Access im Unternehmen 1/2016	25	Beitrag "Artikel 1"	21	Beitrag "Artikel 1"|06_Veröffentlichung	07.12.2015 14:30:00	07.12.2015 14:55:00
11	Access im Unternehmen 1/2016	26	Beitrag "Artikel 2"	22	Beitrag "Artikel 2"|02_Lösung programmieren	07.12.2015 15:00:00	07.12.2015 15:25:00

Datensatz: 1 von 71 | Kein Filter | Suchen

Bild 4: Projekte, Aufgaben und Tätigkeiten in der Datenherkunft des Berichts

Alternierenden Hintergrund ausschalten und Rahmen ausblenden

Den Wechsel der Hintergrundfarbe wollen wir für diesen Bericht deaktiveren. Dazu klicken Sie jeweils auf die Bereichsköpfe, um die jeweiligen Bereiche zu markieren, und stellen dann im Eigenschaftsfenster den Wert der Eigenschaft **Alternative Hintergrundfarbe** auf den Wert der Eigenschaft **Hintergrundfarbe** ein (zum Beispiel **Hintergrund 1**).

Dies ist in unserem Bericht für die Bereiche **ProjektID - Kopfbereich**, **AufgabeID - Kopfbereich** und **Detailbereich** notwendig.

Außerdem wollen wir die Rahmen um die Textfelder herum entfernen, die etwa unter Access 2013 oder Access 2016 automatisch hinzugefügt werden. Dazu markieren Sie die entsprechenden Steuerelemente und stellen die Eigenschaft **Rahmenart** auf **Transparent** ein.

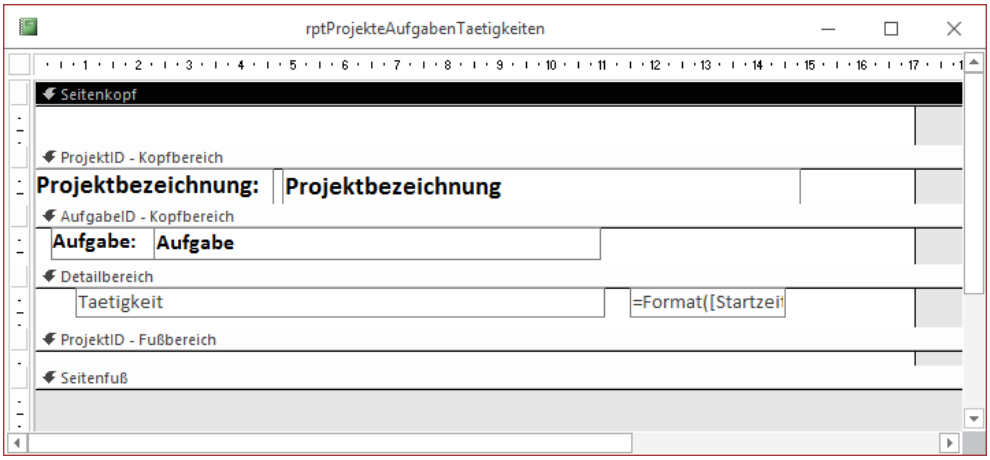


Bild 5: Erster Entwurf der Tätigkeitsübersicht

Dem Bereich **ProjektID - Kopfbereich** fügen Sie nun das Feld **Projektbezeichnung** hinzu und stellen seine **Schriftgröße** auf **14**, die **Schriftbreite** auf **Fett** und die **Schriftfarbe** auf **Schwarz** ein.

Den Bereich **AufgabeID - Kopfbereich** statten Sie mit dem Feld **Aufgabe** aus. Die Formatierung gestalten Sie wie bei der Projektbezeichnung, aber die **Schriftgröße** soll **12** lauten. Rücken Sie die Steuerelemente dieser Gruppe etwa ein, um diese optisch von der übergeordneten Ebene zu trennen.

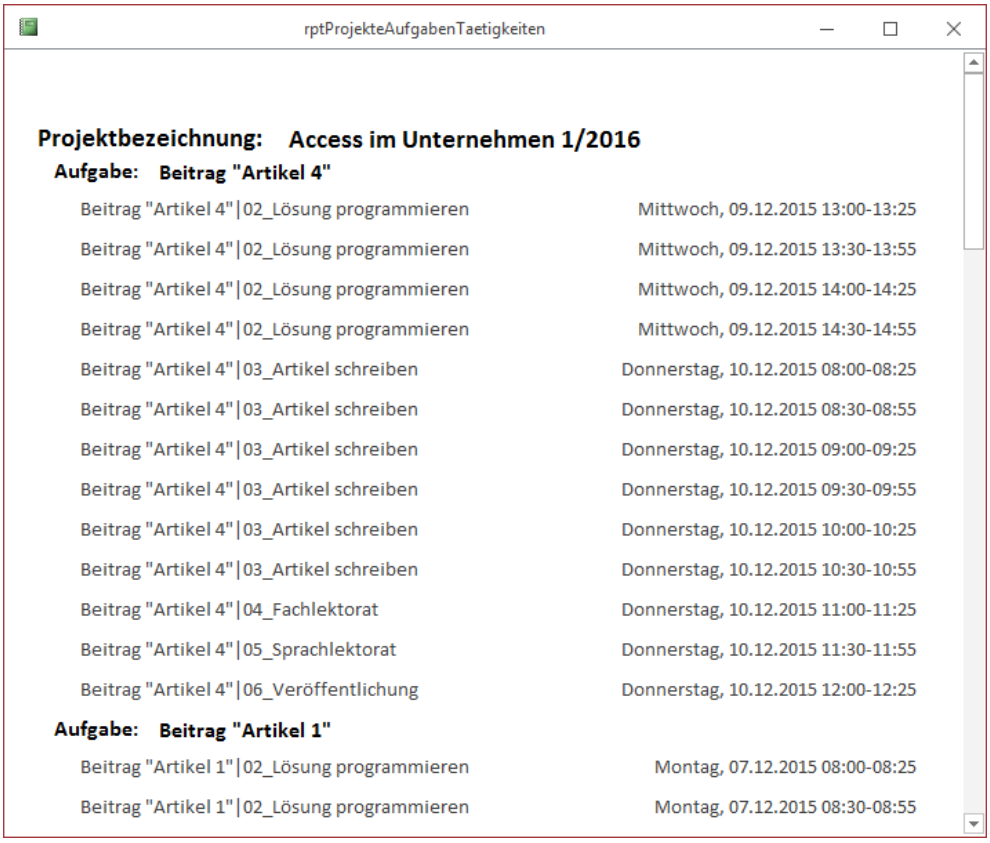


Bild 6: Berichtsansicht der Tätigkeiten

Fehlt noch das Feld **Taetigkeit**, das Sie im Detailbereich platzieren. Hier lassen wir das Bezeichnungsfeld weg. Die Schriftgröße beträgt **11**, die **Schriftbreite** ist **Normal**. Rechts vom Feld **Taetigkeit** bringen Sie ein neues Textfeld namens **txtZeit** unter, das Sie mit dem oben erwähnten Ausdruck für die Tätigkeitszeit füllen (s. Bild 5).

Das Ergebnis sieht dann etwa wie in Bild 6 aus. Dies ist eine gute Basis, um weitere Verfeinerungen darzustellen.

Es wäre beispielsweise für manche Zwecke praktisch, wenn Wochentag und Datum nicht zu jeder Position

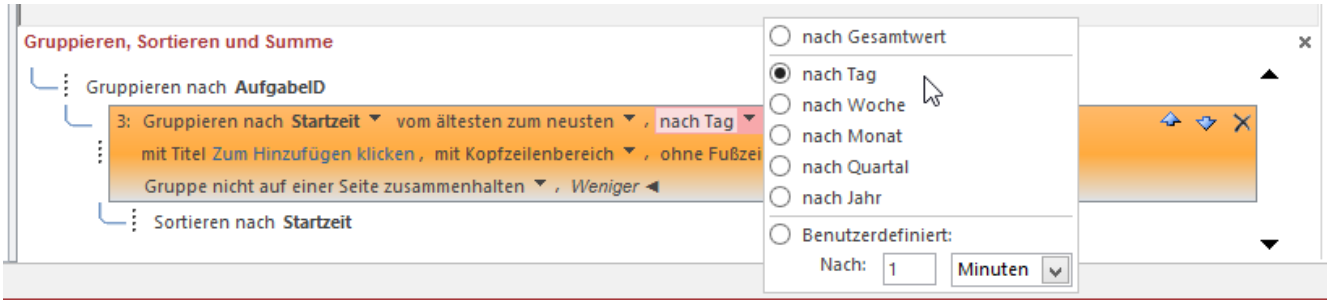


Bild 7: Gruppierung nach dem Tag der Startzeit

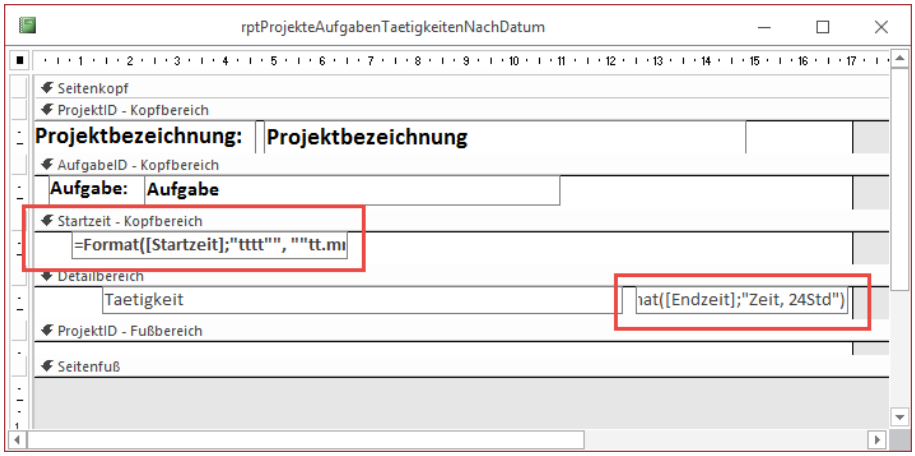


Bild 8: Tätigkeiten gruppiert nach Tagen

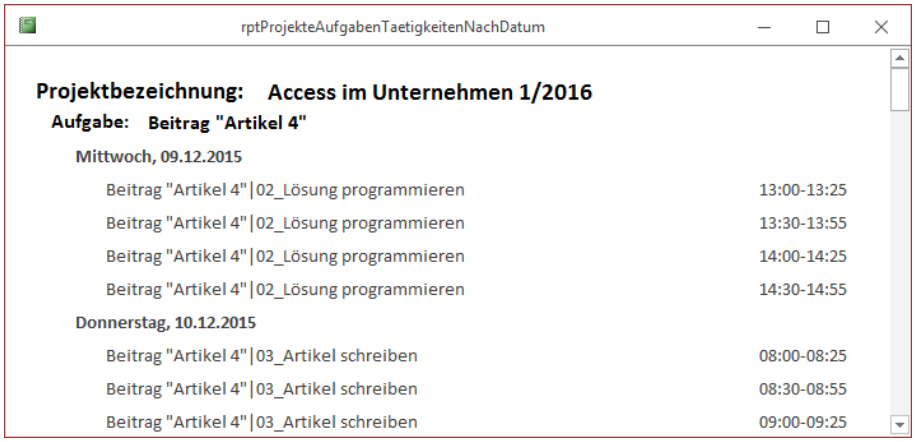


Bild 9: Tagesgruppierung in der Berichtsansicht

erscheinen würden, sondern als weiterer Gruppenkopf. Der passende Entwurf enthält dementsprechend eine weiterere Gruppierungsebene, die diesmal das Feld **Startzeit** enthält und wieder einen Gruppenkopf anzeigen soll. In diesem Fall müssen wir eine weitere Einstellung für die Gruppierungseigenschaften vornehmen: Dabei wählen Sie den Wert **nach Tag** als Gruppierungsintervall aus (s. Bild 7).

Die neue Gruppierung nimmt ein einziges Textfeld namens **txtDatum** mit dem Datum und dem folgenden Ausdruck für die Eigenschaft **Steuerelementinhalt** auf (s. Bild 8):

```
=Format([Startzeit]; "tttt"", ""tt.mm.jjjj")
```

Den Eintrag für die Eigenschaft **Steuerelementinhalt** des Textfeldes **txtZeit** reduzieren wir entsprechend:

```
=Format([Startzeit];"Zeit, 24Std") & "-" & Format([Endzeit];"Zeit, 24Std")
```

Die Berichtsansicht liefert nun die Tätigkeiten nach Tagen gruppiert und zeigt eine entsprechende Gruppierungsüberschrift an (s. Bild 9).

Summen in den Gruppenfüßen

Nun wäre es noch interessant, wie viel Zeit für die Erledigung einzelner Aufgaben und kompletter Projekte

SQL-Text für Filterbedingungen mit Klasse

Mit dem Einsatz von Klassen können Sie Ihren Programmcode übersichtlich gestalten. Wenn Sie konkrete Aufgaben mittels wiederverwendbarer Klassen umsetzen, erhalten Sie zusätzlich eine Code-Sammlung, die die Code-Qualität erhöht, weil Sie Korrekturen im Code nur noch an einer Stelle umsetzen müssen. Die in diesem Beitrag beschriebenen Klassen sind wiederverwendbare Klassen für die Erstellung eines Filterausdrucks, die Sie ohne Änderung in Ihre Anwendung importieren und nutzen können.

Ziel

Das Erstellen von SQL-Filterausdrücken soll standardisiert und ohne hohen Programmieraufwand für unterschiedliche SQL-Dialekte anwendbar werden.

Ausgangssituation

SQL-Text zum Filtern von Formulardaten oder als Bestandteil von SQL-Anweisungen für Recordsets wird regelmäßig benötigt. Beim Zusammenstellen der SQL-Anweisungen muss auf die passende Konvertierung in Text geachtet werden. Die Konvertierung sowie das Zusammenfügen mehrerer Filter-Kriterien können zu schlecht lesbarem Code führen. Eine Sammlung von Hilfsprozeduren und Klassen soll diese Aufgaben vereinfachen.

Anforderungen

Die Lösung stellt die folgenden Anforderungen:

- Unterstützung für verschiedene SQL-Dialekte
- Unterschiedliche Einsatzmöglichkeiten - von der Konvertierung eines Wertes zu SQL-Text über das Erstellen eines einzelnen Kriteriums bis zur automatischen Filterung in Formularen, wenn Werte in Filter-Steuerelementen geändert werden.
- Definieren statt programmieren
- Übersichtlicher Code in den Prozeduren, die die Klassen verwenden
- Wiederverwendbarer Code

Lösungsansatz

Der Lösungsansatz beinhaltet die folgenden Elemente:

- Klassen mit konkreten Aufgaben (Single-Responsibility-Prinzip)
- Austauschbare Klassen – falls spezielle Anforderungen benötigt werden
- Anforderungen für unterschiedliche SQL-Dialekte über Parameter/Eigenschaften einstellbar

Klasse SqlTools

Die Klasse **SqlTools** stellt Methoden zum Erstellen eines SQL-Ausdrucks bereit. Die Methoden **TextToSqlText**, **NumberToSqlText**, **DateToSqlText** und **BooleanToSqlText** konvertieren Werte mit einem bestimmten Datentyp in einen SQL-konformen Text.

Damit diese Konvertierungsfunktion Text für verschiedene SQL-Dialekte erzeugen können, müssen Sie die gewünschten Formate als Parameter an die Konvertier-Funktionen weitergeben oder Sie stellen die Formate mit den Eigenschaften **SqlDateFormat**, **SqlBooleanTrueString** und **SqlWildCardString** für die Klassen-Instanz ein. Die Methode **ConvertToSqlText** bildet einen zentralen Einstiegspunkt dieser Konvertierungsfunktionen. In Listing 1 sehen Sie Auszüge aus der Beispiele-Datei **SqlTools_Beispiele.mdb**.

Zum Erstellen eines Filterausdrucks übergeben Sie den Datenfeldnamen, den Datentyp, den gewünschten

Vergleichsausdruck und den Wert an die Methode **BuildCriteria**. Innerhalb der Methode wird der Wert über die Konvertierungsfunktion für den jeweiligen Datentyp in SQL-Text umgeformt. Aus den **Enum**-Werten vom **RelationalOperator**-Parameter wird der Vergleichsausdruck erstellt.

Diese Texte werden mit dem Feldnamen zu einem Filterausdruck zusammengefügt und als Funktionsergebnis zurückgegeben. Die Beispiele in Listing 2 zeigen übliche Varianten für Vergleichsausdrücke.

Für **Between**-Filterbedingungen gibt es spezielle Features in der **SqlTools**-Klasse. Wenn Sie beispielsweise einen Datumsfilter erzeugen wollen und in

```
Private Sub DatentypToSqlText()
    [...]
    FilterWert = DateSerial(2015, 12, 24)
    SqlText = SqlTools.DateToSqlText(FilterWert, "\#yyyy-mm-dd\#")
    Debug.Print "Aus "; FilterWert; " wird "; SqlText
    ' --> Aus 24.12.2015  wird #2015-12-24#
    [...]
End Sub

Private Sub ConvertToSqlText_JetDaoDialekt()
    [...]
    'SQL-Formate einstellen
    With SqlTools
        .SqlDateFormat = "\#yyyy-mm-dd\#"
        .SqlBooleanTrueString = "True"
    End With
    [...]
    FilterWert = DateSerial(2015, 12, 24)
    SqlText = SqlTools.ConvertToSqlText(FilterWert, SQL_Date)
    Debug.Print "Aus "; FilterWert; " wird "; SqlText
    [...]
End Sub
```

Listing 1: Einsatz der Konvertierungsfunktionen

```
Private Sub BuildCriteria_EqualGreaterThan()
    [...]
    FilterWert = "O'Neill"
    SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, SQL_Equal + SQL_GreaterThan, FilterWert)
   Debug.Print "SQL_Text:", SqlText
   ' --> Textfeld >= 'O''Neill'
    [...]
End Sub

Private Sub BuildCriteria_Like()
    [...]
   FilterWert = "abc"
   SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, SQL_Like, FilterWert)
   Debug.Print "SQL_Text:", SqlText
   ' --> Textfeld Like 'abc'
   FilterWert = "a"
   SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, SQL_Like + SQL_Add_WildCardSuffix, FilterWert)
   Debug.Print "SQL_Text:", SqlText
   ' --> Textfeld Like 'a*'
   FilterWert = "c"
   SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, SQL_Like + SQL_Add_WildCardPrefix, FilterWert)
   Debug.Print "SQL_Text:", SqlText
   ' --> Textfeld Like '*c'
End Sub
```

Listing 2: **BuildCriteria**-Beispiele

```
Private Sub BuildCriteria_Between_Features()
    [...]
    'Datum + SQL_Add_WildCardSuffix:
    ErsterWert = DateSerial(2015, 12, 1)
    ZweiterWert = DateSerial(2015, 12, 31)
    SqlText = SqlTools.BuildCriteria("Datumsfeld", SQL_Date, _
        SQL_Between + SQL_Add_WildCardSuffix, ErsterWert, ZweiterWert)
    Debug.Print "SQL_Date + SQL_Add_WildCardSuffix: zwischen "; _
        ErsterWert; "und "; ZweiterWert; "ergibt:"; vbNewLine; SqlText
    ' --> Datumsfeld >= #2015-12-01# And Datumsfeld < #2016-01-01#
    'Nullwerte:
    ErsterWert = 1
    ZweiterWert = Null
    SqlText = SqlTools.BuildCriteria("Zahlenfeld", SQL_Numeric, _
        SQL_Between, ErsterWert, ZweiterWert)
    Debug.Print "ZweiterWert = Null: "; SqlText
    ' --> Zahlenfeld >= 1
    ErsterWert = Null
    ZweiterWert = 5
    SqlText = SqlTools.BuildCriteria("Zahlenfeld", SQL_Numeric, _
        SQL_Between, ErsterWert, ZweiterWert)
    Debug.Print "ErsterWert = Null: "; SqlText
    ' --> Zahlenfeld <= 5
End Sub
```

Listing 3: BuildCriteria: Between-Features

den zu filternden Werten auch die Uhrzeit enthalten ist, können Sie **SQL_Add_WildCardSuffix** verwenden, um den ganzen Tag des Ende-Datums anzugeben. Der **Between**-Ausdruck wird zu einem **Feld >= Startdatum And Feld < (EndDatum+1)**-Ausdruck umgeschrieben.

Übergeben Sie **Null** an einen der Wert-Parameter, wird der **Between**-Ausdruck wieder umgeformt, da **Null** standardmäßig als Filterwert nicht berücksichtigt werden soll. Beispiele zu diesen Features sehen Sie in Listing 3.

Wie Ihnen bereits bei der **Between**-Bedingung aufgefallen ist, kann man an die **BuildCriteria**-Methode mehr als einen Filterwert für eine Filterbedingung übergeben. Für einen **Between**-Ausdruck werden immer zwei Werte benötigt. Für einen **In**-Filterausdruck können Sie ein Array als Übergabewert verwenden.

Listing 4 zeigt neben der **In**-Variante noch eine weitere Variante, mit der Sie mehrere Filterwerte für ein Datenfeld in einer **BuildCriteria**-Zeile verwenden können. Wenn Sie ein Array mit Werten übergeben und als **RelationalOperator** nicht **SQL_In** verwenden, wird für jeden Array-Wert ein Filterausdruck erzeugt und diese Filterausdrücke werden mit **Or** verbunden.

Die bisher erwähnten Methoden und Eigenschaften sind für den Aufgabenbereich der Klasse ausreichend. In der Klasse sind noch zwei weitere Methoden enthalten, die die Verwendung der Klasse vereinfachen. Mit Clone können Sie eine vorhandene Instanz einschließlich der Formateinstellungen kopieren. NewInstance ermöglicht die Erstellung einer

```
Private Sub BuildCriteria_ArrayUebergabe()
    [...]
    ' In(...):
    SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, _
        SQL_In, Array("a", "c", "l"))
    Debug.Print "SQL_Text, In(...):", SqlText
    ' --> Textfeld In ('a','c','l')
    SqlText = SqlTools.BuildCriteria("Zahlenfeld", SQL_Numeric, _
        SQL_In, Array(1, 3, 5))
    Debug.Print "SQL_Numeric, In(...):", SqlText
    ' --> Zahlenfeld In (1,3,5)
    ' Mehrere Filter mit Or kombiniert
    SqlText = SqlTools.BuildCriteria("Textfeld", SQL_Text, _
        SQL_Like + SQL_Add_WildCardSuffix, Array("a", "c", "l"))
    ' --> Textfeld Like 'a*' Or Textfeld Like 'c*' Or Textfeld Like 'l*'
End Sub
```

Listing 4: BuildCriteria: Array-Übergabe

Kanban mit Outlook und Access

Haben Sie auch manchmal den Eindruck, dass Sie zu viele Projekte gleichzeitig beackern und den Überblick verlieren könnten? Mir passierte das regelmäßig. Bis ich über Kanban gestolpert bin und Teile des Konzepts in meine eigenen Abläufe eingebaut habe. Kanban soll durch eine Visualisierung der aktuellen Projekte und ihres Fortschritts eine bessere Übersicht bieten und vor allem die Anzahl der gleichzeitig begonnenen Projekte begrenzen. Dies baue ich in meine langjährige Praxis zur Zeiterfassung mit Outlook und Access ein. Wie das gelingt, zeigt der vorliegende Beitrag.

Eines der Probleme der heutigen Zeit ist, dass wir immer mehr unter zeitlichem Druck stehen und scheinbar immer mehr Aufgaben zu bewältigen haben. In der Tat sieht man in vielen Bereichen des Lebens, dass Aufgaben viel lieber aufgeschoben als erledigt werden. Ich muss selbst gestehen, dass ich manche unklare Leseranfrage lieber mit einer Gegenfrage beantworte, um diese erstmal vom Tisch zu haben. Tatsache ist: Die Frage kommt wieder zurück, und irgendwann muss ich mich ohnehin darum kümmern.

Die Anzahl der Aufgaben steigt allerdings kontinuierlich, solange man diese immer wieder in eine Warteschlange schiebt, statt sich abschließend um die Bearbeitung zu kümmern und die Aufgabe endlich von der Liste verschwinden lässt.

Die Lösung aus Outlook, Access und Kanban, die dieser Beitrag präsentiert, zeigt, wie ich zuletzt meine Projekte bearbeitet habe. Sie setzt auf einer Projektzeiterfassung auf, die ich bereits seit einigen Jahren nutze, um meine Arbeitszeiten zu protokollieren (die Basis dazu finden Sie im Beitrag **Projektzeiterfassung mit Outlook und Access**, **www.access-im-unternehmen.de/1016**). Dabei verwende ich Outlook, um Aufgabenordner synonym zu Projekten, Aufgaben zu Aufgaben und Termine zu Tätigkeiten einzutragen. Die Daten werden im Hintergrund in einer Access-Datenbank gespeichert. Was hat das mit Kanban zu tun, beziehungsweise mit der hier verwendeten Interpretation dieser Vorgehensweise?

Wir wollen den Kanban-Ansatz insoweit nutzen, als dass wir für unsere Aufgaben verschiedenen Status definie-

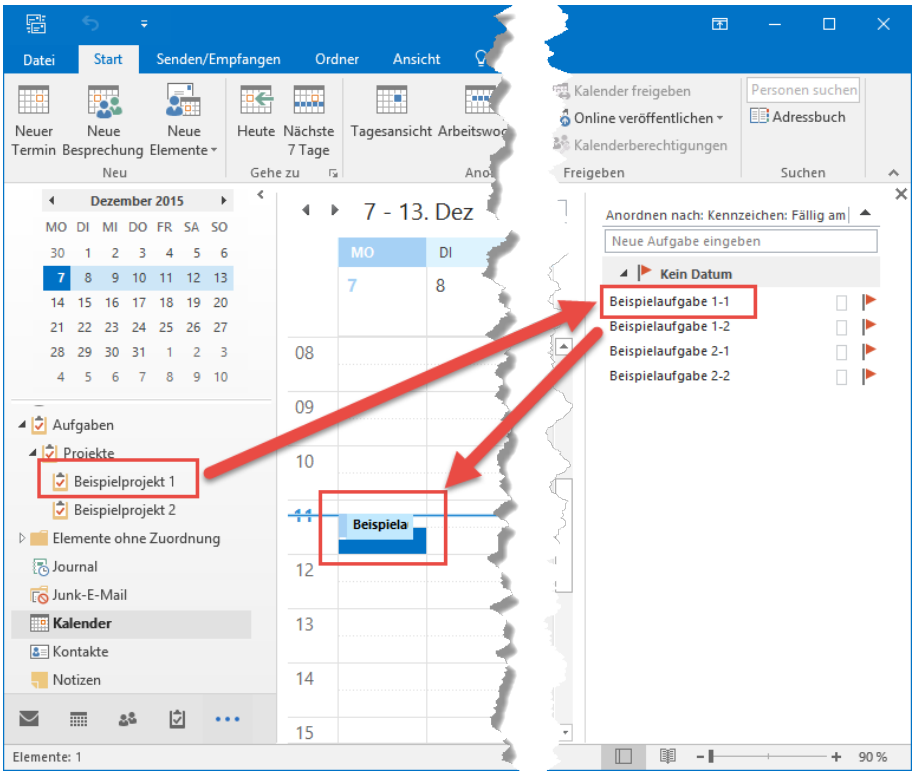


Bild 1: Bisherige Darstellung der Projekte und Aufgaben und Tätigkeiten in der Projektzeiterfassung

ren, die diese dann durchlaufen (ja, Status ist der Plural von Status – ich war selbst anfangs überrascht ...).

Kanban-Kategorien

Die bisherige Vorgehensweise bei der einfachen Variante der Projektzeiterfassung sieht wie in Bild 1 aus. Hier haben Sie unter dem Ordner **Projekte** neue Projekte angelegt (zum Beispiel **Beispielprojekt 1**), darin Aufgaben erstellt (**Beispielaufgabe 1**) und diese als Tätigkeiten in den Kalender gezogen.

Nun wollen wir in die Liste mit den Aufgaben auf der rechten Seite noch Kategorien einfügen. Das gelingt über die Benutzeroberfläche ganz einfach, indem Sie den Aufgaben jeweils eine neue Kategorie hinzufügen. Dazu wählen Sie aus dem Kontextmenü einer der Aufgaben den Eintrag **Kategorisieren** und die jeweilige Kategorie aus (s. Bild 2).

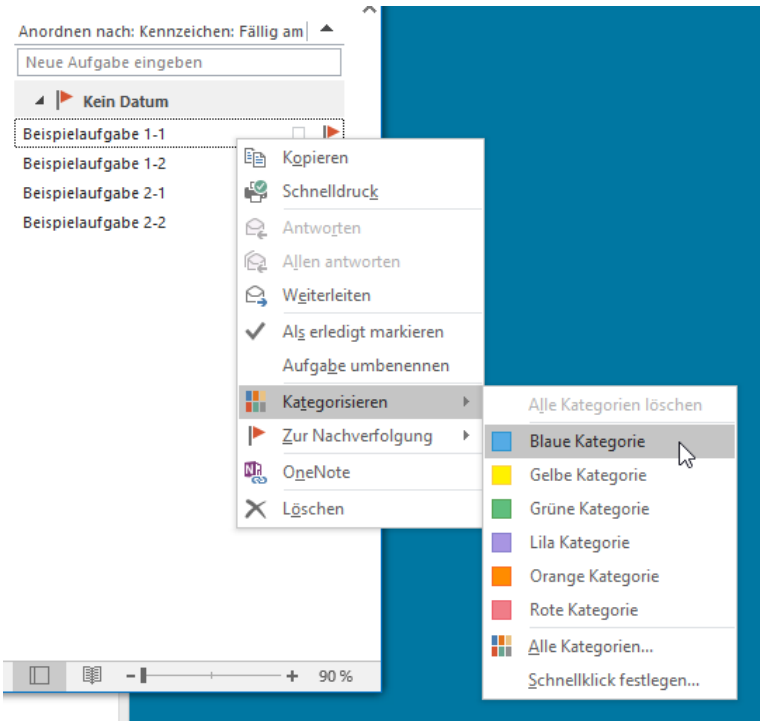


Bild 2: Hinzufügen einer Kategorie von Hand

Mit einem Klick auf den Eintrag **Alle Kategorien** öffnen Sie den Dialog aus Bild 3. Hier legen Sie mit einem Mausklick auf die Schaltfläche **Neu** weitere, benutzerdefinierte Kategorien an.

Nachdem Sie die gewünschten Kategorien angelegt und diese den Aufgaben zugewiesen haben, erhalten Sie das Bild aus Bild 4. Dies reicht uns allerdings noch nicht: Die Aufgaben sollen ja auch noch nach Kategorien gruppiert und möglichst mit der Kategorie als Gruppenüberschrift

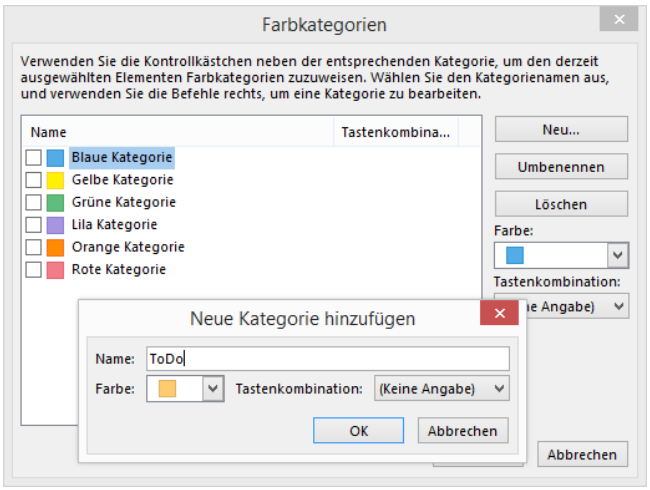


Bild 3: Anlegen einer neuen Kategorie

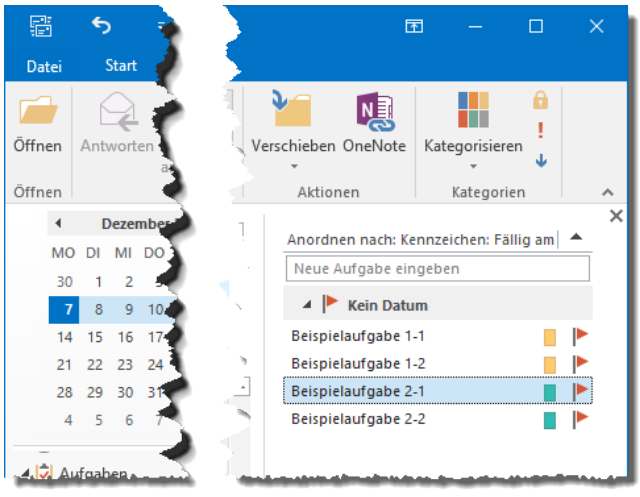


Bild 4: Aufgaben mit zugeordneten Kategorien

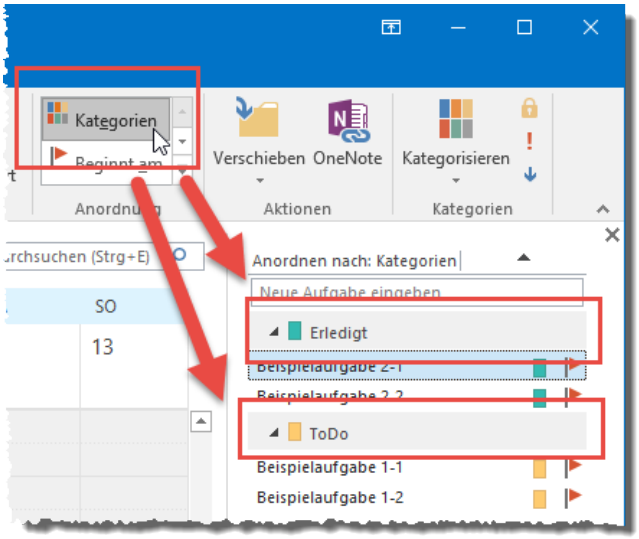


Bild 5: Einblenden der Kategorieüberschriften

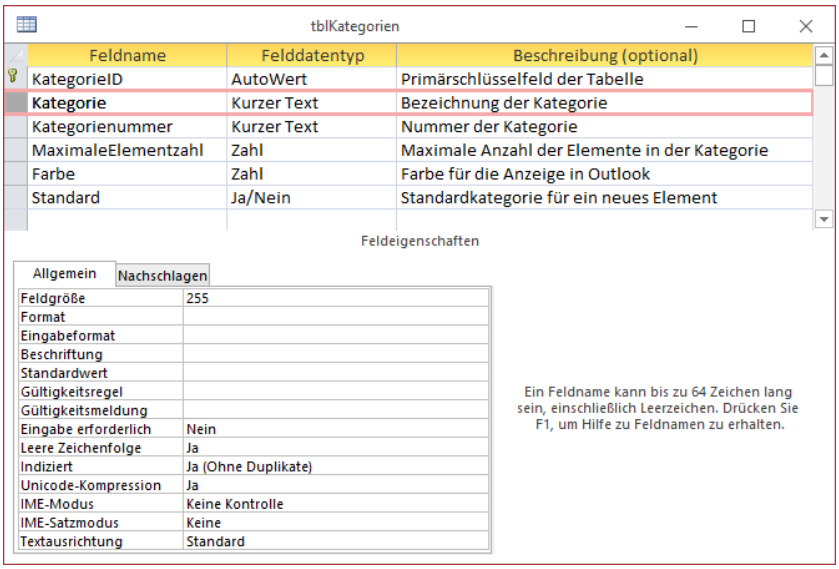


Bild 6: Tabelle zum Speichern der Kategorien

dargestellt werden. Um dies zu erreichen, ist allerdings nur noch ein kleiner Schritt nötig.

Dazu klicken Sie im Ribbon einfach auf den Eintrag **AufgabenlistelAnordnunglKategorien**. Dies blendet die Kategorien als Überschriften ein (s. Bild 5). Nun wollen wir die Kategorien natürlich nicht von Hand anlegen, sondern in einer Tabelle unserer Datenbank zur Projektzeiterfassung speichern und diese beim Start von Outlook dynamisch einlesen und anzeigen.

Speichern der Kategorien

Dazu wollen wir zunächst eine entsprechende Tabelle anlegen. Diese heißt **tblKategorien** und sieht im Entwurf wie in Bild 6 aus. Das Feld **Kategorie** ist mit einem eindeutigen Index versehen, damit jeder Kategoriename nur einmal eingegeben werden kann. Die Kategorienummer soll gegebenenfalls dem Kategorienamen vorangestellt werden können, damit die gewünschte Sortierung der Kategorien in der Aufgabenliste möglich ist.

Das Feld **MaximaleElementzahl** nimmt die Anzahl der maximal in einer Kategorie enthaltenen Elemente aus. Dies ist einer der Kanban-Grundsätze: Für bestimmte Bearbeitungsschritte soll man nur eine begrenzte Anzahl von Elementen vorsehen können. Damit verhindere ich beispielsweise, dass ich mehr als zwei Artikel gleichzeitig schreibe – oder an der Programmierung der Beispieldatenbanken von mehr als zwei Artikeln arbeite. Wenn man in Outlook mehr als die vorgebenene Anzahl Elemente zu einer Kategorie hinzufügt, soll dies eine entsprechende Meldung auslösen.

Das Feld **Farbe** legt den Index der Farbe für die Kategorie fest. Das Feld **Standard** legt fest, welche Kategorie einer neu erstellten Aufgabe zugeordnet werden soll. Das gleich vorgestellte Formular zur Bearbeitung der Kategorien soll sicherstellen, dass nur eine Kategorie gleichzeitig als Standard einstellt werden kann.

Verwalten der Kategorien

Für die Verwaltung der Kategorien erstellen wir ein kleines Access-Formular in der Datenbank zur Projektzeiterfassung. Dieses verwendet die Tabelle **tblKategorien** als Datenherkunft. Mit einem Listenfeld namens **lstKategorien** sollen die vorhandenen Einträge ausgewählt werden können (s. Bild 8).

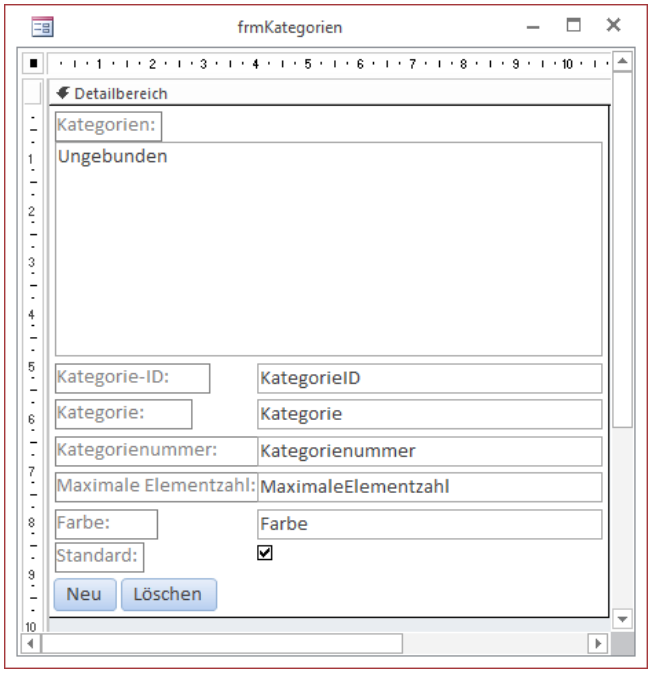


Bild 8: Dialog zum Verwalten der Kategorien in der Entwurfsansicht

Das Listenfeld verwendet die Abfrage aus Bild 7 als Datensatzherkunft. Diese liefert in eckigen Klammern die Nummer der Kategorie plus den Kategorienamen, also etwa so: **[00] ToDo**. Zusätzlich sortiert sie die Datensätze nach dem Wert des Feldes **Kategorienummer**.

Damit das Listenfeld nicht die gebundene Spalte mit dem Feld **KategorieID**, sondern nur das Feld **Kategoriename** der Abfrage anzeigt, stellen wir die Eigenschaft **Spaltenanzahl** auf den Wert **2** und **Spaltenbreiten** auf **0** ein.

Das Formular zeigt, wie üblich, standardmäßig den ersten Datensatz der Datenherkunft an. Damit das Listenfeld den gleichen Eintrag markiert, legen Sie für das Ereignis **Beim Laden** des Formulars die folgende Ereignisprozedur an:

```
Private Sub Form_Load()
    Me!lstKategorien = Me!KategorieID
End Sub
```

Andersherum soll das Formular immer den aktuell im Listenfeld ausgewählten Eintrag anzeigen. Dazu verwenden wir eine weitere Ereignisprozedur, die durch das Ereignis **Nach Aktualisierung** des Listenfeldes ausgelöst wird. Diese enthält nur einen einzigen Befehl, der das Recordset des Formulars auf den Datensatz mit dem Wert im Feld **KategorieID** sucht, der gerade im Listenfeld markiert ist:

```
Private Sub lstKategorien_AfterUpdate()
    Me.Recordset.FindFirst "KategorieID = " _
        & Me!lstKategorien
End Sub
```

Nun soll der Benutzer natürlich auch neue Kategorien anlegen können. Dies erledigen wir über eine Schaltfläche namens **cmdNeu**, die folgende Ereignisprozedur auslöst und so den Datensatzzeiger des Formulars auf einen neuen, leeren Datensatz verschiebt:

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

Damit der Benutzer nicht über das Formular von Datensatz zu Datensatz wechseln kann, sondern nur über die Auswahl eines Eintrags im Listenfeld, stellen Sie die Eigenschaft **Zyklus** des Formulars auf **Aktueller Datensatz** ein. Deshalb benötigen wir auch eine eigene Schaltfläche,

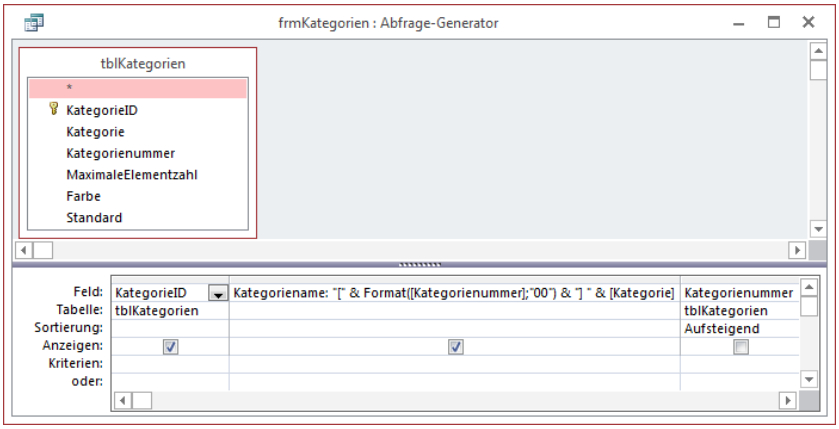


Bild 7: Abfrage für die Datensatzherkunft des Listenfeldes **lstKategorien**

damit der Benutzer einen neu hinzugefügten Datensatz speichern kann, ohne anderweitig zu einem anderen Datensatz zu wechseln. Diese löst die folgende Prozedur aus:

```
Private Sub cmdSpeichern_Click()
    RunCommand acCmdSaveRecord
End Sub
```

Nun kann der Speichervorgang natürlich auch durch andere Aktionen ausgelöst werden, beispielsweise durch das Wechseln des Datensatzes über die Auswahl eines anderen Eintrags im Listenfeld oder auch durch das Schließen des Formulars. Dies fangen wir mit dem Ereignis **Nach Aktualisierung** ab, das wir wie folgt implementieren:

```
Private Sub Form_AfterUpdate()
    Me!lstKategorien.Requery
    Me!lstKategorien = Me!KategorieID
End Sub
```

Die Prozedur aktualisiert die Datensatzherkunft des Listenfeldes und stellt dieses auf den aktuell im Formular ausgewählten Datensatz ein. Schließlich wollen wir auch noch Datensätze löschen können, was wir mit der Schaltfläche **cmdLoeschen** realisieren. Diese löst folgende Prozedur aus:

```
Private Sub cmdLoeschen_Click()
    If IsNull(DLookup("AufgabeID", "tblAufgaben", _
            "KategorieID = " & Me!KategorieID)) Then
        RunCommand acCmdDeleteRecord
        Me!lstKategorien.Requery
        Me!lstKategorien = Me!KategorieID
    Else
        MsgBox "Die Kategorie kann nicht gelöscht werden, _
                    da sie bereits Einträge enthält."
    End If
End Sub
```

Die Prozedur prüft, ob die zu löschende Kategorie bereits einer Aufgabe zugewiesen ist. Ist dies der Fall, wird der Löschvorgang nicht durchgeführt, sondern es erscheint

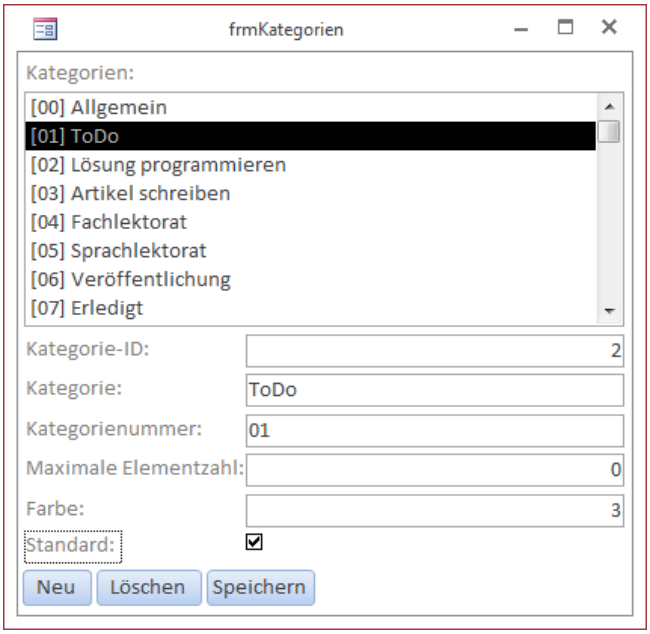


Bild 9: Verwalten der Kategorien per Formular

eine entsprechende Meldung. Anderenfalls löscht die Prozedur den aktuellen Datensatz, aktualisiert die Datensatzherkunft des Listenfeldes und markiert dort den aktuell im Formular angezeigten Datensatz. Das Formular sieht schließlich wie in Bild 9 aus.

Initialisieren der Kategorieansicht

Nun wollen wir dafür sorgen, dass alle in der Tabelle gespeicherten Tabellen in Outlook vorhanden sind und mit den entsprechenden Farben angezeigt werden. Dazu benötigen wir eine Prozedur, die beim Starten von Outlook aufgerufen wird. Im Beitrag **Projektzeiterfassung mit Outlook und Access** haben Sie ja bereits die Ereignisprozedur **Application_Startup** kennen gelernt, die beim Start von Outlook automatisch ausgelöst wird. Dieser fügen wir den Aufruf unserer kleinen Routine zum Kontrollieren und Hinzufügen der Kategorien hinzu:

```
Public Sub Application_Startup()
    ...
    KategorienKontrollieren
    ...
End Sub
```

```
Public Sub KategorienKontrollieren()
    Dim rst As DAO.Recordset
    Dim objCategory As Outlook.Category
    Dim objCategories As Outlook.Categories
    Dim bolKategorie As Boolean
    Dim lngFarbe As Long
    Set rst = CurrentDBC.OpenRecordset("SELECT * FROM tblKategorien", dbOpenDynaset)
    Set objCategories = GetMAPINamespace.Categories
    Do While Not rst.EOF
        bolKategorie = False
        lngFarbe = 0
        For Each objCategory In objCategories
            If objCategory.Name = rst!Kategorienummer & "_" & rst!Kategorie Then
                bolKategorie = True
                Exit For
            End If
        Next objCategory
        If bolKategorie = True Then
            If IsNull(rst!Farbe) Then
            rst.Edit
            rst!Farbe = objCategory.Color
            rst.Update
            End If
        Else
            If Not IsNull(rst!Farbe) Then
                lngFarbe = rst!Farbe
            End If
            Set objCategory = objCategories.Add(rst!Kategorienummer & "_" & rst!Kategorie, lngFarbe)
        End If
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Prüfen und Aktualisieren der Kategorien in Outlook

Die Prozedur **KategorienKontrollieren** selbst finden Sie im Modul **mdlKanban** (s. Listing 1). Die Prozedur verwendet ein **Recordset**-Objekt namens **rst**, um alle Datensätze der Tabelle **tblKategorien** zu durchlaufen. Die **Property Get**-Prozedur **CurrentDBC** aus dem Modul **mdlGlobal** liefert einen Verweis auf das **Database**-Objekt der Datenbank **Projektzeiterfassung.accdb**, deren Name in einer Konstanten im gleichen Modul verankert ist. Weiterhin füllt die Prozedur das Objekt **objCategories** mit einer Liste der Kategorien des aktuellen **Namespace**-Objekts von Outlook, das die Funktion **GetMAPINamespace** liefert.

Jede Kategorie wird in Outlook nämlich in einem eigenen **Category**-Objekt vorgehalten. Nun durchläuft die Prozedur in einer äußeren **Do While**-Schleife alle Datensätze der Tabelle **tblKategorien**, also alle Kategorien, die eigentlich in Outlook sichtbar sein sollten. Dabei stellt sie die Variablen **bolKategorie** und **lngFarbe** auf die Werte **False** und **0** ein, um diese für den aktuellen Schleifendurchlauf zu initialisieren. Innerhalb der Schleife durchläuft sie dann alle **Category**-Objekte der Auflistung in **objCategories** und referenziert das aktuelle Objekt in der Variablen **objCategory**. Sollte der Name der Kategorie aus **objCategory.**

Rund um das Ribbon

Das Ribbon und seine Programmierung enthält immer noch viele Geheimnisse für uns Access-Entwickler. Dieser Beitrag zeigt einige Techniken, mit denen Sie oft angefragte Konstellationen erhalten – beispielsweise das Ein- und Ausblenden zusätzlicher Ribbon-Elemente mit der Anzeige bestimmter Formulare oder auch das Aktivieren und Deaktivieren von Steuerelementen oder kompletter Ribbon-Bereiche in Abhängigkeit vom Vorhandensein bestimmter Elemente der Benutzeroberfläche.

Voraussetzungen

Für die nachfolgenden Beispiele benötigen Sie ein paar Elemente:

Die Tabelle **USysRibbons** aus Bild 1 nimmt die Ribbon-Definitionen auf. Sie enthält drei Felder: **ID** ist das Primärschlüsselfeld, **RibbonName** speichert den Namen der Ribbon-Definition, der auch in der Eigenschaft **Name des Menübands** etwa von Formularen erscheint, und **RibbonXML** schließlich liefert die eigentliche Ribbon-Definition im XML-Format.

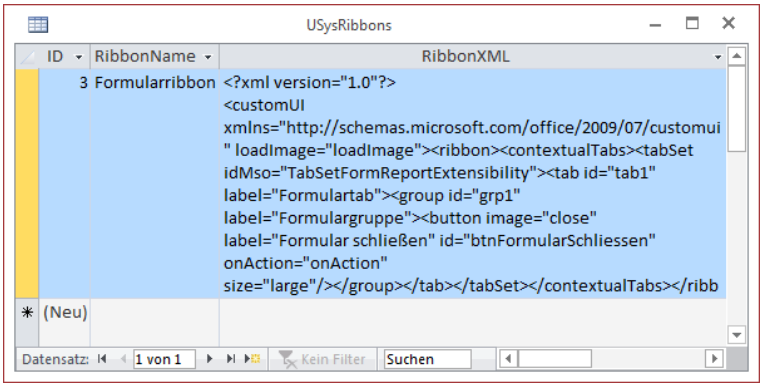


Bild 1: Die Tabelle **USysRibbons** speichert die Ribbon-Definitionen der Anwendung.

Die Tabelle **MSysResources** speichert die Bilddateien, die im Ribbon angezeigt werden sollen (entweder in der Größe 16x16 oder 32x32) in einem Anlagefeld (s. Bild 2). Den Wert des Feldes **Name** benötigen Sie, wenn Sie eines der hier gespeicherten Bilder im Ribbon anzeigen möchten.

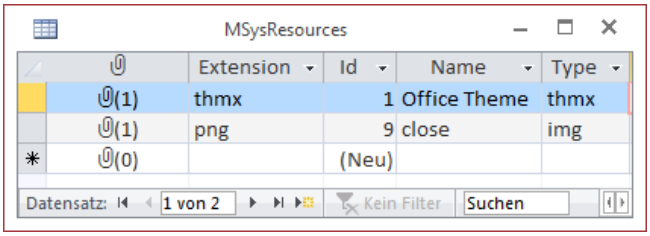


Bild 2: Die Tabelle **MSysResources** speichert Bilddateien.

Das Modul **mdlRibbonImages** enthält einige Funktionen rund um die Anzeige von Bildern im Ribbon, vor allem aber die Funktion **PicFromSharedResource_Ribbon**. Diese erwartet den Namen einer Bilddatei aus der Tabelle **USysResources**, also den Wert des Feldes **Name** der gewünschten Bilddatei.

Damit Bilder im Ribbon angezeigt werden, müssen Sie dem Attribut **loadImage** des **customUI**-Elements den Wert **loadImage** zuweisen (**loadImage="loadImage"**). Diese Funktion wird immer dann aufgerufen, wenn das Attribut **image** eines Ribbon-Elements den Namen eines Bildes enthält, also zum Beispiel **close**.

Die Funktion **loadImage** finden Sie im Modul **mdlRibbon**, wo wir gleich auch weitere Callback-Funktionen eintragen, die etwa beim Anklicken eines Ribbon-Elements ausgelöst werden.

Außerdem benötigen Sie einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** für die Verwendung einiger VBA-Elemente für den Zugriff auf das Ribbon.

Ribbon-Definition mit Formular einblenden

Wenn Sie eine Ribbon-Definition erstellen, die formularabhängige Elemente enthält, müssen Sie einfach nur

den Namen der in der Tabelle **USysRibbons** gespeicherten Definition als Wert der Eigenschaft **Name des Menübands** für das Formular hinterlegen. Wenn Sie dann noch das Attribut **startFromScratch** des Elements **customUI** auf **True** einstellen, blendet die Ribbon-Definition alle eingebauten Ribbon-Elemente aus und zeigt nur noch die benutzerdefinierten Elemente an.

Dies sieht dann beispielsweise wie in Bild 3 aus. Das Ribbon mit der Schaltfläche **Formular schließen** als einzigem Element wird eingeblendet, wenn der Benutzer das Formular **frmFormularMitEigenerDefinition** öffnet.

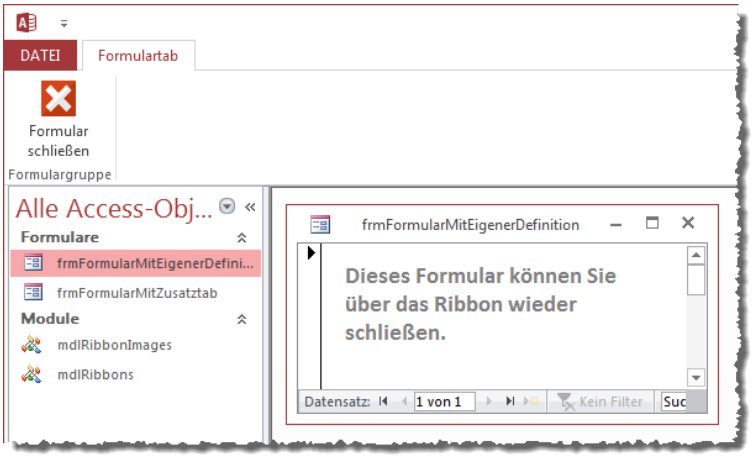


Bild 3: Beispiel für ein Ribbon-Tab, das zusammen mit einem Formular eingeblendet wird

Die Ribbon-Definition für dieses Beispiel finden Sie in Listing 1. Hier beginnen wir mit dem Element **customUI**, das für das Attribut **loadImage** den Wert **loadImage** enthält. Dies ist wichtig, damit später auftauchende **image**-Attribute so verarbeitet werden, dass die Funktion **loadImage** im Modul **mdlRibbon** aufgerufen wird und so die Bilddatei aus der Tabelle **USysResources** einliest und im Ribbon anzeigt.

Das **ribbon**-Element enthält das Attribut **startFromScratch** mit dem Wert **true**, was dafür sorgt, dass alle eingebauten Elemente des Ribbons ausgeblendet werden.

Das **tabs**-Element fasst ein oder mehrere **tab**-Elemente zusammen, die den Registerreitern im Ribbon entsprechen. Das einzige **tab**-Element ist dann auch das mit der Beschriftung **Formulartab** aus der Abbildung. Gleiches gilt für das **group**-Element, das im Wesentlichen die Benennung der Gruppe beisteuert.

Fehlt noch die Schaltfläche mit dem Wert **btnFormularSchliessen** für das Attribut **id**. Der Wert **close** für das Attribut **image** sorgt dafür, dass das Bild namens **close** aus der Tabelle **USysResources** geladen wird. Dieses wird nach dem Laden groß dargestellt, also mit einer Auflösung von 32x32 Pixeln (**size="large"**). Schließlich soll ein Klick auf die Schaltfläche die VBA-Funktion **onAction** auslösen

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab1" label="Formulartab">
        <group id="grp1" label="Formulargruppe">
          <button id="btnFormularSchliessen" image="close" label="Formular schließen" onAction="onAction" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 1: Ribbon-Definition für ein Ribbon, das bei Anzeige seines Formulars alle anderen Elemente ausblendet

(**onAction="onAction"**). Diese Funktion sieht so aus:

```
Sub onAction(control As IRibbonControl)
    Select Case control.ID
        Case Else
            DoCmd.Close acForm, Screen.Active-
Form.Name
    End Select
End Sub
```

Die Callback-Funktion enthält eine **Select Case**-Bedingung, weil sie durch verschiedene Schaltflächen ausgelöst werden soll. In unserem Fall soll diese standardmäßig das aktuelle Formular schließen, was die Anweisung **DoCmd.Close acForm, Screen.ActiveForm.Name** zuverlässig erledigt.

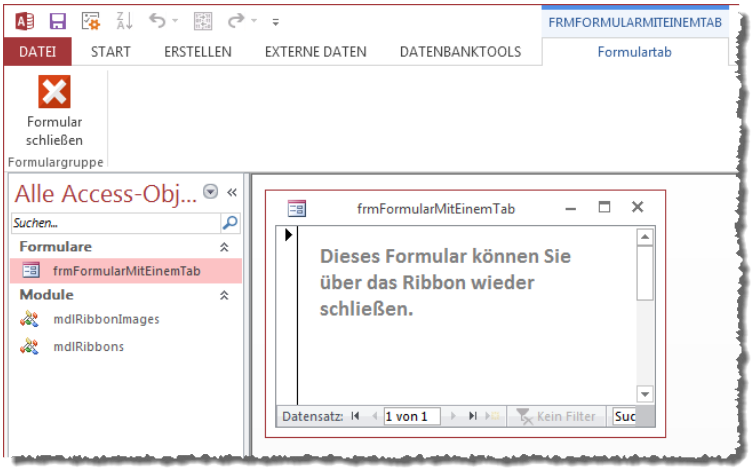


Bild 4: Beispiel für ein kontextabhängiges Ribbon-Tab

Ribbon-Tab mit Formular einblenden

Sie können auch, wie es etwa bei der Datenblattansicht der Fall ist, einfach ein zusätzliches Tab einfügen, das speziell gekennzeichnet und direkt im Vordergrund angezeigt wird – ein sogenanntes kontextabhängiges **tab**-Element.

Dies sieht dann beispielsweise wie in Bild 4 aus.

Die notwendige Ribbon-Definition finden Sie in Listing 2. Statt des **tabs**-Elements verwenden wir hier das **contextualTabs**-Element, dem wiederum ein **tabSet**-Element untergeordnet ist.

Erst darunter geht es wie gewohnt weiter – und zwar mit dem **tab**-Element, dem **group**-Element und dem **button**-Element (und weiteren Elementen, je nach Ihren Anforderungen). Das **tabSet**-Element müssen Sie unbedingt mit dem Wert **TabSetFormReportExtensibility** für das Attribut **idMso** ausstatten. Dies gibt an, dass danach benutzerdefi-

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" loadImage="loadImage">
  <ribbon>
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="tab1" label="Formulartab">
          <group id="grp1" label="Formulargruppe">
            <button image="close" label="Formular schließen" id="btnFormularSchliessen" onAction="onAction"
              size="large"/>
          </group>
        </tab>
      </tabSet>
    </contextualTabs>
  </ribbon>
</customUI>
```

Listing 2: Ribbon-Definition für ein Ribbon, das mit einem Formular eingeblendet wird

nierte Elemente folgen, die beim Aktivieren eines Formulars oder Berichts erscheinen, dessen Eigenschaft **Name des Menübands** auf den Namen der Ribbon-Definition eingestellt ist.

Dummerweise wird das zusätzliche Tab nur dann direkt in den Vordergrund geholt, wenn Sie das Formular zum ersten Mal öffnen. Außerdem stört noch etwas die übergeordnete Beschriftung, die sich scheinbar nach dem Formularnamen richtet (hier **FRMFORMULARMITZUSATZTAB**).

Dies lässt sich leicht ändern: Wir tragen einfach den Wert **Formular mit Zusatztab** in die Eigenschaft **Beschriftung** des Formulars **frmFormularMitZusatztab** ein, um die Überschrift zu ändern.

Fehlt allerdings noch die direkte Aktivierung auch nach dem ersten Öffnen des Formulars. Dazu ist ein kleiner Trick erforderlich: Sie müssen die Ribbon-Definition beim Erstellen mit einer entsprechenden VBA-Objektvariable referenzieren und können das gewünschte **tab**-Element dann nach Öffnen des Formulars zuverlässig in den Vordergrund holen – auch nach dem ersten Öffnen des Formulars, wo dies automatisch geschieht.

Dazu sind folgende Schritte nötig:

- Hinzufügen des Attributs **onLoad** zum Element **customUI** mit dem Wert **OnLoad_Formularribbon**
- Deklarieren einer Variablen namens **objRibbon_Formularribbon** des Typs **IRibbonUI**, welche den Verweis auf die Ribbon-Anpassung speichert im Modul **mdlRibbons**
- Anlegen einer Callback-Funktion namens **OnLoad_Formularribbon**, die beim Laden des Ribbons ausgelöst wird und die Variable **objRibbon_Formularribbon** mit dem Verweis auf die Ribbon-Erweiterung füllt (dies ist der einzige Zeitpunkt, an dem dies möglich ist)
- Anlegen einer Ereignisprozedur im Formular, welche das benutzerdefinierte **tab**-Element der Ribbon-Definition nach dem Öffnen in den Vordergrund holt

Die Ribbon-Definition passen Sie wie folgt an und ergänzen dabei das Attribut **onLoad** des **customUI**-Elements mit dem Wert **OnLoad_Formularribbon**:

```
<?xml version="1.0"?>
<customUI xmlns="..." onLoad="OnLoad_Formularribbon"
   loadImage="loadImage">
  <ribbon>
    <contextualTabs>
    ...
    </contextualTabs>
  </ribbon>
</customUI>
```

Die Deklaration der Variablen sieht wie folgt aus:

```
Public objRibbon_Formularribbon As IRibbonUI
```

Die Prozedur, die durch das Ereignis **onLoad** der Ribbon-Anpassung ausgelöst wird, sieht so aus:

```
Sub onLoad_Formularribbon(ribbon As IRibbonUI)
    Set objRibbon_Formularribbon = ribbon
End Sub
```

Sie speichert lediglich den mit dem Parameter **ribbon** gelieferten Verweis auf die Ribbon-Anpassung in der Variablen **objRibbon_Formularribbon**.

Nun müssen wir noch dafür sorgen, dass unser gewünschtes **tab**-Element beim Öffnen des Formulars aktiviert wird.

In einem ersten Versuch würden wir dazu die folgende Prozedur nutzen, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird und mit der Methode **ActivateTab** das **tab**-Element namens **tabForm** aktivieren soll: