HAUFE.
Ausgabe 06/2025

ACCESS
IM UNTERNEHMEN

LISTVIEW-CONTROL VON A-Z

Lernen Sie das ListView-Steuerelement als Alternative zum Listenfeld kennen (ab Seite 7).

In diesem Heft:

OBJEKTNAMEN-AUTOKORREKTUR

So funktioniert die automatische Übernahme von Änderungen an Tabellen und Feldern in andere Objekte.

SEITE 2

E-MAILS MIT CDO UND GMAIL

Verschicken Sie E-Mails mit der CDO-Bibliothek über Ihren Gmail-Account.

SEITE 59

DATEN ANONYMISIEREN PER ACCESS-ADD-IN

Anonymisieren Sie die Daten Ihrer Datenbank mit unserer praktischen Access-Erweiterung.

SEITE 64

Mat-Nr. 01583-5152

Alternative Liste

Nein, nicht die grüne alternative Liste – sondern eine Alternative zum Listenfeld von Access stellen wir Ihnen in dieser Ausgabe im Detail vor. Genau genommen geht es um das ListView-Steuerelement, das wesentlich mehr Features bietet als das eingebaute Listenfeld: Anzeigen von Icons, einfaches Sortieren nach Spalteninhalten und Drag and Drop-Funktionen sind nur einige davon. Aber alles hat seinen Preis: Im Gegensatz zum Listenfeld kann man das ListView-Steuerelement nicht einfach an eine Tabelle oder Abfrage binden, sondern muss die Daten per VBA aus der Tabelle auslesen und zum Steuerelement hinzufügen.

Aber wer Wert auf Funktionalität gepaart mit einer hübschen Optik legt, sollte sich die Beiträge dieser Ausgabe von Access im Unternehmen samt der Beispieldatenbanken einmal zu Gemüte führen. Im ersten Beitrag zum Thema stellen wir unter dem Titel **Listen anzeigen mit dem ListView-Steuerelement** ab Seite 7 die Grundlagen zu diesem Steuerelement vor: Welche Eigenschaften hat es, wie fügen wir einfache Daten hinzu, wie können wir sein Aussehen beeinflussen?

Weiter geht es mit **ListView-Steuerelement mit VBA programmieren** ab Seite 19, wo wir das Steuerelement unter dem Aspekt der VBA-Programmierung beleuchten. Hier gehen wir sehr in die Tiefe und zeigen die Grundlagen zum Sortieren, Umbenennen von Inhalten, Drag and Drop und vieles mehr.

Zur praktischen Umsetzung kommen wir dann ab Seite 37 im Beitrag **m:n-Beziehung mit Drag and Drop per ListView**. Hier zeigen wir am Beispiel einer Newsletter-Verteilerliste, wie wir einen Newsletter über zwei Listenfelder die Empfänger verwalten. Im Mittelpunkt stehen die Drag and Drop-Möglichkeiten des **ListView**-Steuerelements, denn zum Zuweisen eines Empfängers zu einem Newsletter verschieben wir diesen von der Liste der Nicht-Empfänger in die Empfängerliste – und umgekehrt.

Damit die Optik nicht zu kurz kommt, beleuchten wir im Beitrag **Icons per ImageList und VBA im ListView-Control** ab Seite 47 im Detail, wie wir auf einfache Weise Icons zu einem **ListView**-Steuerelement hinzufügen können. Besonderes Augenmerk legen wir dabei darauf, die Icons aus einer Systemtabelle von Access zu nutzen, die auch die Icons für andere Elemente wie Bild-Steuerelemente oder Schaltflächen aufnimmt.

Schließlich wollen wir nicht vorenthalten, wie man Daten aus Tabellen im **ListView**-Steuerelement anzeigt, was wir am Beispiel einer Mitarbeiterliste erledigen – siehe **ListView aus Tabellen oder Abfragen füllen**, Teil 1 ab Seite 52.

Neben dem **ListView**-Steuerelement beschäftigen wir uns mit der Funktion, die Änderungen an Tabellen- oder Feldnamen automatisch an abhängige Objekte wie Formulare oder Berichte weitergibt – und zwar im Beitrag **Objektnamen-Autokorrektur in Access nutzen** ab Seite 2.

Wer sich allmählich von Outlook verabschiedet und seine E-Mails mit Gmail versenden möchte, findet die passenden Techniken ab Seite 59 im Beitrag **E-Mails senden mit CDO und Gmail**.

Und schließlich zeigen wir, wir man mit wenigen Mausklicks die Daten seiner Datenbank anonymisiert – siehe **Daten anonymisieren per Access-Add-In** ab Seite 64.

Herzlichst, Ihr André Minhorst

Objektnamen-Autokorrektur in Access nutzen

Microsoft Access bietet eine praktische Funktion namens Objektnamen-Autokorrektur. Sie sorgt dafür, dass Änderungen an den Namen von Tabellen oder Tabellenfeldern automatisch auf andere Elemente übertragen werden, welche diese Tabellen oder Felder referenzieren. In diesem Beitrag schauen wir uns an, wie die Objektnamen-Autokorrektur aktiviert wird, wie sie funktioniert und warum sie nicht in jeder Phase des Lebenszyklus einer Access-Anwendung sinnvoll ist.

Welches Problem löst die Objektnamen-Autokorrektur?

Wenn wir in Access beispielsweise ein Formular hinzufügen, dass die Daten einer Tabelle anzeigen soll, fügen wir diesem Formular die Tabelle (oder eine darauf basierende Abfrage) als Datensatzquelle hinzu und ziehen die benötigten Felder in den Formularentwurf.

Das Formular zeigt nun für jeden Datensatz die Werte der entsprechenden Felder der Datensatzquelle an.

Sollten wir nun aus irgendeinem Grund nachträglich den Namen eines der Felder der zugrunde liegenden Datenbank ändern, ohne dass die Objektnamen-Autokorrektur aktiviert ist, finden wir beim nächsten Öffnen des Formulars den Fehler aus Bild 1 vor.

Wenn wir dieses Feld nicht nur in einem Formular genutzt haben, sondern in mehreren – und gegebenenfalls auch noch in Abfragen und Berichten, müssen wir Eigenschaften wie Steuerelementinhalt an allen relevanten Stellen anpassen.

Wie löst die Objektnamen-Autokorrektur dieses Problem?

Hier kommt die Objektnamen-Autokorrektur ins Spiel. Diese ist gerade bei neu angelegten Access-Datenbanken nicht umsonst standardmäßig aktiviert. Gerade wenn man beginnt, eine Anwendung zu programmieren, gibt es öfter Änderungen am Datenmodell und speziell auch an den Namen der Tabellenfelder.

Und genau diese registriert die Objektnamen-Autokorrektur. Sie prüft dann, ob es bereits Objekte gibt, deren Felder über verschiedene Eigenschaften wie beispielsweise Steuerelementinhalt auf ein geändertes Feld verweisen.

Was ist das Problem der Objektnamen-Autokorrektur?

Wenn man die ersten Suchergebnisse zum Begriff **Objektnamen-Autokorrektur** betrachtet, fällt zuerst einmal auf, dass hier oft von Abschalten die Rede ist und nicht von der Funktionsweise selbst. Das liegt daran, dass Performance im Kontext von Access ein wichtiger Faktor ist und die Entwickler an allen denkbaren Schrauben drehen.

Die Objektnamen-Autokorrektur scheint ständig zu beobachten, ob es irgendwelche Änderungen am Entwurf des

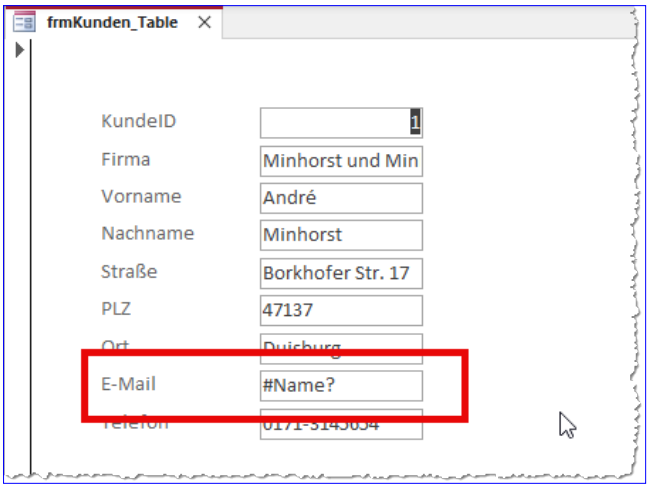


Bild 1: Fehler nach dem Ändern eines Tabellenfeldnamens ohne Verwendung der Objektnamen-Autokorrektur

Datenmodells gibt, und das geht zu Lasten der Performance.

Sobald die Anwendung jedoch ein Stadium erreicht hat, an dem nicht mehr viele Änderungen zu erwarten sind, kann man die Objektnamen-Autokorrektur abschalten. Das ist insbesondere der Fall, wenn die Anwendung bereits an den Benutzer ausgeliefert ist.

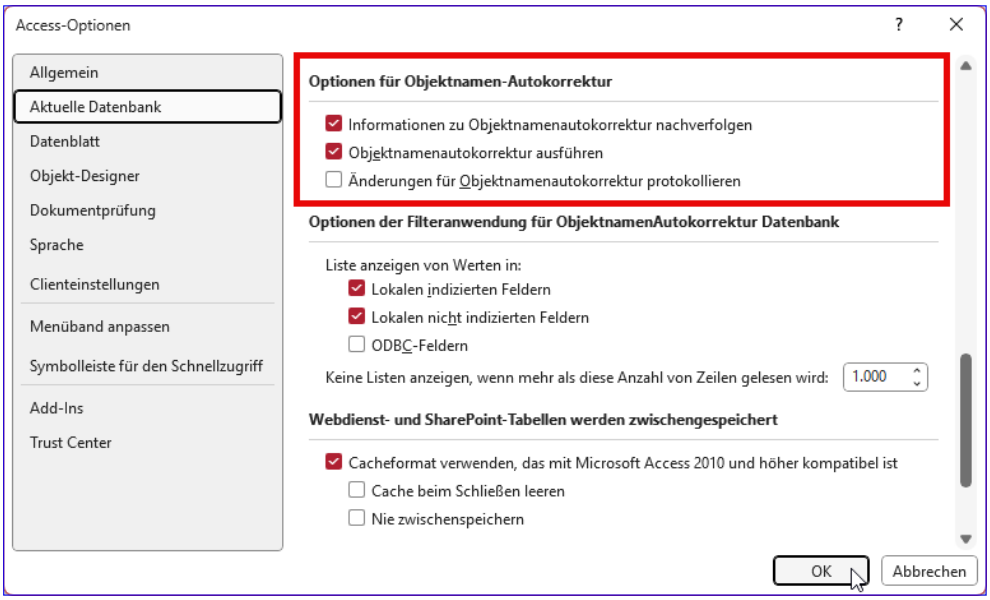


Bild 2: Optionen für die Objektnamen-Autokorrektur

Vor der Auslieferung sollten wir die Objektnamen-Autokorrektur also auf jeden Fall abschalten.

Objektnamen-Autokorrektur aktivieren und deaktivieren

Wenn wir eine neue, leere Access-Datenbank erstellen und für diese die Access-Optionen öffnen, sehen wir im Bereich **Aktuelle Datenbank** den Abschnitt **Optionen für die Objektnamen-Autokorrektur** (siehe Bild 2).

Hier sehen wir drei Optionen:

- **Informationen zu Objektnamen-Autokorrektur nachverfolgen**: Diese Option startet lediglich die Nachverfolgung der Änderungen, die in einer zugleich erstellten Tabelle namens **MSysNameMap** gespeichert werden. Die Änderungen selbst werden für jedes Objekt der Datenbank gespeichert und liegen in einem Feld namens **NameMap**, das binäre Daten enthält (siehe Bild 3).
- **Objektnamen-Autokorrektur ausführen**: Erst wenn diese Option aktiviert ist, werden die vom Benutzer durchgeführten Änderungen tatsächlich auch auf andere Objekte übertragen.
- **Änderungen für Objektnamen-Autokorrektur protokollieren**: Diese Option sorgt dafür, dass die Änderungen auch in einem für User lesbaren Format in einer Tabelle namens **Objektnamen-Autokorrekturprotokoll** gespeichert werden. Mehr dazu weiter unten.

MSysNameMap

GUID	Id	Name	NameMap	Type
{EDEFC776-	1	tblBestellpositionen	Long binary-Daten	1
1AC18C189A3}	3	tblKunden	Long binary-Daten	1
C115EA30EC55}	4	tblProdukte	Long binary-Daten	1
B029241F99F4}	6	frmKunden_Table	Long binary-Daten	32768
CBA233088F11}	7	qryKunden	Long binary-Daten	5
F01E16A77481}	8	frmKunden_Query	Long binary-Daten	32768
DD4B32BC0E7}	9	frmKunden_SELECT	Long binary-Daten	32768
1183A79AE47}	10	frmHaupt	Long binary-Daten	32768
D744AC9015F}	11	tblBestellungen	Long binary-Daten	1
	(Neu)			

Bild 3: Die Tabelle **MSysNameMap**

Listen anzeigen mit dem ListView-Steuerelement

Die Datenblatt-Ansicht und das Listenfeld sind die bevorzugten Bordmittel, um Daten in Access in Listenform anzuzeigen. Beide haben Vor- und Nachteile. Doch es gibt auch noch das ListView-Steuerelement, das als ActiveX-Control bereitgestellt wird und mit dem sich viele unterschiedliche Ansichten realisieren lassen. In diesem Beitrag schauen wir uns an, wie wir Listen mit dem ListView-Steuerelement abbilden können – inklusive Funktionen wie Kontrollkästchen, Icons, Umbenennen und vieles mehr.

Listen werden in Access-Anwendungen immer wieder benötigt. Wenn die Daten bearbeitet werden sollen und Spalten angeordnet, vergrößert und verkleinert oder ein- und ausgeblendet werden sollen, ist die Datenblattansicht die beste Wahl.

Wenn Daten nur zur Auswahl angeboten werden sollen, bietet sich das Listenfeld an. Im Gegensatz zur Datenblattansicht offeriert dieses Steuerelement auch noch verschiedener Mehrfachauswahlmodi. Beide haben den großen Vorteil, dass man Datenquellen wie Tabellen oder Abfragen direkt binden kann und dazu kein zusätzlicher Code erforderlich ist.

Eine weitere Möglichkeit zur Darstellung von Listen ist das in diesem Beitrag vorgestellte **ListView**-Steuerelement. Der Nachteil gleich zu Beginn: Um es mit Daten zu füllen, benötigen wir zwangsläufig VBA-Code.

Es kann nicht einfach an eine Tabelle oder Abfrage gebunden werden, sondern wir müssen die einzelnen Einträge programmgesteuert hinzufügen.

Vorteile sind unter anderem die verschiedenen Ansichten, mit denen wir sowohl Daten in tabellarischer Form oder auch Bilder in Matrix-Form darstellen können. Außerdem können wir ein Kontrollkästchen zum einfachen Markieren von Einträgen anzeigen lassen. Und wir können einfachen Listeneinträgen sogar Icons hinzufügen. Nachfolgend schauen wir uns die verschiedenen Techniken zum Anzeigen von Daten in ListView-Steuerelementen an und gehen auch darauf ein, wie wir mit den enthaltenen Einträgen arbeiten können – beispielsweise, um per Doppelklick ein Detailformular zur Bearbeitung anzuzeigen.

Das Bearbeiten ist nämlich beim **ListView**-Steuerelement nur für die erste Spalte möglich.

Anlegen eines ListView-Steuerelements

Um einem Formular ein ListView-Steuerelement hinzuzufügen, öffnen wir das Formular in der Entwurfsansicht und wählen im Ribbon den Befehl **Formularentwurf|Steuerelemente|ActiveX-Steuerelemente** aus (siehe Bild 1).

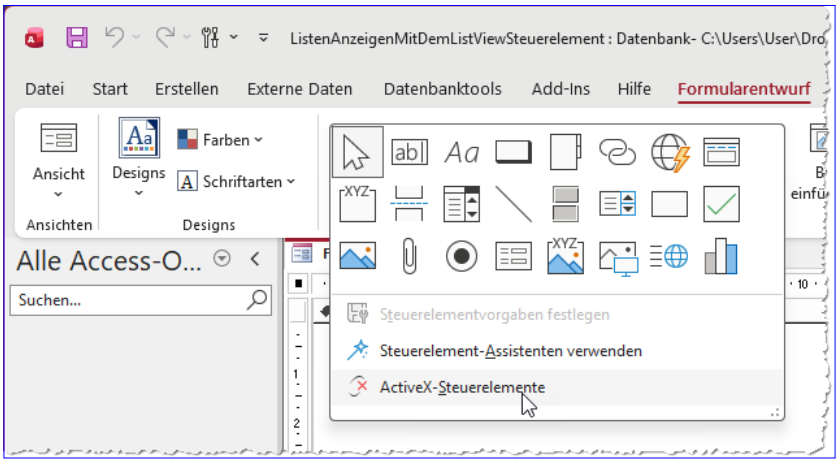


Bild 1: Auswahl der **ActiveX**-Steuerelemente

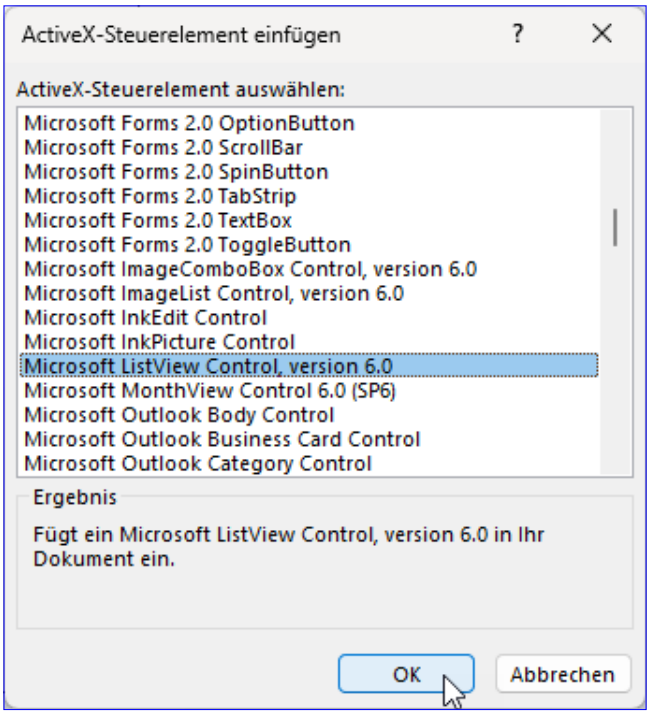


Bild 2: Auswahl des **ListView**-Steuerelements

Dies blendet den Dialog **ActiveX-Steuerelement einfügen** ein, in dem wir den Eintrag **Microsoft ListView Control, version 6.0** finden (siehe Bild 2). Nach der Auswahl klicken wir auf **OK**, um das Steuerelement zum Formularentwurf hinzuzufügen.

Das neue **ListView**-Steuerelement erscheint erst einmal recht unauffällig (siehe Bild 3), weshalb wir erst einmal die Größe anpassen.

Danach vergeben wir einen Namen. Wenn nur ein **ListView**-Steuerelement vorhanden ist, nennen wir es üblicherweise **ctlListView**. Natürlich kann man auch den üblichen Benennungskonventionen folgen und einen Namen wie **lvwKunden** oder **lvwProdukte** wählen.

Zu Beispielzwecken belassen wir es hier bei **ctlListView**.

Grundeinstellungen vornehmen

Das **ListView**-Steuerelement bietet verschiedene Einstellungen für das Erscheinungsbild an. Diese können wir in

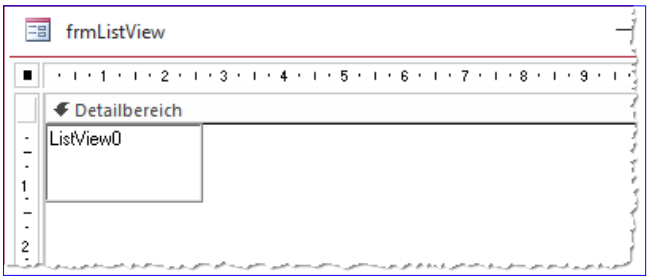


Bild 3: **ListView**-Steuerelement im Formularentwurf

den Eigenschaften des **ListView**-Steuerelements anpassen oder wir legen diese per VBA fest.

Im Eigenschaftenblatt finden wir die **ListView**-spezifischen Einstellungen auf der Registerseite **Andere** (siehe Bild 4). Diese schauen wir uns in den folgenden Abschnitten im Detail an.

Eigenschaften des ListView-Steuerelements

Die nachfolgend vorgestellten Eigenschaften können wir über das Eigenschaftenblatt oder per VBA einstellen. Geschickterweise werden die Eigenschaften des **ListView**-

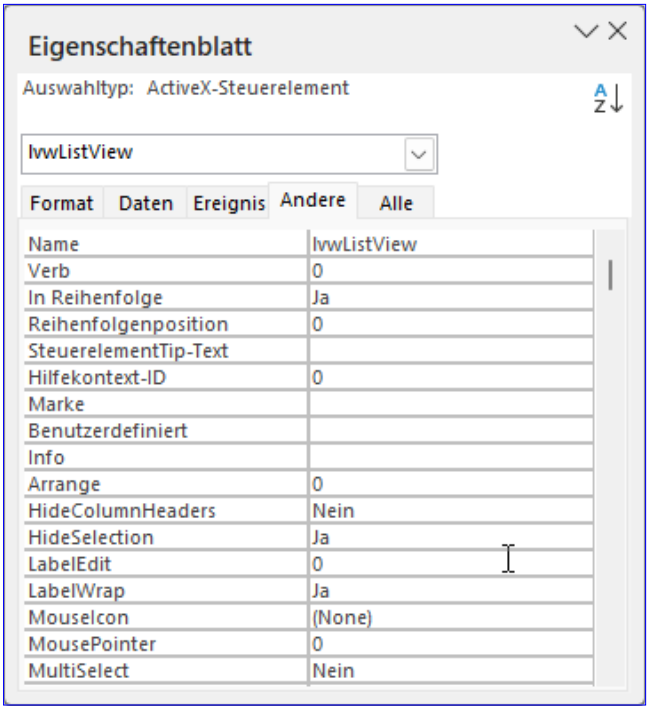


Bild 4: Eigenschaften des **ListView**-Steuerelements

Steuerelements hier in englischer Sprache abgebildet, sodass wir nicht lange nach den VBA-Pendants suchen müssen.

Eigener Eigenschaftendialog

Noch praktischer ist allerdings, dass das **ListView**-Steuerelement einen eigenen Eigenschaften-Dialog anbietet, den wir mit dem Kontextmenü-Eintrag **ListViewCtrl-Objekt|Properties** öffnen (siehe Bild 5).

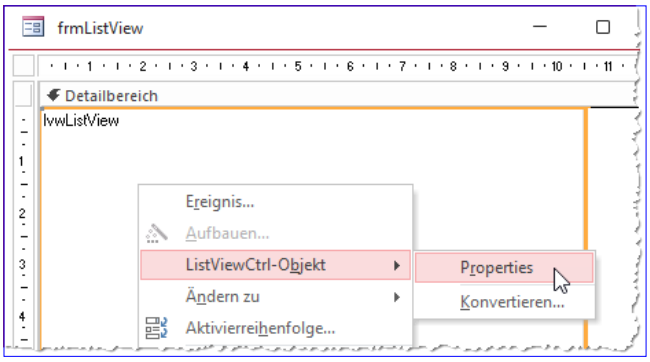


Bild 5: Anzeigen der Eigenschaften im eigenen Dialog

Dieser zeigt die Eigenschaften wie in Bild 6 an. Im Gegensatz zum herkömmlichen Eigenschaftenblatt sehen wir hier auch gleich die Bezeichnungen der Eigenschaftswerte und nicht nur die entsprechenden Zahlenwerte.

Im Reiter **General** sehen wir zunächst einige Auswahlfelder für die folgenden Eigenschaften:

- **AllowColumnReorder**: Gibt an, ob der Benutzer die Anordnung in der Ansicht **lvwReport** selbstständig anpassen kann (**True/False**).
- **Arrange**: Erwartet einen der Werte **lvwNone**, **lvwAutoLeft** oder **lvwAutoTop**. Die Werte geben an, in welcher Orientierung Icons in den Ansichten **lvwIcon** und **lvwSmallIcon** hinzugefügt werden. **lvwAutoLeft** füllt erst die erste Spalte und dann die folgenden Spalten, **lvwAutoTop** füllt erst die erste Zeile und dann die folgenden Zeilen.
- **Appearance**: Legt fest, ob das Steuerelement dreidimensional angezeigt wird (**cc3d**) oder flach (**ccFlat**).
- **BorderStyle**: Legt den Stil des Rahmens fest. Kann die Werte **ccFixedSingle** oder **ccNone** annehmen.
- **Checkboxes**: Gibt an, ob Kontrollkästchen zur Auswahl von Einträgen angezeigt werden sollen (**True/False**).
- **Enabled**: Gibt an, ob das Steuerelement aktiviert oder deaktiviert ist (**True/False**).
- **FlatScrollbar**: Legt fest, ob die Bildlaufleisten in der 2D- oder 3D-Ansicht angezeigt werden sollen (**True/False**).
- **FullRowSelect**: Gibt an, ob markierte Zeilen vollständig markiert werden sollen oder nur die erste Spalte (**True/False**).
- **Gridlines**: Gibt an, ob in der Ansicht **lvwReport** Gitternetzlinien zwischen Zeilen und Spalten angezeigt werden sollen (**True/False**).

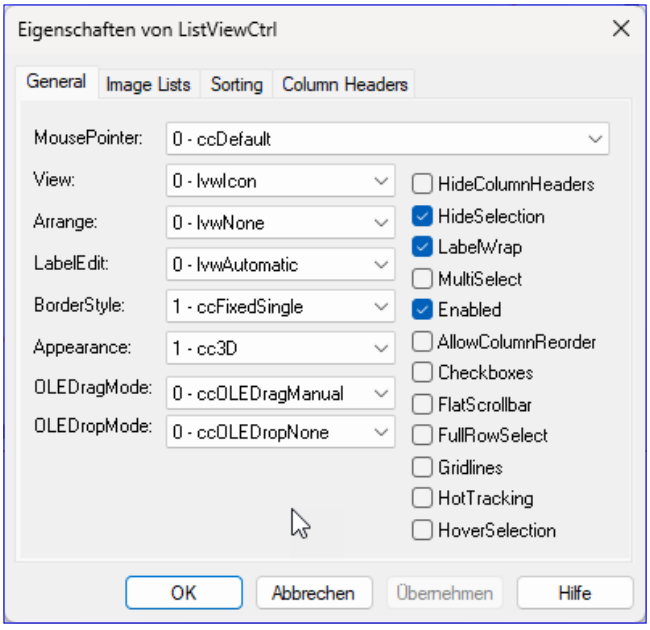


Bild 6: Eigener Eigenschaftendialog

- **HideColumnHeaders**: Gibt an, ob Spaltenüberschriften ein- oder ausgeblendet werden sollen (**True**/**False**).

- **HideSelection**: Gibt an, ob die Auswahl markiert bleibt, wenn das Steuerelement den Fokus verliert (**True**/**False**).

- **HotTracking**: Legt fest, ob der Eintrag, der gerade mit der Maus überfahren wird, optisch hervorgehoben werden soll (**True**/**False**).

- **HoverSelection**: Legt fest, ob Einträge, über denen die Maus kurze Zeit verweilt, automatisch markiert werden sollen (**True**/**False**).

- **LabelEdit**: Gibt an, ob die Beschriftung der ersten Spalte in der Ansicht **lvwReport** beziehungsweise die einzige Beschriftung der übrigen Ansichten geändert werden darf. **lvwAutomatic** erlaubt das Bearbeiten nach dem Anklicken des aktiven Elements, **lvwManual** erlaubt nur das codegesteuerte Ändern.

- **LabelWrap**: Gibt an, ob längere Texte in der Ansicht **lvwIcon** umbrochen werden sollen (**True**/**False**).

- **MousePointer**: Stellt verschiedene Mauszeiger ein.

- **MultiSelect**: Gibt an, ob bei gedrückter **Umschalt**- oder **Strg**-Taste mehrere Einträge gleichzeitig markiert werden können (**True**/**False**).

- **OLEDragMode**: Gibt an, wie das Steuerelement Drag-and-Drop-Operationen initialisiert, wenn der Benutzer ein Objekt zieht. Bei **ccOLEDragManual** wird Drag and Drop manuell per Code gestartet (zum Beispiel durch Aufruf von **OLEStartDrag**). Bei **ccOLEDragAutomatic** wird Drag and Drop automatisch gestartet, sobald der Benutzer ein Element zieht.

- **OLEDropMode**: Legt fest, wie das Steuerelement auf ein beim Drag and Drop fallengelassenes OLE-Objekt reagiert. Der Wert **ccOLEDropNone** erlaubt keine OLE-Drop-Operation. Bei **ccOLEDropManual** werden Drag and Drop-Ereignisse manuell im Code behandelt (in den Ereignissen **OLEDragOver** und **OLEDragDrop**). Bei **ccOLEDropAutomatic** wird der Inhalt automatisch an das Steuerelement übergeben.

- **View**: Stellt die Ansicht ein. Die verfügbaren Werte sind **lvwIcon**, **lvwSmallIcon**, **lvwList** und **lvwReport**. Wenn wir Daten mit mehreren Spalten in Listenform anzeigen wollen, nutzen wir hier den Eintrag **lvwReport**. Für einspaltige ListViews stellen wir **lvwList** ein. Wenn wir Icons oder Bilder anzeigen wollen, nutzen wir **lvwIcon** oder **lvwSmallIcons**.

Eigenschaften per Eigenschaftenblatt oder VBA festlegen?

Die Mehrzahl der Entwickler stellt die Eigenschaften von Formularen und Steuerelementen über das Eigenschaftenblatt ein. Beim **ListView**-Steuerelement ist das jedoch etwas unkomfortabel, da man hier immer wieder den Eigenschaften-Dialog über das Kontextmenü öffnen muss, statt direkt darauf zuzugreifen wie bei den eingebauten Steuerelementen.

Zudem haben wir festgestellt, dass wir immer wieder ein festes Set von Eigenschaftswerten für den Einsatz von **ListView**-Steuerelementen nutzen.

Außerdem muss man ohnehin eine Prozedur schreiben, die beispielsweise beim Laden des Formulars das **ListView**-Steuerelement mit Daten füllt. Dann kann man auch gleich die Eigenschaften einstellen. Deshalb erledigen wir dies in diesem Beitrag ebenfalls per VBA.

Me.ctlListView oder objListView?

Bei eingebauten Access-Steuerelementen greifen wir in VBA am einfachsten über den Verweis auf das Klassenmodul und den Steuerelementnamen auf das jeweilige Steuerelement zu, zum Beispiel mit **Me.txtBeispiel**. Das gelingt auch mit dem **ListView**-Steuerelement. Allerdings

erhalten wir dann nur die Standardeigenschaften von Steuerelementen und können nicht per IntelliSense auf die spezifischen Eigenschaften des **ListView**-Steuerelements zugreifen. Um das zu ermöglichen, benötigen wir eine Objektvariable, die wir beispielsweise wie folgt deklarieren:

```
Dim objListView As MSComctlLib.ListView
```

Danach weisen wir dieser nicht etwa **Me.ctlListView** als Wert zu, sondern **Me.ctlListView.Object**:

```
Set objListView = Me.ctlListView.Object
```

Damit können wir nun Intellisense für die Variable **objListView** nutzen (siehe Bild 7).

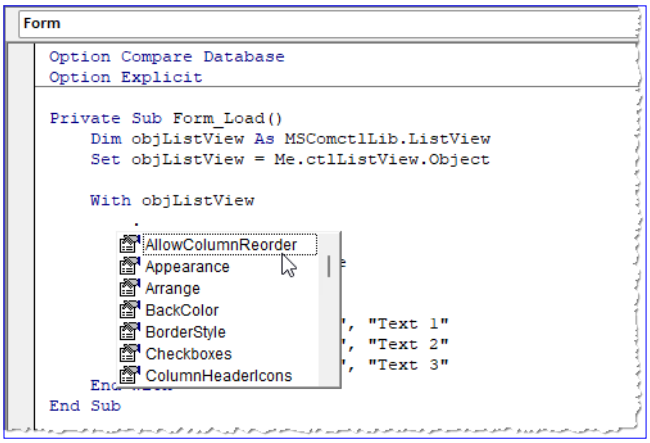


Bild 7: Eigenschaften des **ListView**-Steuerelements per IntelliSense

Einspaltiges ListView-Steuerelement mit Daten füllen

Es ist eine einfache Aufgabe, ein **ListView**-Steuerelement mit einer einzigen Zeile mit Daten zu füllen. Dazu nutzen wir das Listenfeld aus dem Formular **frmListView_Einspaltig**.

Mit der folgenden Prozedur nehmen wir zunächst einige Standardeinstellungen vor, durch die das Steuerelement optisch ansprechend dargestellt wird (**BorderStyle** und **Appearance**). Außerdem legen wir die **View** auf **lvwList** fest für die einspaltige Listenansicht.

Danach füllen wir das **ListView**-Steuerelement mit drei Einträgen. Dazu nutzen wir die Methode **Add** der **ListItems**-Auflistung:

```
Private Sub Form_Load()
    Dim objListView As MSComctlLib.ListView
    Set objListView = Me.ctlListView.Object

    With objListView
        .View = lvwList
        .BorderStyle = ccNone
        .Appearance = ccFlat
        .ListItems.Add , "a1", "Text 1"
        .ListItems.Add , "a2", "Text 2"
        .ListItems.Add , "a3", "Text 3"
    End With
End Sub
```

Die **Add**-Methode erwartet die folgenden Parameter:

- **Index**: Gibt die Position an, an der das Element eingefügt werden soll. Wenn die Elemente in der gewünschten Reihenfolge hinzugefügt werden, kann **Index** leer bleiben. Ansonsten führt beispielsweise die Verwendung des Wertes **1** dieses **1**-basierten Indexes dazu, dass das neue Element an erster Stelle vor eventuell bereits enthaltenen Elementen eingefügt wird.

- **Key**: Muss einen eindeutigen Identifizierer enthalten. Dieser muss mit einem Buchstaben beginnen. Wenn man Daten aus Tabellen einfügt, bietet sich die Kombination aus einem Buchstaben und dem Primärschlüsselwert des jeweiligen Eintrags an.

- **Text**: Erwartet den für dieses Element anzuzeigenden Text.

- **Icon**: Hier können wir, wenn wir ein **ImageList**-Steuerelement mit Icons zum Formular hinzugefügt haben, den Key oder den Index des für diesen Eintrag anzuzei-

genden Icons angeben. Diesen Parameter können wir in der Ansicht **lvwIcon** nutzen. Auf Icons gehen wir weiter unten ein.

- **SmallIcon**: Damit können wir ebenfalls ein Icon zu einem Eintrag hinzufügen. Dieser Parameter ist für die Ansichten **lvwSmallIcon**, **lvwList** und **lvwDetail** zu verwenden.

Geben wir also nun nur Werte für die Parameter **Key** und **Text** an, erhalten wir die Ansicht aus Bild 8. Damit sich das **ListView**-Steuerelement vom Hintergrund abhebt, haben wir die Eigenschaft **Rahmenart** auf **Durchgezogen** eingestellt.

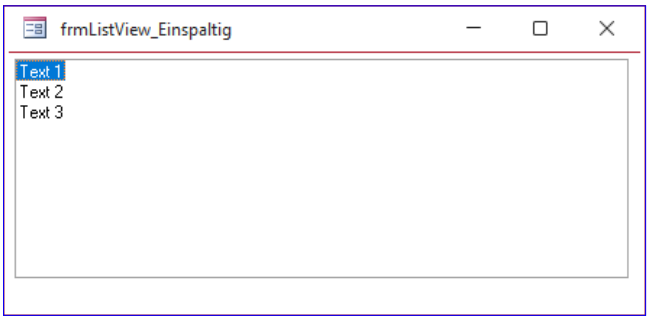


Bild 8: **ListView**-Steuerelement mit einspaltiger Ansicht

Mehrspaltiges ListView-Steuerelement mit Daten füllen

Mehrspaltige **ListView**-Steuerelemente sind ein wenig aufwendiger zu befüllen. Wir schauen uns dies im Formular **frmListView_Mehrspaltig** an.

Hier haben wir ebenfalls ein **ListView**-Steuerelement namens **ctlListView** hinzugefügt.

Wir führen ebenfalls beim Laden des Formulars eine Ereignisprozedur aus (siehe Listing 1). Diese deklariert neben dem **ListView**-Objekt noch ein Objekt mit dem Typ **ListItem**.

Das ist nötig, weil wir zum Anzeigen weiterer Spalten neben der ersten Spalte nicht einfach die **Add**-Methode des **ListView**-Objekts nutzen können. Stattdessen nutzen wir das Objekt, das wir mit dieser **Add**-Methode erstellen und das wir mit der Variablen **objListItem** referenzieren. Dieses bietet seinerseits eine Auflistung namens **ListSubItems** mit einer **Add**-Methode an, die allerdings etwas andere Parameter bietet. Die beiden Parameter **Icon** und **SmallIcon** fallen weg, dafür gibt es den Parameter

```
Private Sub Form_Load()
    Dim objListView As MSComctlLib.ListView
    Dim objListItem As MSComctlLib.ListItem
    Set objListView = Me.ctlListView.Object

    With objListView
        .View = lvwReport
        .LabelEdit = lvwManual
        .BorderStyle = ccNone
        .Appearance = ccFlat

        .ColumnHeaders.Add , "c1", "Spalte 1"
        .ColumnHeaders.Add , "c2", "Spalte 2"
        .ColumnHeaders.Add , "c3", "Spalte 3"

        Set objListItem = .ListItems.Add(, "a11", "Text 1 Spalte 1")
        objListItem.ListSubItems.Add , "a12", "Text 1 Spalte 2"
        objListItem.ListSubItems.Add , "a13", "Text 1 Spalte 3"
        Set objListItem = .ListItems.Add(, "a21", "Text 2 Spalte 1")
        objListItem.ListSubItems.Add , "ab22", "Text 2 Spalte 2"
        objListItem.ListSubItems.Add , "ab23", "Text 2 Spalte 3"
        Set objListItem = .ListItems.Add(, "a31", "Text 3 Spalte 1")
        objListItem.ListSubItems.Add , "ab32", "Text 3 Spalte 2"
        objListItem.ListSubItems.Add , "ab33", "Text 3 Spalte 3"
    End With
End Sub
```

Listing 1: Füllen eines **ListView**-Steuerelements mit mehreren Spalten

ListView-Steuerelement mit VBA programmieren

Im Beitrag »Listen anzeigen mit dem ListView-Steuerelement« (www.access-im-unternehmen.de/1572) haben wir gezeigt, wie wir mit dem ListView-Steuerelement arbeiten können. Dort haben wir bereits einige grundlegende VBA-Techniken gezeigt, mit denen wir das ListView-Steuerelement in den verschiedenen Ansichten mit Daten gefüllt haben. Im vorliegenden Beitrag geht es nun weiter: Wir zeigen, wie wir das Steuerelement per VBA programmieren können. Dabei liegt der Fokus auf den Methoden, mit denen wir auf Benutzereingaben reagieren oder verschiedene Informationen auslesen – zum Beispiel das aktuell markierte Element. Auch auf Mehrfachauswahl gehen wir ein, die wir sowohl setzen als auch auslesen werden. Und natürlich wird auch Drag and Drop eine Rolle in diesem Beitrag spielen.

Ereignisse des ListView-Steuerelements

Das **ListView**-Steuerelement bietet einige Ereignisse an, die wir uns in den folgenden Abschnitten genauer ansehen. Hier zunächst eine Übersicht mit den Funktionen der Ereignisse:

- **AfterLabelEdit**: Wird ausgelöst, nachdem der Benutzer die Bearbeitung einer Beschriftung beendet hat. Dient zur Prüfung oder Speicherung des geänderten Textes.
- **BeforeLabelEdit**: Tritt auf, bevor ein Benutzer die Beschriftung bearbeiten darf. Ermöglicht das Zulassen oder Verhindern des Editierens, wenn der Benutzer ein Element zum Bearbeiten anklicken will.
- **Click**: Wird ausgelöst, wenn der Benutzer auf das Steuerelement klickt (einfacher Klick, unabhängig vom ausgewählten Element).
- **ColumnClick**: Wird ausgelöst, wenn der Benutzer auf eine Spaltenüberschrift klickt. Wird häufig genutzt, um die Sortierung nach Spalten zu steuern.
- **DblClick**: Wird ausgelöst, wenn der Benutzer auf ein Element doppelklickt. Dient oft zum Öffnen oder Bearbeiten des gewählten Eintrags.
- **ItemCheck**: Tritt auf, wenn ein Kontrollkästchen neben einem **ListItem** aktiviert oder deaktiviert wird. Nur relevant, wenn **Checkboxes = True** gesetzt ist.
- **ItemClick**: Wird ausgelöst, wenn der Benutzer auf ein bestimmtes **ListItem**-Element klickt. Liefert direkten Zugriff auf das betroffene Element.
- **KeyDown**: Wird ausgelöst, wenn eine Taste gedrückt wird. Liefert Informationen über Tastencode und Status von **Umschalt**-, **Strg**- oder **Alt**-Tasten.
- **KeyPress**: Wird ausgelöst, wenn ein druckbares Zeichen eingegeben wird. Dient häufig zur Filterung von Eingaben.
- **KeyUp**: Wird ausgelöst, wenn eine Taste losgelassen wird. Kann genutzt werden, um Tastenkombinationen auszuwerten.
- **MouseDown**: Wird ausgelöst, wenn eine Maustaste gedrückt wird. Liefert Informationen über Maustaste, Position und Status der Umschalttasten.
- **MouseMove**: Wird ausgelöst, wenn der Mauszeiger über das Steuerelement bewegt wird. Nützlich für eigene Hover-Effekte.

- **MouseUp**: Wird ausgelöst, wenn eine zuvor gedrückte Maustaste losgelassen wird.
- **OLECompleteDrag**: Wird ausgelöst, nachdem eine Drag and Drop-Operation abgeschlossen wurde. Dient zum Bereinigen oder Aktualisieren des Zustands.
- **OLEDragDrop**: Wird ausgelöst, wenn ein Objekt über dem Steuerelement abgelegt wird. Hier wird üblicherweise der Drop-Inhalt verarbeitet.
- **OLEDragOver**: Wird fortlaufend ausgelöst, während ein Objekt über dem Steuerelement bewegt wird. Eignet sich zur Anzeige von Drop-Zielen.
- **OLEGiveFeedback**: Wird während einer Drag and Drop-Operation ausgelöst. Ermöglicht die Anpassung des Mauszeigers oder Feedback-Verhaltens.
- **OLESetData**: Wird ausgelöst, wenn das Steuerelement Daten bereitstellen soll, die per Drag and Drop übergeben werden.
- **OLEStartDrag**: Wird ausgelöst, wenn der Benutzer eine Drag and Drop-Operation startet. Hier werden üblicherweise die zu übertragenden Daten festgelegt.
- **Updated**: wird in VBA nicht ausgelöst

Spaltenköpfe und Einträge vor dem erneuten Füllen leeren

Manchmal werden die angelegten Einträge für die Spaltenköpfe und die Zeilen des **ListView**-Steuerelements beim Schließen gespeichert, sodass ein erneutes Öffnen und der Versuch, die Elemente erneut anzulegen, zu einem Fehler führt.

Daher sollte man diese Elemente vorsichtshalber immer löschen, bevor man das **ListView**-Steuerelement erneut füllt:

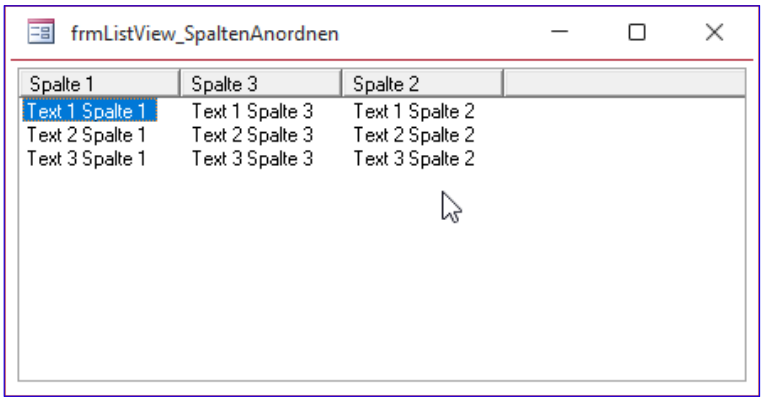


Bild 1: Umsortierte Spalten im **ListView**-Steuerelement

```
objListView.ColumnHeaders.Clear
objListView.ListItems.Clear
```

Spalten anordnen

Damit der Benutzer die Spalten eines **ListView**-Steuerelements anordnen kann, müssen wir die Eigenschaft **AllowColumnReorder** auf **True** einstellen:

```
objListView.AllowColumnReorder = True
```

Danach kann der Benutzer die Spalten wie in Bild 1 umsortieren.

Leider gibt es kein Ereignis, das durch das Umsortieren ausgelöst wird, sodass wir die Anordnung nicht ohne größeren Aufwand speichern und beim nächsten Öffnen des Formulars wiederherstellen können (siehe Formular **frmListView_SpaltenAnordnen** der Beispieldatenbank).

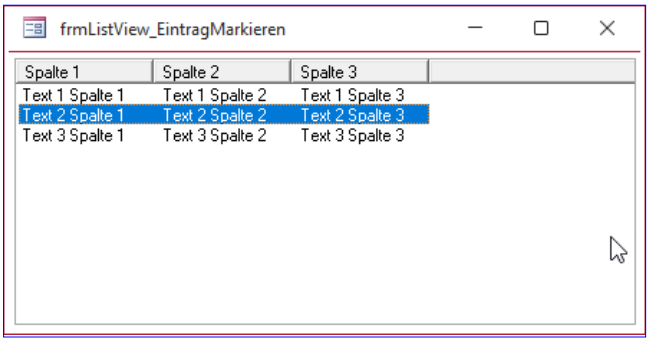


Bild 2: Vollständige Zeile markieren

Selektierte Spalten vollständig markieren

Wir sind es vom Listenfeld von Access gewohnt, dass markierte Einträge vollständig markiert werden. Im **ListView**-Steuerelement wird standardmäßig nur die erste Spalte des ausgewählten Eintrags markiert. Dies können wir ändern, indem wir die Eigenschaft **FullRowSelect** auf **True** einstellen:

```
objListView.FullRowSelect = True
```

Markieren wir nun eine Zeile, werden alle Spalten markiert (siehe Bild 2) – siehe **frmListView_EintragMarkieren**.

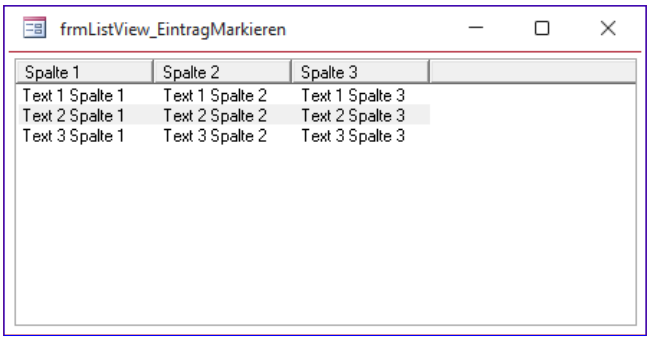


Bild 3: Markierung im **ListView**-Steuerelement bei Fokusverlust beibehalten

Markierung bei Fokusverlust beibehalten

Standardmäßig ist die Markierung des aktuellen Eintrags nicht mehr sichtbar, wenn das **ListView**-Steuerelement den Fokus verliert.

Dem können wir zumindest teilweise entgegenwirken, indem wir die Eigenschaft **HideSelection** auf **False** einstellen:

```
objListView.HideSelection = False
```

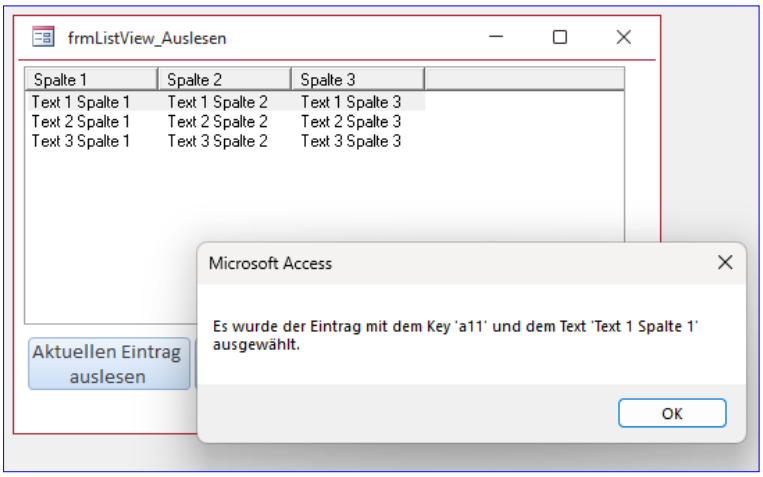


Bild 4: Eintrag im **ListView**-Steuerelement auslesen

Die aktuelle Zeile wird nun immerhin noch grau hinterlegt (siehe Bild 3).

Objektvariable modulweit deklarieren

In den folgenden Beispielen wollen wir aus anderen Prozeduren heraus auf das **ListView**-Steuerelement zugreifen. Dazu könnten wir dieses immer wieder neu deklarieren und referenzieren, aber diesen Aufwand wollen wir uns sparen. Also deklarieren wir **objListView** wie folgt im Kopf der Prozedur:

```
Dim objListView As MSComctlLib.ListView
```

Dann reicht es, wenn wir diese Variable einmal in der Ereignisprozedur **Beim Laden** füllen:

```
Set objListView = Me.ctlListView.Object
```

In den weiteren Prozeduren können wir anschließend einfach auf **objListView** zugreifen.

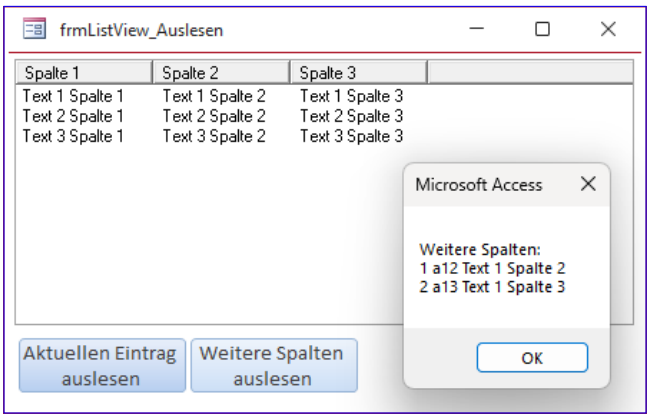


Bild 5: Weitere Spalten des markierten Eintrags auslesen

Markierten Eintrag auslesen

Im Formular **frmListView_Auslesen** zeigen wir, wie der aktuell markierte Listeneintrag ausgelesen werden kann. Dabei greifen wir auf den Key und den Text der ersten Spalte zu und geben diese Informationen in einem Meldungsfenster aus (siehe Bild 4).

Für die Schaltfläche **cmdAktuellenEintragAuslesen** hinterlegen wir den folgenden Code:

```
Private Sub cmdAktuellenEintragAuslesen_Click()
    Dim objListItem As MSComctlLib.ListItem
    Set objListItem = objListView.SelectedItem
    If Not objListItem Is Nothing Then
        MsgBox "Es wurde der Eintrag mit dem Key '" _
            & objListItem.Key & "' und dem Text '" _
            & objListItem.Text & "' ausgewählt."
    End If
End Sub
```

Hier lesen wir mit der Eigenschaft **SelectedItem** den aktuellen Eintrag aus. Ist das Ergebnis nicht **Nothing**, können wir über die Eigenschaften **Key** und **Text** von **objListItem** auf die entsprechenden Werte zugreifen.

Werte weiterer Spalten auslesen

Gegebenenfalls sind auch noch die übrigen Spalten des aktuell markierten Eintrags interessant. Diese wollen wir in einer Meldung wie in Bild 5 ausgeben.

Dazu fügen wir für die zweite Schaltfläche diese Prozedur hinzu:

```
Private Sub cmdWeitereSpaltenAuslesen_Click()
    ...
    Dim objListSubItem As MSComctlLib.ListSubItem
    Dim strSubItems As String
    Set objListItem = objListView.SelectedItem
    If Not objListItem Is Nothing Then
        For Each objListSubItem In objListItem.ListSubItems
            With objListSubItem
                strSubItems = strSubItems & vbCrLf _
                    & .Index & " " & .Key & " " & .Text
            End With
        Next objListSubItem
    End If
    MsgBox "Weitere Spalten: " & strSubItems
End Sub
```

Die Prozedur deklariert zusätzlich eine Variable des Typs **ListSubItem**. Nachdem wir mit **objListItem** den aktuellen Eintrag ermittelt haben, durchlaufen wir in einer **For Each**-Schleife alle Elemente der Auflistung **ListSubItems**. Für diese stellen wir jeweils eine Zeile mit dem **Index**, dem **Key** und dem **Text** in der Variablen **strSubItems** zusammen.

Einen Eintrag markieren

Wir wollen nicht nur den aktuellen Eintrag auslesen, sondern diesen auch einstellen können. Dazu haben wir das Formular **frmListView_EintragSelektieren** angelegt. Hier finden wir drei Schaltflächen. Die erste Schaltfläche löst die folgende Prozedur aus und selektiert damit den zweiten Eintrag durch Angabe des **Index**-Wertes dieses Eintrags beim Zuweisen des Wertes **True** an die Eigenschaft **Selected**:

```
Private Sub cmdZweitenEintragPerIndex_Click()
    objListView.ListItems(2).Selected = True
    Me.ctlListView.SetFocus
End Sub
```

Wir können auch einen Eintrag über den **Key**-Wert selektieren. Das erledigen wir mit der zweiten Schaltfläche:

```
Private Sub cmdDrittenEintragPerKey_Click()
    objListView.ListItems("a31").Selected = True
    Me.ctlListView.SetFocus
End Sub
```

Schließlich wollen wir auch noch die Selektion leeren können.

Die folgende Prozedur ermittelt den markierten Eintrag und wählt diesen ab:

```
Private Sub cmdAlleEintraegeAbwaehlen_Click()
    Dim objListItem As MSComctlLib.ListItem
    Set objListItem = objListView.SelectedItem
    objListItem.Selected = False
End Sub
```

Mehrfachauswahl im ListView-Steuerelement

Im Formular **frmListView_Mehrfachauswahl_Auslesen** setzen wir uns mit der Mehrfachauswahl im **ListView**-Steuerelement auseinander (siehe Bild 6). Dazu stellen wir zunächst die Eigenschaft auf **True** ein:

```
objListView.MultiSelect = True
```

Damit können wir nun bei gedrückter **Umschalt**- oder **Strg**-Taste wie im Windows Explorer mehrere Einträge gleichzeitig markieren.

```
Private Sub cmdAktuelleEintraegeAuslesen_Click()
    Dim objListItem As MSComctlLib.ListItem
    Dim strSelected As String
    For Each objListItem In objListView.ListItems
        If objListItem.Selected Then
            strSelected = strSelected & vbCrLf _
                & objListItem.Key & " " & objListItem.Text
        End If
    Next objListItem
    MsgBox "Markierte Einträge:" & strSelected
End Sub
```

Die Prozedur durchläuft alle Einträge des Listenfeldes. Darin prüft sie mit der **Selected**-Eigenschaft, ob der jeweilige Eintrag markiert ist.

Wenn wir anschließend die Schaltfläche betätigen, erhalten wir eine Liste der markierten Einträge (siehe Bild 7).

Mehrere Einträge im ListView per VBA selektieren

Nun wollen wir mehrere Einträge in einer Mehrfachauswahl per VBA selektieren (siehe **frmListView_MehrereEintraegeSelektieren**). Im Beispiel wollen wir mit einer Schaltfläche den ersten und den dritten Eintrag selektieren.

Die Schaltfläche löst die folgende Prozedur aus:

```
Private Sub cmdMehrereEintraegeSelektieren_Click()
    Dim objListItem As MSComctlLib.ListItem
    Set objListItem = objListView.ListItems(1)
    objListItem.Selected = True
    Set objListItem = objListView.ListItems(3)
    objListItem.Selected = True
    Me.ctlListView.SetFocus
End Sub
```

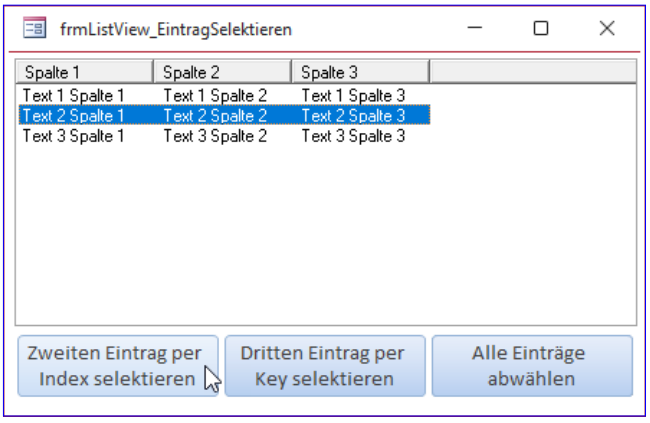


Bild 6: Markieren von Einträgen im **ListView**-Steuerelement

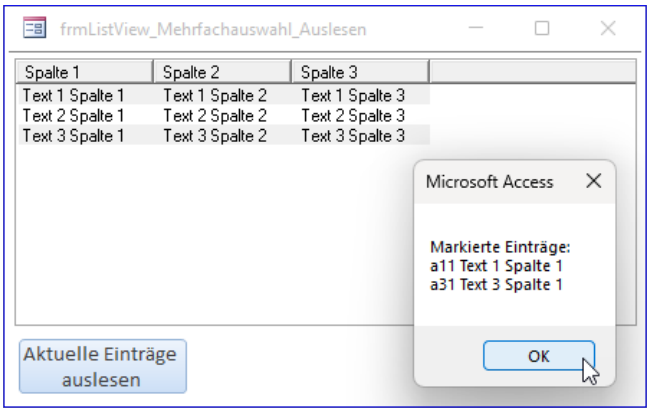


Bild 7: Ausgabe der Einträge einer Mehrfachauswahl

Hier selektieren wir zuerst den ersten Eintrag und markieren diesen und führen die gleichen Schritte dann für den dritten Eintrag aus. Das Ergebnis sehen wir in Bild 8.

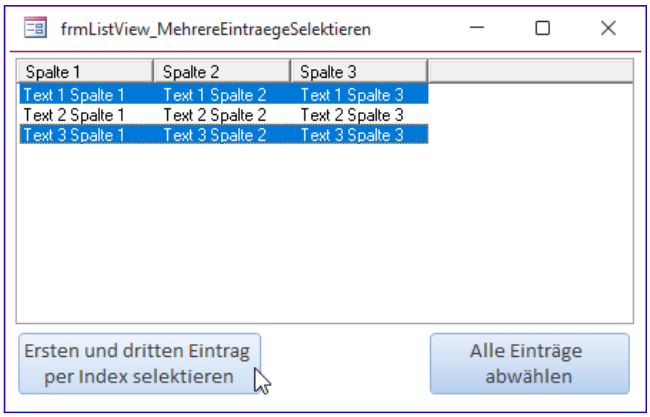


Bild 8: Mehrfachauswahl per VBA selektieren

Die zweite Schaltfläche wählt alle Einträge wieder ab.

Sie durchläuft alle Einträge und stellt den Wert der Eigenschaft **Selected** jeweils auf **False** ein:

```
Private Sub cmdAlleEintraegeAbwaehlen_Click()
    Dim objListItem As MSComctlLib.ListItem
    For Each objListItem In objListView.ListItems
        objListItem.Selected = False
    Next objListItem
End Sub
```

Sortieren der Einträge

Wir können die Einträge des Listenfeldes jeweils nach einer Spalte aufsteigend oder absteigend sortieren (siehe Formular **frmListView_Sortieren**). Dazu benötigen wir eine Prozedur, die durch das Ereignis **ColumnClick** des **ListView**-Steuerelements ausgelöst wird.

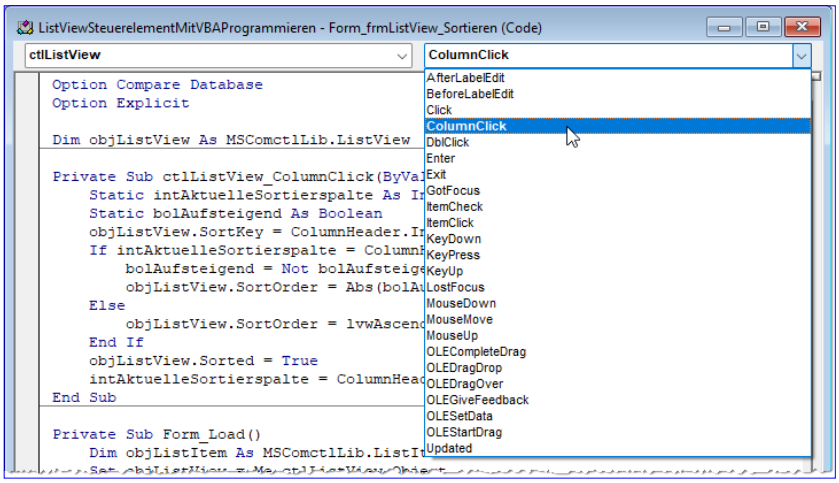


Bild 9: Anlegen einer Ereignisprozedur für das **ListView**-Steuerelement

Diese Ereignisprozedur legen wir an, indem wir im Codefenster für das Klassenmodul des Formulars aus der linken Liste den Eintrag für das **ListView**-Steuerelement auswählen und aus der rechten den Eintrag **ColumnClick** (siehe Bild 9).

Die so erstellte Ereignisprozedur **lvwListView_ColumnClick** füllen wir wie folgt:

```
Private Sub ctlListView_ColumnClick(ByVal ColumnHeader _
        As Object)
    Static intAktuelleSortierspalte As Integer
    Static bolAbsteigend As Boolean
    objListView.SortKey = ColumnHeader.Index - 1
    If intAktuelleSortierspalte = ColumnHeader.Index - 1 Then
        bolAbsteigend = Not bolAbsteigend
    Else
        bolAbsteigend = False
    End If
    objListView.SortOrder = Abs(bolAbsteigend)
    objListView.Sorted = True
    intAktuelleSortierspalte = ColumnHeader.Index - 1
End Sub
```

Sie enthält zwei als statisch deklarierte Variablen. Das heißt, dass diese Variablen auch nach dem Verlassen der Prozedur ihre Werte behalten, aber außerhalb der Prozedur dennoch nicht zugreifbar sind. Die Prozedur bekommt mit dem Parameter **Columnheader** einen Verweis auf den Spaltenkopf, den der Benutzer angeklickt hat.

Den aktuellen Sortierindex ermitteln wir aus **ColumnHeader.Index - 1** und weisen diesen der Eigenschaft **SortKey** des **ListView**-Steuerelements zu. Damit legen wir fest, welche Spalte sortiert werden soll.

Danach prüfen wir, ob **intAktuelleSortierspalte** den gleichen Wert wie **ColumnHeader.Index - 1** hat. In diesem Fall kehren wir den Wert der Variablen **bolAufsteigend**, der standardmäßig **0** ist, um und weisen den Wert ohne Vorzeichen der Eigenschaft **SortOrder** zu.

Der Wert **0** entspricht der Konstanten **lvwAscending** (aufsteigend), der Wert **1** der Konstanten **lvwDescending** (absteigend).

Falls **intAktuelleSortierspalte** nicht der markierten Spalte entspricht, soll die Sortierreihenfolge auf **lvwAscending** (aufsteigend) eingestellt werden. Das ist immer der Fall, wenn nicht die gleiche Spalte wie zuvor angeklickt wurde.

Durch das Einstellen der Eigenschaft **Sorted** auf den Wert **True** wird die Sortierung angewendet.

Schließlich stellt die Prozedur die Variable **intAktuelleSortierspalte** auf den aktuellen Index minus 1 ein. Damit stellen wir sicher, dass diese Spalte nun andersherum sortiert wird, wenn der Benutzer erneut auf den gleichen Spaltenkopf klickt.

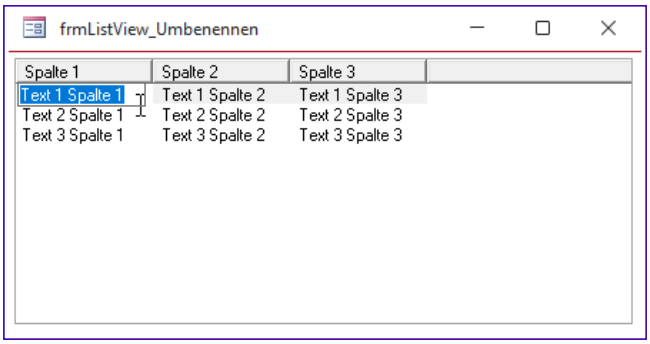


Bild 10: Umbenennen eines Eintrags

Wert der ersten Spalte ändern

Wir können zwar im **ListView**-Steuerelement nicht direkt Daten bearbeiten, wie es im Datenblatt der Fall ist. Aber wir können immerhin den Wert der ersten Spalte anpassen.

Dies müssen wir allerdings zunächst aktivieren, indem wir die Eigenschaft **LabelEdit** auf den Wert **lvwAutomatic** einstellen:

```
objListView.LabelEdit = lvwAutomatic
```

Standardmäßig steht dieser Wert auf **lvwManual**. Sobald wir diesen geändert haben, können wir den Wert der ersten Spalte anklicken und dieser wird wie im Windows Explorer zum Ändern markiert (siehe Bild 10).

Auf Änderungen reagieren

Wenn man im **ListView**-Steuerelement reale Daten anzeigt und den Benutzer diese ändern lässt, werden

```
Private Sub ctlListView_AfterLabelEdit(Cancel As Integer, NewString As String)
    Dim objListItem As MSComctlLib.ListItem
    Dim strKey As String
    Dim strOldString As String

    Set objListItem = objListView.SelectedItem
    strKey = objListItem.Key
    strOldString = objListItem.Text

    MsgBox "Der Eintrag mit dem Key '" & strKey & "' soll geändert werden von '" & strOldString & "' auf '" & NewString & "'."
End Sub
```

Listing 1: Reagieren auf die Änderung eines **ListView**-Eintrags

die Änderungen nicht automatisch in die Datenherkunft übernommen. Wir verwenden in diesem Beispiel zwar nur einige Beispieldaten, aber wir stellen dennoch die Vorgehensweise vor, um auf Änderungen zu reagieren und den neuen Wert zu ermitteln.

Wir haben gleich zwei Ereignisse, die durch das Ändern eines Eintrags ausgelöst werden. **BeforeLabelEdit** wird bereits beim Anklicken des jeweiligen Eintrags zum Ändern ausgelöst, aber bevor dieser zum Ändern freigegeben wird. Wir können hier beispielsweise prüfen, ob der aktuelle Eintrag geändert werden darf. Falls nicht, können wir in der Prozedur eine Meldung ausgeben und brechen den Änderungsvorgang durch Setzen des Parameters **Cancel** auf den Wert **True** ab.

Interessanter ist die zweite Ereignisprozedur **AfterLabelEdit**. Diese wird ausgelöst, wenn der Benutzer die Änderung eingetragen hat und etwa mit der Eingabetaste abschließt. Diese Prozedur liefert nicht nur den Parameter **Cancel**, mit dem wir den Änderungsvorgang abbrechen können, wenn der eingegebene Wert beispielsweise nicht validiert werden kann, sondern auch noch den Parameter **NewString**, der den neu eingetragenen Wert zurückliefert.

Wir fügen die Prozedur **ctlListView_AfterLabelEdit** über die beiden Auswahlfelder des Codefensters hinzu und füllen sie wie in Listing 1. Die Prozedur deklariert eine Variable namens **objListItem**, in die wir mit der Eigenschaft **SelectedItem** den aktuell markierten Eintrag beim Durchführen der Änderung eintragen. Aus diesem Eintrag lesen wir den Key und den Text aus und speichern beide in den Variablen **strKey** und **strOldString**.

Danach geben wir diese Informationen in einer Meldung wie in Bild 11 aus.

Auf Doppelklick reagieren

Der einfache Klick selektiert einen Eintrag, daher macht es wenig Sinn, hier noch weitere Funktionen zu hinterlegen.

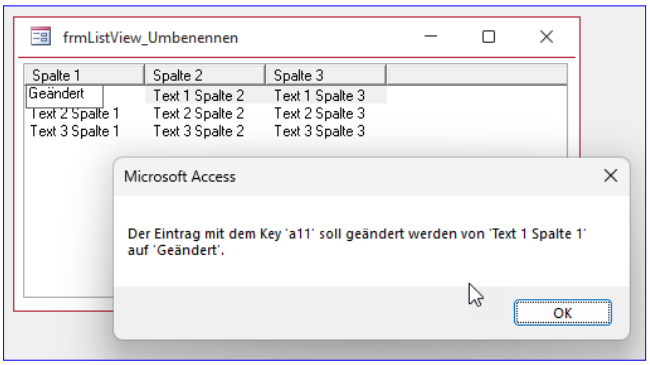


Bild 11: Ereignis vor dem Umbenennen

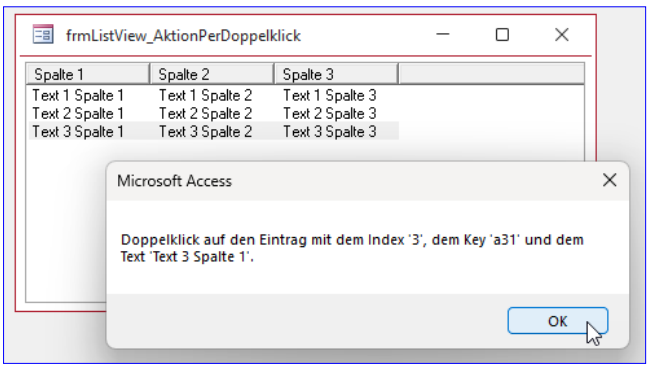


Bild 12: Infos zum Element per Doppelklick

Wenn wir Details zu einem Datensatz im **ListView**-Steuerelement anzeigen wollen, können wir das beispielsweise per Doppelklick machen (siehe **frmListView_AktionPerDoppelklick**).

Dazu nutzen wir das Ereignis **DblClick**, das wir über das Codefenster anlegen und wie folgt füllen:

```
Private Sub ctlListView_DblClick()
    Dim objListItem As MSComctlLib.ListItem
    Dim lngIndex As Long
    Dim strKey As String
    Dim strText As String
    Set objListItem = objListView.SelectedItem
    If Not objListItem Is Nothing Then
        With objListItem
            lngIndex = .Index
            strKey = .Key
            strText = .Text
        End With
```

m:n-Beziehung mit Drag and Drop per ListView

Wir haben uns bereits in einigen Beiträgen angesehen, wie man die Daten einer m:n-Beziehung mit zwei nebeneinander liegenden Listenfeldern verwalten kann. Das Hinzufügen oder Entfernen erfolgte dabei per Doppelklick auf den jeweiligen Eintrag oder über entsprechende Schaltflächen. Im Gegensatz zu Listenfeldern können wir im ListView-Steuerelement jedoch auch Drag and Drop einsetzen. Wie das grundlegend funktioniert, haben wir uns bereits im Beitrag »ListView-Steuerelement mit VBA programmieren« (www.access-im-unternehmen.de/1573) angesehen. Im vorliegenden Beitrag schauen wir uns an, wie wir die Daten aus zwei ListView-Steuerelementen per Drag and Drop hin- und herschieben können.

Datenmodell des Beispiels

Als Beispiel wollen wir einen Verteiler für Newsletter verwenden. Wir benötigen also eine Tabelle mit den Empfängern des Newsletters, eine Tabelle für die einzelnen Newsletter und eine weitere, um die Empfänger den verschiedenen Newslettern zuordnen zu können.

Die erste Tabelle namens **tblEmpfaenger** enthält die Felder **EmpfaengerID**, **Empfaenger** und **EMail** (siehe Bild 1). Hier speichern wir die Empfänger für die Newsletter.

Die Newsletter wiederum speichern wir in der Tabelle **tblNewsletter**, die wir in Bild 2 sehen. Hier haben wir nur die wichtigsten Felder eingefügt, man könnte noch weitere Felder wie Versanddatum et cetera hinzufügen, wenn man die Anwendung in der Praxis nutzen möchte.

Zusätzlich zu den dazu notwendigen Tabellen **tblNewsletter** und **tblEmpfaenger** brauchen wir eine Tabelle zum Verknüpfen der Datensätze der beiden Tabellen namens **tblVerteiler** über eine m:n-Beziehung. Diese enthält neben dem Primärschlüsselfeld **VerteilerID** zwei weitere Felder, über die wir die Datensätze der Tabellen **tblEmpfaenger** und **tblNewsletter** miteinander

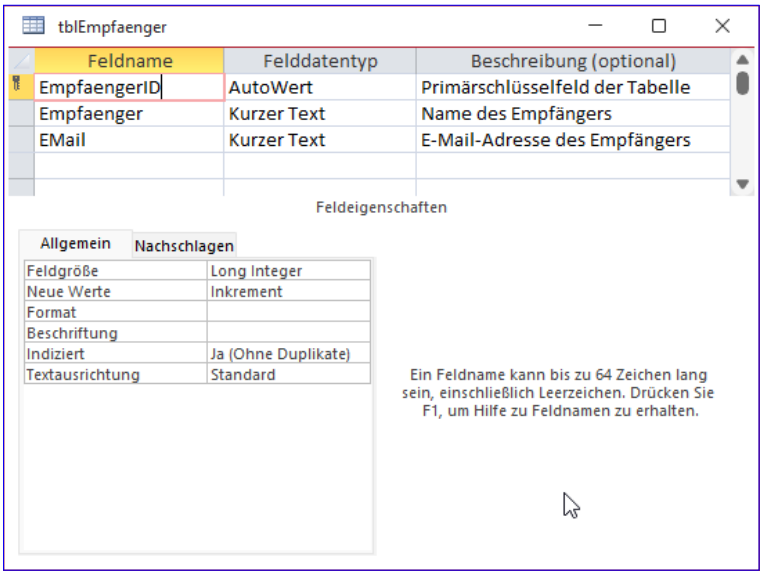


Bild 1: Die Tabelle **tblEmpfaenger**

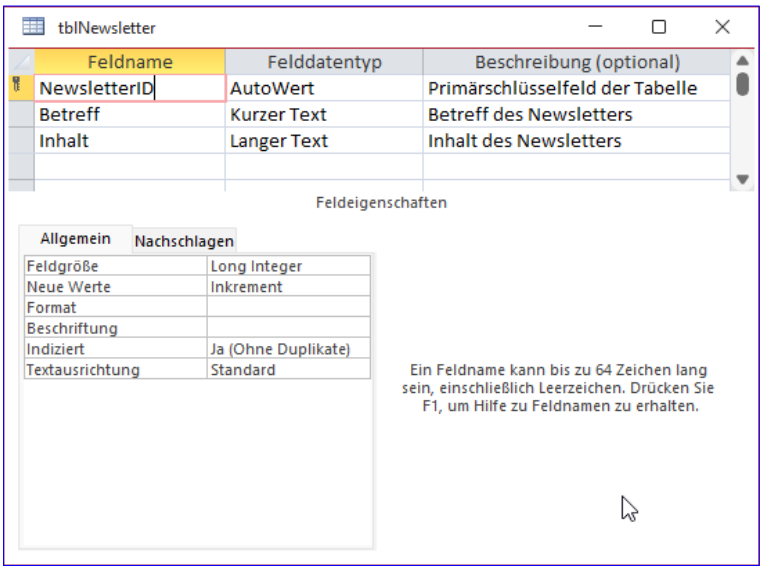


Bild 2: Die Tabelle **tblNewsletter**

verknüpfen. Dies realisieren wir über die beiden Felder **EmpfaengerID** und **NewsletterID**. In diesem Fall wollen wir keine Nachschlagefelder einrichten, da wir diese im zu erstellenden Formular nicht benötigen.

Damit jeder Empfänger nur einmal jedem Newsletter zugewiesen werden kann, haben wir außerdem einen eindeutigen Schlüssel über die beiden Felder **EmpfaengerID** und **NewsletterID** hinzugefügt (siehe Bild 3).

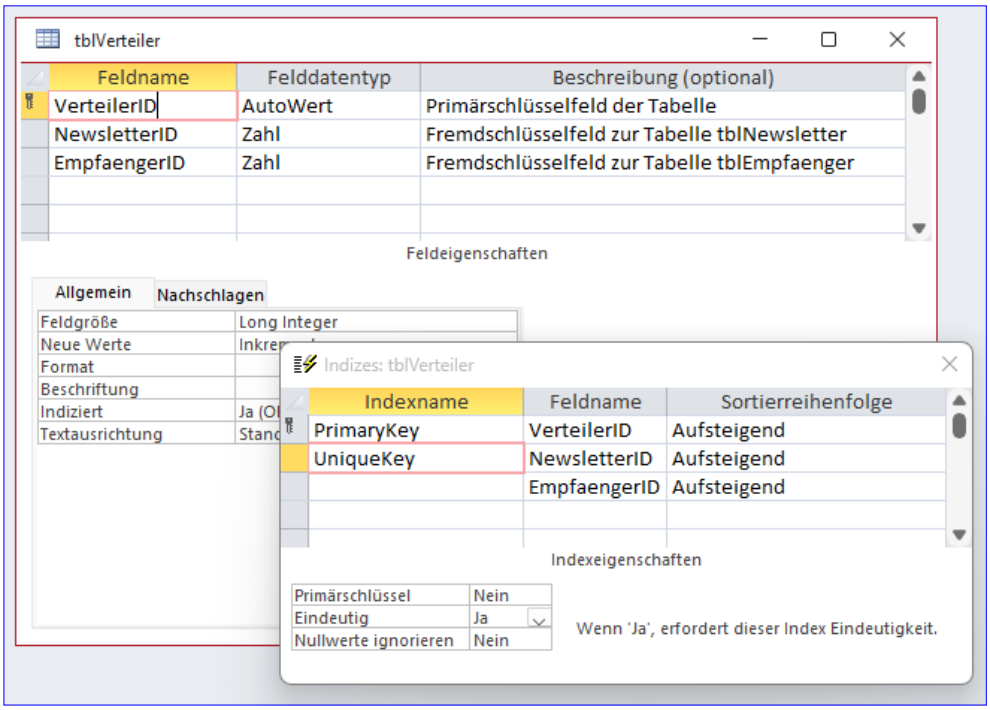


Bild 3: Die m:n-Verknüpfungstabelle **tblVerteiler**

Stattdessen legen wir die Beziehungen direkt im Beziehungen-Fenster von Access an. Hier ziehen wir alle drei Tabellen der Anwendung hinein und verknüpfen diese wie in Bild 4. Für beide Beziehungen definieren wir referenzielle Integrität mit Löschweitergabe. Damit erreichen wir, dass beim Löschen eines Empfängers oder eines Newsletters auch die damit verknüpften Einträge der Tabelle **tblVerteiler** gelöscht werden.

Formular zum Zuordnen von Empfängern zu den Newslettern

Das Formular soll in zwei **ListView**-Steuerelementen die Empfänger und die Nicht-Empfänger der Newsletter anzeigen. Als Datensatzquelle des Formulars dient die Tabelle **tblNewsletter**. Das Formular zeigt beide Felder der Tabelle untereinander im Detailbereich an (siehe Bild 5).

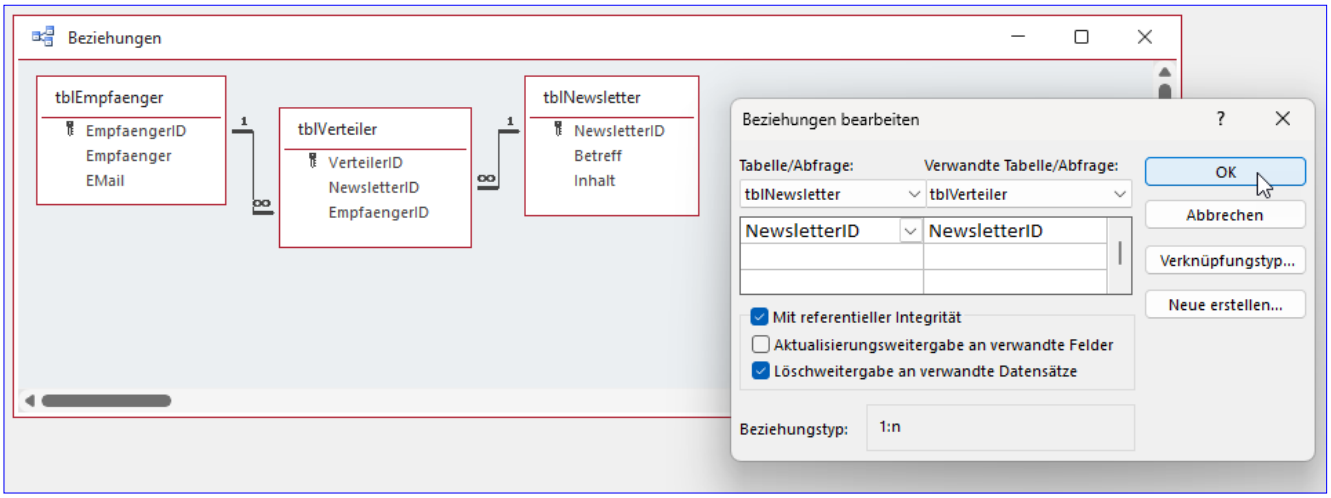


Bild 4: Beziehungen zum Herstellen der m:n-Beziehung

Außerdem fügen wir zwei **ListView**-Steuerelemente namens **lvwEmpfaenger** und **lvwKeinEmpfaenger** hinzu.

Dazu betätigen wir den Befehl **Formularentwurfl-SteuerelementelActiveX-Steuerelemente**. Dies öffnet den Dialog **ActiveX-Steuerelemente** einfügen, wo wir den Eintrag **Microsoft ListView Control** finden (siehe Bild 6).

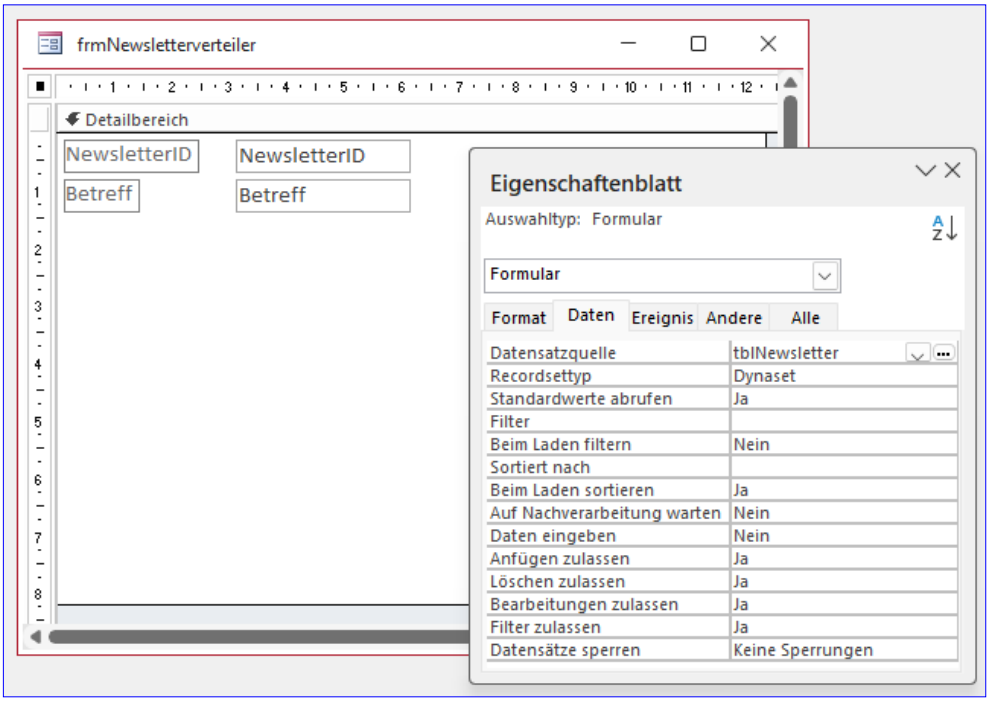


Bild 5: Hinzufügen der Felder der Tabelle **tblNewsletter**

Die beiden **ListView**-Steuerelemente fügen wir unterhalb der bereits vorhandenen Steuerelemente ein und passen ihre Bezeichnungen an (siehe Bild 7).

Einstellungen für die ListView-Steuerelemente

In der Ereignisprozedur, die durch das Ereignis **Beim Laden** ausgelöst wird, wollen wir die Einstellungen für die beiden **ListView**-Steuerelemente vornehmen.

Den Großteil der Arbeit wollen wir jedoch in eine weitere Prozedur auslagern, weil die Befehle für beide **ListView**-Steuerelemente fast identisch sind. Zunächst jedoch deklarieren wir Objektvariablen für die beiden **ListView**-Steuerelemente im Kopf des Klassenmoduls des Formulars:

```
Dim objEmpfaenger As MSComctlLib.ListView
Dim objKeinEmpfaenger As MSComctlLib.ListView
```

Die **Form_Load**-Ereignisprozedur füllt diese mit Verweisen auf die **Object**-Eigenschaften der entsprechenden Steuerelemente und ruft für jedes die Prozedur **ListView-Einstellen** auf.

Dabei übergibt sie den Verweis auf das jeweilige **ListView**-Steuerelement und den Text, der im Spaltenkopf erscheinen soll:

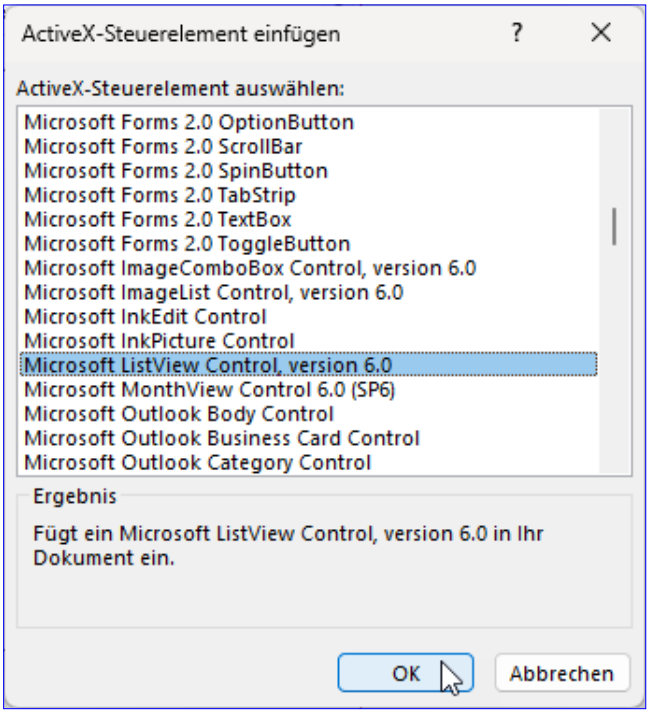


Bild 6: Auswählen des **ListView**-Steuerelements

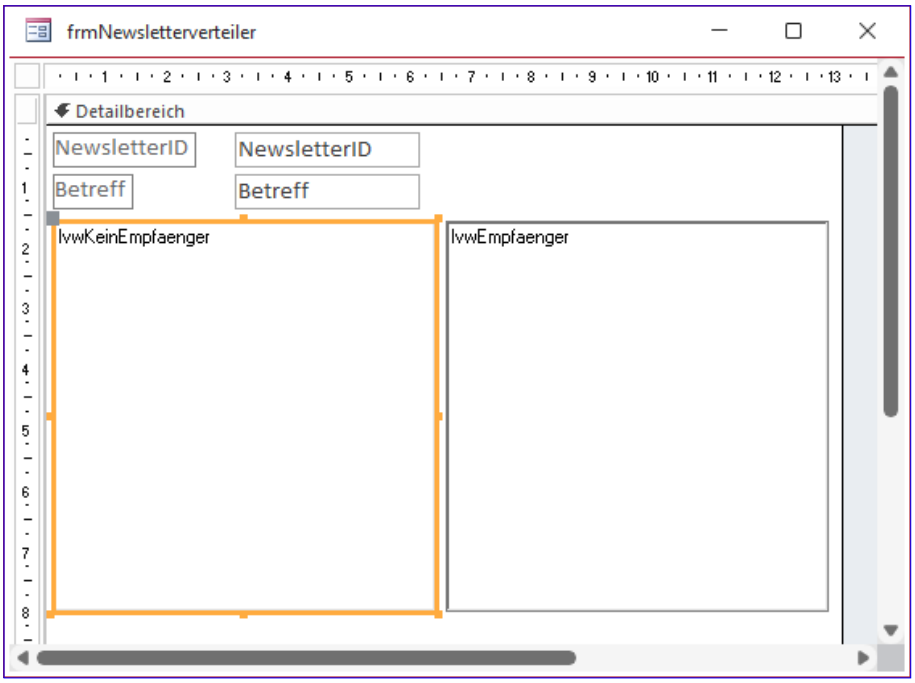


Bild 7: Hinzufügen der **ListView**-Steuerelemente

Füllen der ListView-Steuerelemente beim Anzeigen

Die beiden **ListView**-Steuerelemente sollen im Ereignis **Beim Anzeigen** gefüllt werden. Zuvor legen wir jedoch noch zwei Abfragen an, die wir als Basis für das Füllen der **ListView**-Steuerelemente nutzen.

Abfrage für die Empfänger

Die erste Abfrage heißt **qryEmpfaenger** und enthält die beiden Tabellen **tblEmpfaenger** und **tblVerteiler**. Aus der Tabelle **tblEmpfaenger** fügen wir die beiden Felder **EmpfaengerID** und **Empfaenger** hinzu, aus der Tabelle **tblNewsletter** das Feld **NewsletterID**. Das Feld **NewsletterID** dient lediglich als Kriteriumsfeld. Hier legen wir als Vergleichswert einen Parameter namens **prmNewsletterID** fest (siehe Bild 8). Auf

```
Private Sub Form_Load()
    Set objKeinEmpfaenger = Me.lvwKeinEmpfaenger.Object
    Set objEmpfaenger = Me.lvwEmpfaenger.Object
    Call ListViewEinstellen(objKeinEmpfaenger, _
        "Nicht in Empfängerliste")
    Call ListViewEinstellen(objEmpfaenger, _
        "In Empfängerliste")
End Sub
```

Die Prozedur **ListViewEinstellen** nimmt diese Parameter entgegen und stellt einige Eigenschaften ein (siehe Listing 1).

Dazu gehören die Ansicht als **lvwReport**, das Ausblenden des Randes, das Umstellen von 3-D zum flachen Erscheinungsbild, das vollständige Markieren der selektierten Zeile, das Sichtbarmachen der Markierung, auch wenn das **ListView**-Steuerelement nicht den Fokus hat, Einstellungen für die Schriftart und den Drag and Drop-Modus sowie das Einfügen einer Spaltenüberschrift.

```
Private Sub ListViewEinstellen(objListView As _
        MSComctlLib.ListView, strHeader As String)
    With objListView
        .View = lvwReport
        .BorderStyle = ccNone
        .Appearance = ccFlat
        .FullRowSelect = True
        .HideSelection = False
        .MultiSelect = True
        .Font.Name = "Calibri"
        .Font.Size = 11
        .OLEDragMode = ccOLEDragAutomatic
        .OLEDropMode = ccOLEDropManual
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , "c1", strHeader
        .ColumnHeaders(1).Width = Me.lvwEmpfaenger.Width
    End With
End Sub
```

Listing 1: Einstellen der Eigenschaften für die **ListView**-Steuerelemente

Icons per ImageList und VBA im ListView-Control

Wer einmal mit dem ListView-Steuerelement gearbeitet hat und diesem Icons hinzufügen wollte, hat diese üblicherweise erst umständlich ins ImageList-Steuerelement eingefügt, um sie dann im ListView-Steuerelement anzeigen zu können. Dabei bietet Microsoft Access seit langem die Möglichkeit, Icons komfortabel in die Tabelle namens MSysResources zu speichern. Die Icons lassen sich einfach durch Auswahl über die Eigenschaft »Bild« zu Bild-Steuerelementen oder Schaltflächen hinzufügen. In diesem Beitrag zeigen wir, wie wir die Bilder aus der Tabelle MSysResources einfach zu einem ImageList-Steuerelement addieren können, um diese dann im ListView-Steuerelement zu den einzelnen Einträgen hinzuzufügen.

Formular vorbereiten

Der erste Schritt, um Icons in einem **ListView**-Steuerelement anzeigen zu können, ist das Hinzufügen eines **ImageList**- und eines **ListView**-Steuerelements zum Formular. Diese nennen wir in diesem Beispiel **ctlImageList** und **ctlListView**.

Icons hinzufügen auf die unkomfortable Art

Die eingangs beschriebene Technik ist für Einsteiger einfach zu realisieren. Wir öffnen die Eigenschaften des **ImageList**-Steuerelements, stellen dort die Größe für die hinzuzufügenden Icons ein und fügen dann auf der Seite Images die gewünschten Icons ein – nebst der Angabe der Eigenschaft **Key**, damit wir diese referenzieren können (siehe Bild 1).

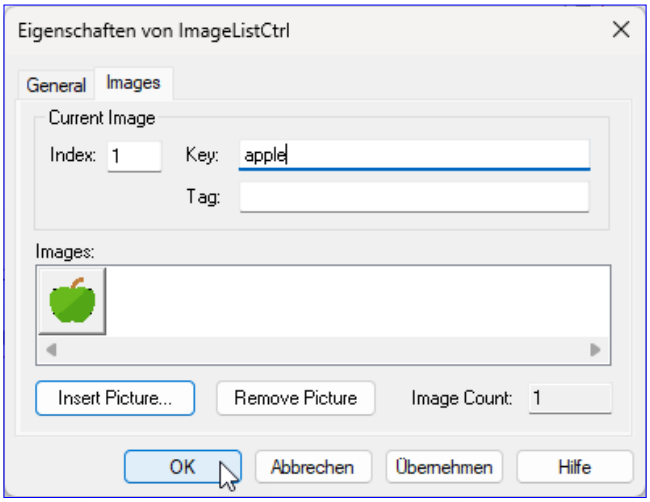


Bild 1: Herkömmliches Hinzufügen von Icons zum **ImageList**-Control

Beim Hinzufügen sehen wir bereits den ersten Nachteil: Wir können hier zwar viele verschiedene Dateiformate verwenden, aber zum Beispiel **PNG** ist nicht darunter.

Und da viele Icon-Sammlungen ihre Icons in diesem Format liefern, ist das nicht besonders praktisch. Gut, dass unsere Lösung aus diesem Beitrag dies umgeht!

Der zweite Nachteil ist, dass wir immer, wenn wir einmal ein neues Icon hinzufügen wollen, das Formular in der Entwurfsansicht öffnen und die oben beschriebenen Schritte durchführen müssen. Also schauen wir uns einen alternativen, einfacheren Weg an.

Icons zur Tabelle MSysResources hinzufügen

Dieser sieht so aus, dass wir die Icons einfach zur Tabelle **MSysResources** hinzufügen. Diese Tabelle nimmt alle Icons auf, die wir zum Beispiel über den Formularentwurf einer Schaltfläche oder einem Bild-Steuerelement hinzufügen.

Hinweis: Um die Systemtabelle **MSysResources** zu sehen, müssen wir die Systemobjekte über die Optionen des Navigationsbereichs einblenden.

Dazu klicken wir im einfachsten Fall im Formularentwurf auf den Ribbon-Eintrag **Formularentwurf|Steuerelemente|Bild einfügen** und dann auf **Durchsuchen**. Hier wählen wir das gewünschte Icon aus und fügen es so hinzu.

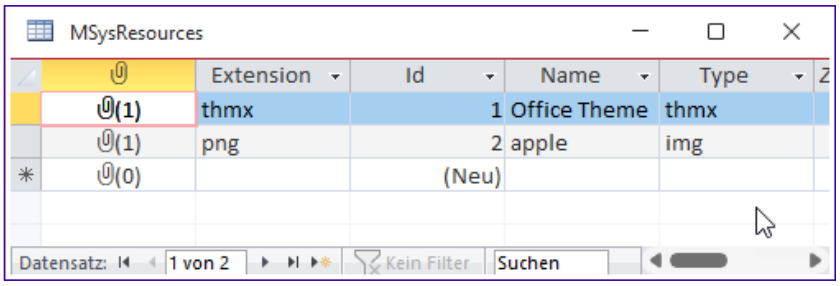


Bild 2: Die Tabelle **MSysResources** mit einer Icon-Datei

Auch ohne das anschließende Einfügen des Bildes im Formularentwurf landet die Bilddatei in der Tabelle **MSysResources**. Die Bilddatei finden wir dort wie in Bild 2 in einem neuen Datensatz vor.

Auf diese Weise fügen wir noch einige weitere Icons zur Tabelle **MSysResources** hinzu. Allerdings stellen wir hier fest, dass es etwas mühselig ist, jedes Icon einzeln hinzuzufügen. Zu diesem Problem haben wir im Beitrag **Icons in der Datenbank verwalten** (**www.access-im-unternehmen.de/1564**) bereits eine Lösung veröffentlicht.

Damit lassen sich leicht mehrere Icons auf einen Rutsch zur Tabelle **MSysResources** hinzufügen.

Icon aus MSysResources zum ImageList-Control hinzufügen

Nun stellt sich jedoch die Frage, wie wir dieses Icon in das **ImageList**-Steuerelement bekommen, damit wir es im **ListView**-Steuerelement anzeigen können.

Die entscheidende Vorarbeit dazu haben wir bereits im Beitrag **Icons in Ribbon, Kontextmenü, Formular und TreeView** (**www.access-im-unternehmen.de/1557**) geleistet. Dort haben wir Funktionen definiert, mit denen wir einfach Bilder aus der Tabelle **MSysResources** in ein **ImageList**-Steuerelement einfügen können.

Aus der Beispieldatenbank zu diesem Beitrag haben wir die folgenden Objekte in die aktuelle Beispieldatenbank importiert:

- **MDL_AMV_API**
- **MDL_AMV_Pictures**

Aus letzterem Modul kommentieren wir noch die Funktion **amvFileToResources** aus, da sich diese in einem anderen Modul befindet – beide benötigen wir hier nicht.

```
Private Sub Form_Load()
    Dim objListView As MSComctlLib.ListView
    Dim objImageList As MSComctlLib.ImageList

    Set objListView = Me.ctlListView.Object
    Set objImageList = Me.ctlImageList.Object

    objImageList.ImageHeight = 16
    objImageList.ImageWidth = 16

    Call ImageListFuellen

    With objListView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .Font.Name = "Calibri"
        .Font.Size = 11
        .View = lvwReport
        .SmallIcons = Me.ctlImageList.Object
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , "c1", "Icons"
    End With

    Call ImagesInListView
End Sub
```

Listing 1: Prozedur, die beim Laden des Formulars ausgeführt wird

ListView aus Tabellen oder Abfragen füllen, Teil 1

In den Beiträgen »Listen anzeigen mit dem ListView«-Steuerelement (www.access-im-unternehmen.de/1572) und »ListView-Steuerelement mit VBA programmieren« (www.access-im-unternehmen.de/1573) haben wir die Grundlagen zum ListView-Steuerelement und die Programmierung per VBA erläutert. Dort haben wir das ListView-Steuerelement erst einmal nur mit einfachen Beispieldaten gefüllt. Im vorliegenden Artikel gehen wir einen Schritt weiter und ziehen als Datenquelle echte Daten aus Tabellen oder Abfragen heran. Die Besonderheit des ListView-Steuerelements gegenüber dem herkömmlichen Listenfeld ist dabei, dass dieses nicht einfach an eine Datenquelle gebunden werden kann. Stattdessen müssen wir jeden einzelnen Eintrag per VBA-Code einfügen. Wie das gelingt, zeigen wir auf den folgenden Seiten.

Datenquellen für das ListView-Steuerelement

Grundsätzlich bieten sich alle Tabellen oder Abfragen als Datenquellen für die Einträge von **ListView**-Steuerelementen an.

Während wir in der Datenblattansicht von Formularen aber sogar den Inhalt von Nachschlagefeldern wie **Anrede**, **Kategorien** et cetera einfach aus der Tabelle übernehmen können, müssen wir beim **ListView**-Steuerelement genau wie beim Listenfeld hier auf eine Abfrage zurückgreifen, welche nicht nur die Werte der jeweiligen Tabelle (wie zum Beispiel **tblMitarbeiter**) berücksichtigt, sondern wir müssen auch noch die Lookup-Tabellen mit weiteren anzuzeigenden Feldern zur Abfrage hinzufügen.

Sprich: Wir können nicht einfach den Inhalt eines Fremdschlüsselfeldes wie **AnredeID** im **ListView**-Steuerelement abbilden, da dieser nur den enthaltenen Wert anzeigen würde, der den entsprechenden Datensatz in der Lookup-Tabelle referenziert.

Auch eventuelle Format-Einstellungen, die wir in der zugrunde liegenden Tabelle oder Abfrage vorgenommen haben, um beispielsweise Inhalte von Datum/Uhrzeit-Feldern darzustellen, müssen wir manuell nachbilden, da sonst nur die ursprünglichen Daten im **ListView**-Steuerelement erscheinen würden.

Mit diesen Informationen im Hinterkopf können wir gleich in das Füllen eines **ListView**-Steuerelements aus einer Tabelle oder Abfrage starten.

Icons je nach Datensatz

Ein feiner Vorteil des **ListView**-Steuerelements ist, dass wir die einzelnen Einträge mit Icons versehen können. Und dabei lassen sich sogar individuelle Icons je Datensatz realisieren. Nun stellt sich die Frage: Wenn wir beispiels-

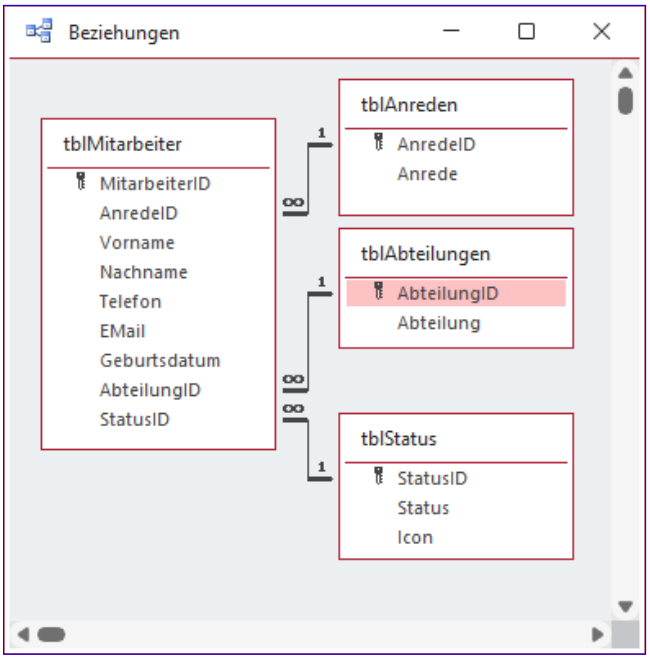
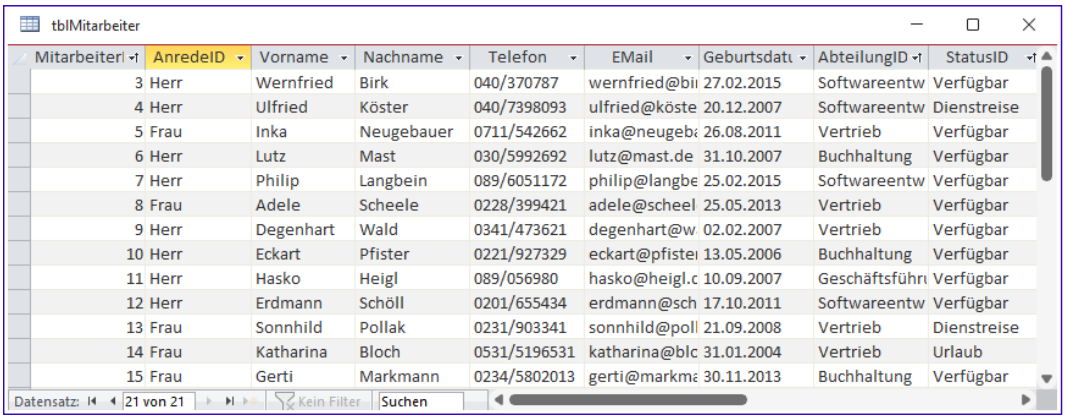


Bild 1: Datenmodell der Beispieldatenbank

weise Mitarbeiter in einem **ListView**-Steuerelement anzeigen, wozu benötigen wir dann Icons? Wir können damit beispielsweise verschiedene Status visualisieren. So ließen sich etwa Mitarbeiter, die zum aktuellen Zeitpunkt verfügbar sind, mit einem bestimmten Icon versehen.



tblMitarbeiter

MitarbeiterI	AnredeID	Vorname	Nachname	Telefon	EMail	Geburtsdatu	AbteilungID	StatusID
3	Herr	Wernfried	Birk	040/370787	wernfried@bi	27.02.2015	Softwareentw	Verfügbar
4	Herr	Ulfried	Köster	040/7398093	ulfried@köste	20.12.2007	Softwareentw	Dienstreise
5	Frau	Inka	Neugebauer	0711/542662	inka@neugeb	26.08.2011	Vertrieb	Verfügbar
6	Herr	Lutz	Mast	030/5992692	lutz@mast.de	31.10.2007	Buchhaltung	Verfügbar
7	Herr	Philip	Langbein	089/6051172	philip@langbe	25.02.2015	Softwareentw	Verfügbar
8	Frau	Adele	Scheele	0228/399421	adele@scheel	25.05.2013	Vertrieb	Verfügbar
9	Herr	Degenhart	Wald	0341/473621	degenhart@w	02.02.2007	Vertrieb	Verfügbar
10	Herr	Eckart	Pfister	0221/927329	eckart@pfiste	13.05.2006	Buchhaltung	Verfügbar
11	Herr	Hasko	Heigl	089/056980	hasko@heigl.c	10.09.2007	Geschäftsführ	Verfügbar
12	Herr	Erdmann	Schöll	0201/655434	erdmann@sch	17.10.2011	Softwareentw	Verfügbar
13	Frau	Sonnhild	Pollak	0231/903341	sonnhild@poll	21.09.2008	Vertrieb	Dienstreise
14	Frau	Katharina	Bloch	0531/5196531	katharina@blc	31.01.2004	Vertrieb	Urlaub
15	Frau	Gerti	Markmann	0234/5802013	gerti@markma	30.11.2013	Buchhaltung	Verfügbar

Datensatz: 21 von 21 | Kein Filter | Suchen

Bild 2: Daten der Tabelle **tblMitarbeiter**

Beispiel: Mitarbeitertabelle

In diesem Beitrag wollen wir die Daten einer Tabelle namens **tblMitarbeiter** samt Anreden aus der Tabelle **tblAnreden**, Abteilungen aus der Tabelle **tblAbteilungen** und **Status** aus der Tabelle **tblStatus** im **ListView**-Steuerelement anzeigen. Das Datenmodell der Anwendung sieht wie in Bild 1 aus.

Der Tabelle **tblStatus** haben wir gleich noch ein Feld namens Icon hinzugefügt, damit wir für die verschiedenen Status ein entsprechendes Icon im **ListView**-Steuerelement hinterlegen können (siehe Bild 2).

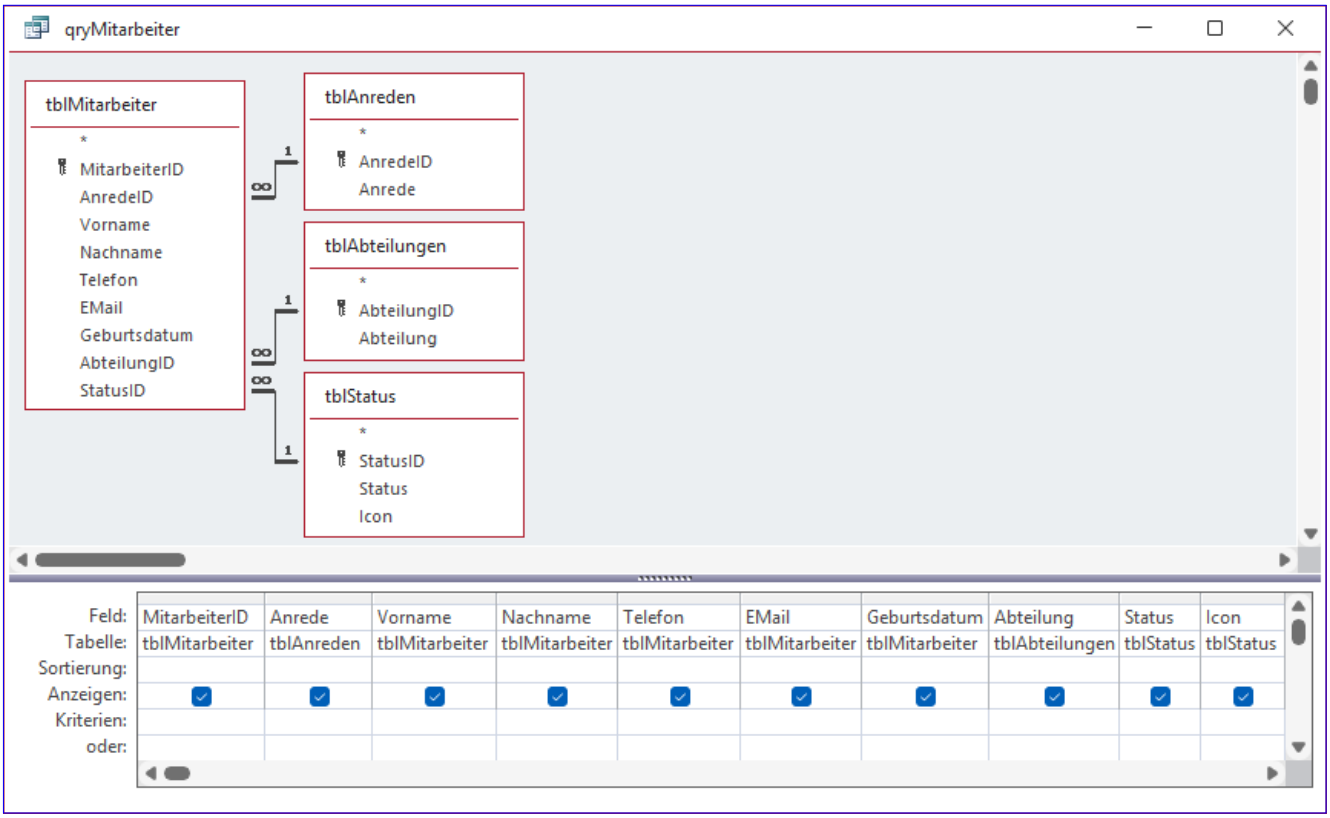


Bild 3: Entwurf der Abfrage mit allen Daten zu den Mitarbeitern

Abfrage zum Zusammenstellen der Daten für das ListView-Steuerelement

Wie eingangs erwähnt, können wir die Informationen wie die Anrede, den Status oder die Abteilung nicht im **ListView**-Steuerelement anzeigen, wenn wir nur die Felder der Tabelle **tblMitarbeiter** verwenden. Also erstellen wir eine Abfrage, die auch alle Lookup-Tabellen enthält und die wie in Bild 3 aussieht.

ListView-Steuerelement einrichten

Nun stellen wir die Eigenschaften des **ListView**-Steuerelements ein und legen die Spaltenüberschriften an. Das erledigen wir gleich beim Öffnen des Formulars im Ereignis **Beim Laden**. Dafür hinterlegen wir die Ereignisprozedur aus Listing 1. Die Prozedur deklariert eine Variable für das **ListView**-Objekt und füllt es anschließend. Dann stellt es verschiedene Eigenschaften für das **ListView**-Steuerelement ein – Ansicht, Rahmenart, Schriftart, ganze Zeile markieren und so weiter.

```
Private Sub Form_Load()
    Dim objListView As MSComctlLib.ListView

    Set objListView = Me.ctlListView.Object

    With objListView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .Font.Name = "Calibri"
        .Font.Size = 11
        .View = lvwReport
        .FullRowSelect = True
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , "c1", "ID"
        .ColumnHeaders.Add , "c2", "Anrede"
        .ColumnHeaders.Add , "c3", "Vorname"
        .ColumnHeaders.Add , "c4", "Nachname"
        .ColumnHeaders.Add , "c5", "Telefon"
        .ColumnHeaders.Add , "c6", "E-Mail"
        .ColumnHeaders.Add , "c7", "Geburtstag"
        .ColumnHeaders.Add , "c8", "Abteilung"
        .ColumnHeaders.Add , "c9", "Status"
    End With

    Call ListViewFuellen
End Sub
```

Listing 1: Einrichten des **ListView**-Steuerelements

Anschließend leert es die Auflistung der Spaltenüberschriften, um eventuell bereits vorhandene Elemente zu löschen, und legt diese neu an. Dabei orientieren wir uns an den Bezeichnungen der Felder der zu verwendenden Abfrage **qryMitarbeiter**. Anschließend ruft sie die Prozedur **ListViewFuellen** auf, um die Daten aus der Tabelle einzufügen. Das erledigen wir in einer eigenen Prozedur, damit wir die Elemente auch nach dem Hinzufügen oder Löschen von Elementen aktualisieren können.

ListView-Steuerelement mit Mitarbeiterdaten füllen

Die Prozedur **ListViewFuellen** finden wir in Listing 2. Sie öffnet ein Recordset auf Basis der Abfrage **qryMitarbeiter** im Modus **dbOpenSnapshot**, weil wir nur lesenden Zugriff benötigen und dies performanter als **dbOpenDynaset** ist.

Dann referenziert sie das **ListView**-Steuerelement und löscht eventuell vorhandene Einträge mit der **Clear**-Methode der **ListItems**-Auflistung.

Dann durchläuft sie die Datensätze des Recordsets in einer **Do While**-Schleife und legt zuerst das Hauptelement in der ersten Spalte an, hier gefüllt mit dem Feld **MitarbeiterID**. Dieses **ListItem**-Element referenzieren wir mit der Variablen **objListItem**, damit wir für dieses weitere Unterelemente für die weiteren Spalten hinzufügen können.

Diese fügen wir mit der **Add**-Methode der **ListSubItems**-Auflistung hinzu. Dabei vergeben wir für jede Spalte einen **Key**, der aus einem Buchstaben und dem Primärschlüsselwert besteht.

E-Mails senden mit CDO und Gmail

Im Beitrag »E-Mails versenden mit CDO« (www.access-im-unternehmen.de/1363) haben wir beschrieben, wie man grundsätzlich E-Mails mit der CDO-Bibliothek versenden kann. Im vorliegenden Beitrag schauen wir uns nun an, wie dies bewerkstelligt werden kann, wenn ein Gmail-Konto verwendet wird. Dazu legen wir ein Gmail-Konto an, sofern noch nicht vorhanden, holen uns das benötigte App-Passwort und stellen dann den Code zusammen, mit dem wir per VBA eine E-Mail über dieses Konto versenden können. Die auf diese Weise versendeten E-Mails können wir anschließend sogar in der Benutzeroberfläche von Gmail einsehen.

Gmail-Konto erstellen

Voraussetzung für das Versenden von E-Mails über ein Gmail-Konto ist erst einmal das Vorhandensein eines solchen Kontos. Dieses legen wir unter **gmail.com** an. Nachdem wir die grundlegenden Informationen eingegeben haben, stellen wir die gewünschte Gmail-Adresse ein (siehe Bild 1).

Anschließend legen wir das Kennwort für den Zugriff auf unser Konto fest (siehe Bild 2).

Vorneweg: Dieses Kennwort werden wir später nicht für das Versenden von E-Mails per CDO verwenden!

Nun geben wir noch eine E-Mail-Adresse an, die wir für die Wiederherstellung des Gmail-Kontos nutzen können – etwa wenn wir das Passwort vergessen haben sollten.

Und schließlich benötigt Gmail noch eine Telefonnummer, um das Konto zu schützen. Wir bekommen dann eine SMS an diese Nummer, um den Vorgang abzuschließen. Den dort enthaltenen Code geben wir anschließend unter **Telefonnummer bestätigen** ein.

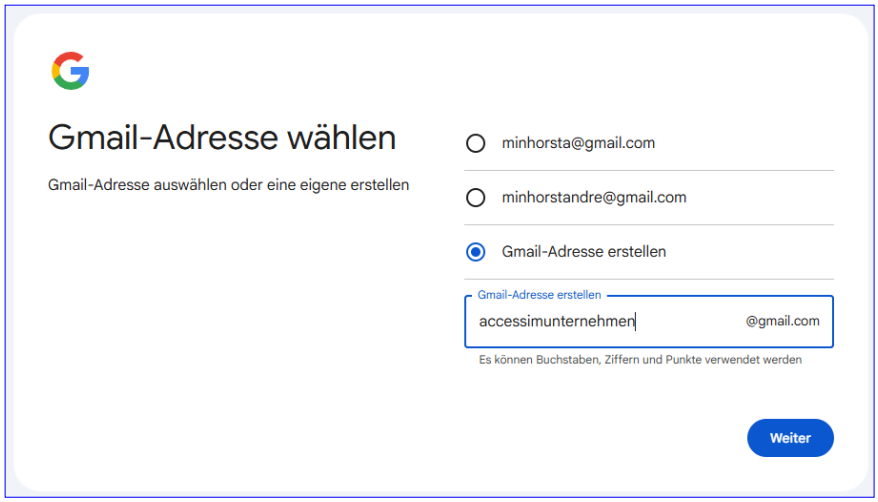


Bild 1: Anlegen eines Gmail-Kontos

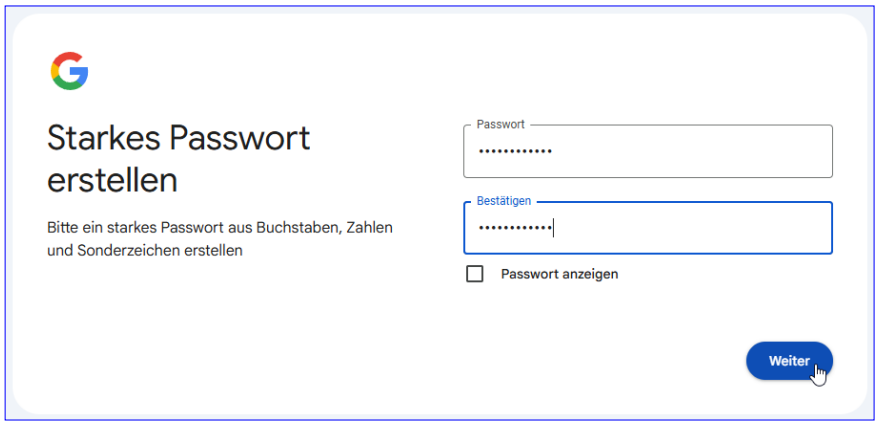


Bild 2: Erstellen eines Passworts

Danach können wir angeben, wie wir verschiedene Einstellungen vornehmen wollen. Nach dem Bestätigen der Datenschutzinformationen landen wir bereits in unserem neuen Konto.

2-Faktor-Authentifizierung aktivieren

Als Nächstes müssen wir die 2-Faktor-Authentifizierung aktivieren. Dazu gehen wir zu folgender URL:

```
https://myaccount.google.com/security
```

Hier finden wir unter **So melden Sie sich in Google an** den Hinweis, dass die 2-Faktor-Authentifizierung deaktiviert ist (siehe Bild 3). Klicken Sie dort auf den Pfeil nach rechts.

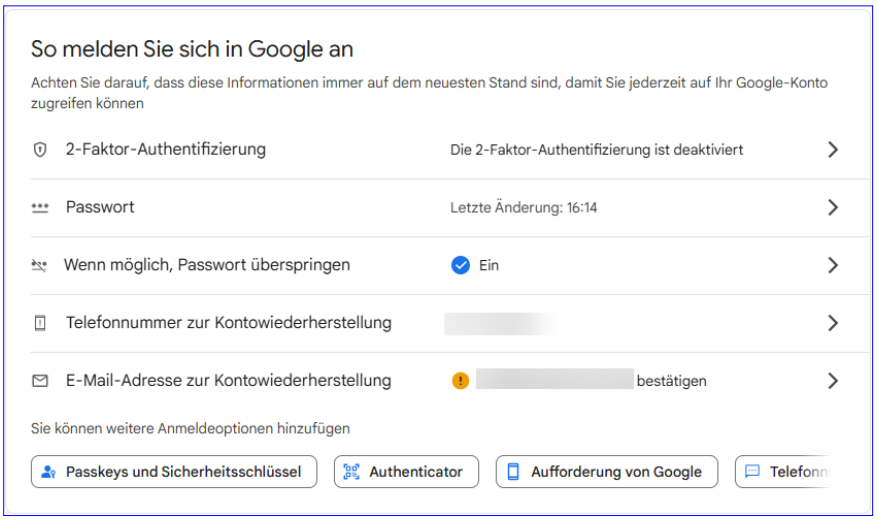


Bild 3: Sicherheitseinstellungen des Google-Kontos

Bild 4: Telefonnummer für die 2-Faktor-Authentifizierung hinzufügen

Danach müssen Sie sich noch einmal neu anmelden und landen dann im Bereich **2-Faktor-Authentifizierung**.

Hier klicken wir auf **Telefonnummer hinzufügen** (siehe Bild 4) und gelangen zurück zum vorherigen Bildschirm, wo die Aktivierung der 2-Faktor-Authentifizierung bestätigt wird.

App-Passwort holen

Nun rufen wir einen weiteren Link auf:

```
https://myaccount.google.com/apppasswords
```

Hier sehen wir die Möglichkeit, ein App-Passwort zu erstellen (siehe Bild 5).

Wir tragen den App-Namen ein und klicken auf **Erstellen**.

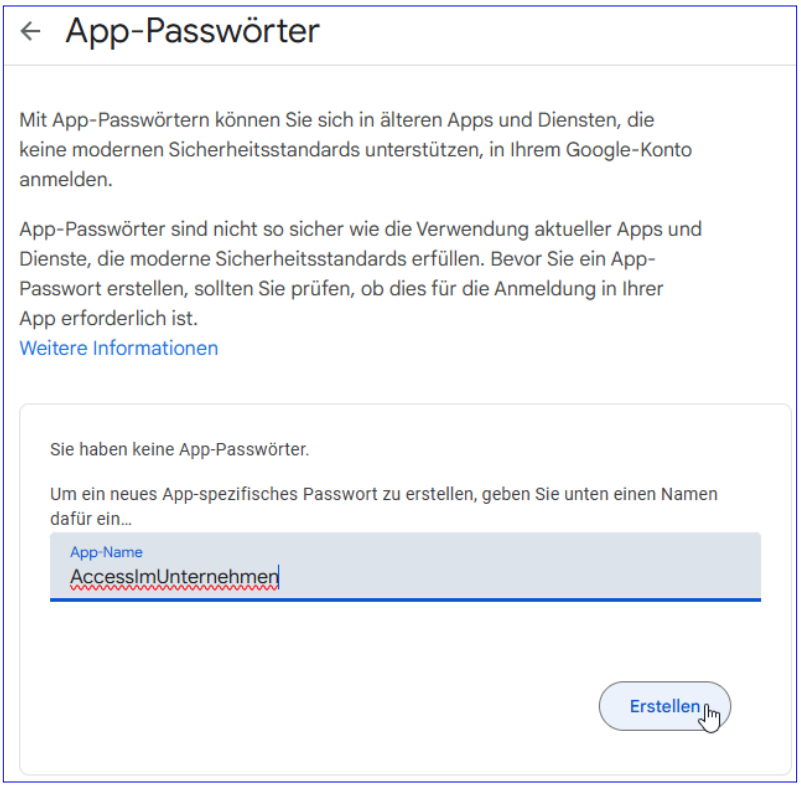


Bild 5: App-Passwort anlegen

Daten anonymisieren per Access-Add-In

Es gibt verschiedene Gründe zum Anonymisieren von Daten. Einer ist, dass man Daten aus Datenschutzgründen verfremden möchte, deren Aufbewahrungsfrist abgelaufen ist, die man aber grundsätzlich noch behalten möchte – zum Beispiel um eine Historie der Umsatzentwicklung zu erhalten und gleichzeitig die Kundendaten nicht unnötig lange vorzuhalten. Ein anderer ist, dass man eine Datenbank zur Ansicht oder für die Weiterentwicklung an einen anderen Software-Entwickler weitergeben möchte. Auch wenn es das Instrument der Vertraulichkeitsvereinbarung gibt, kann es doch nicht schaden, diese Daten erst gar nicht aus dem Haus zu geben. In diesem Beitrag stellen wir ein Add-In vor, das Access eine Funktion zum Anonymisieren von Daten hinzufügt. Das Grundgerüst dazu haben wir bereits im Beitrag »Daten anonymisieren« (www.access-im-unternehmen.de/1112) vorgestellt. Im vorliegenden Beitrag erweitern und optimieren wir diese Lösung und bauen sie so um, dass sie auch als Add-In auf die Inhalte der aktuellen Datenbank angewendet werden kann.

Im Beitrag **Daten anonymisieren** (**www.access-im-unternehmen.de/1112**) haben wir erstmals ein Beispiel dafür geliefert, wie man die Daten in Access-Tabellen anonymisieren kann. Hier haben wir nur zwei verschiedene Anonymisierungsarten genutzt. Die eine hat eine zufällige Zeichenfolge generiert, die eine bestimmte Mindest- und Höchstanzahl zufälliger Zeichen enthält. Die zweite hat die Anzahl der Zeichen beibehalten und diese lediglich verfremdet.

In der Lösung aus dem vorliegenden Beitrag gibt es verschiedene Neuerungen:

- Es gibt drei neue Anonymisiserungsarten, die bereits eingebaut sind, und für zwei weitere beschreiben wir den Ablauf, um diese selbst hinzuzufügen.
- Die Datenbank kann nun als Access-Add-In direkt in der Anwendung geöffnet werden, deren Daten anonymisiert werden sollen.
- Dafür werden direkt die Daten der aktuell geöffneten Datenbank anonymisiert.

Dies bedeutet, dass Sie eine Kopie der Anwendung anlegen müssen, deren Daten Sie dann anonymisieren und beispielsweise für die Weitergabe an einen externen Entwickler nutzen können.

Achtung: SQL-Server-Datenbank

Besondere Vorsicht ist bei Verwendung des SQL Servers geboten. Hier reicht es nicht aus, einfach eine Kopie des Frontends zu machen, von dem aus wir das Access-Add-In zum Anonymisieren öffnen. Eine Kopie des Frontends verweist nämlich ohne weitere Änderungen auf die gleiche SQL Server-Datenbank wie das Original – zumindest, wenn wir keine weiteren Änderungen vornehmen. In diesem Fall muss also eine Kopie der SQL Server-Datenbank erstellt werden und das Frontend, von dem aus die Daten anonymisiert werden sollen, muss die Tabellen der Kopie einbinden.

Neue Anonymisierungsarten

Wir haben drei neue Anonymisierungsarten hinzugefügt:

- Die erste Variante kann wie die bereits vorhandenen Methoden auf Textfelder angewendet werden. Sie

trägt einfach einen immer gleichen Ausdruck ein und kombiniert diesen mit dem Primärschlüsselwert. Daraus entstehen Werte wie **Artikel1**, **Artikel2**, **Artikel3** und so weiter, die gegebenenfalls eine bessere Grundlage zum Nachvollziehen von Zusammenhängen liefern.

- Die zweite neue Möglichkeit ist das Anonymisieren von Zahlenwerten. Hier bieten wir die Möglichkeit, für die neuen Werte eines Zahlenfeldes einen kleinsten und einen größten Wert sowie die Anzahl der Dezimalstellen festzulegen.

- Die dritte Variante ist für Datumsfelder gedacht. Auch hier können wir einen Bereich für die einzufügenden Datumsangaben festlegen.

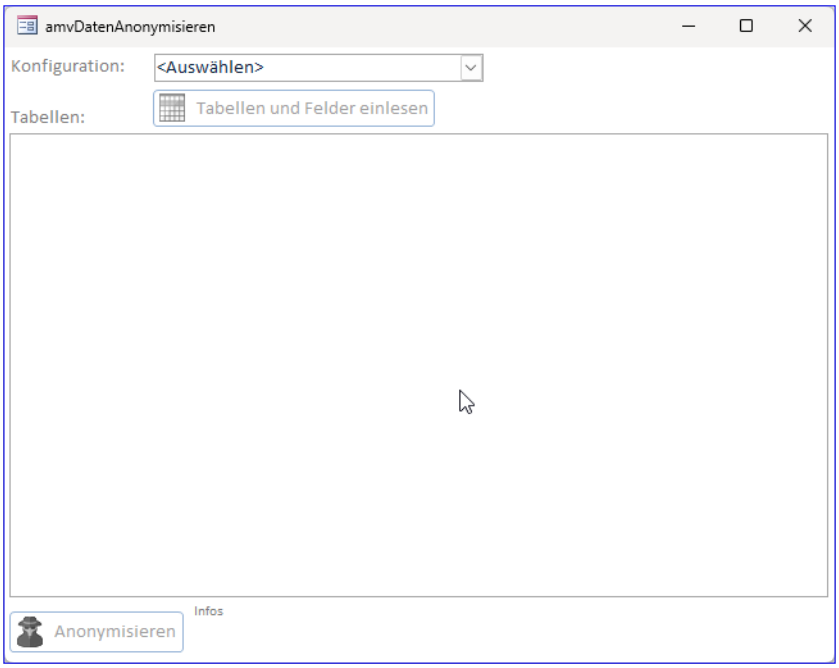


Bild 1: Der Daten-Anonymisierer nach dem ersten Start

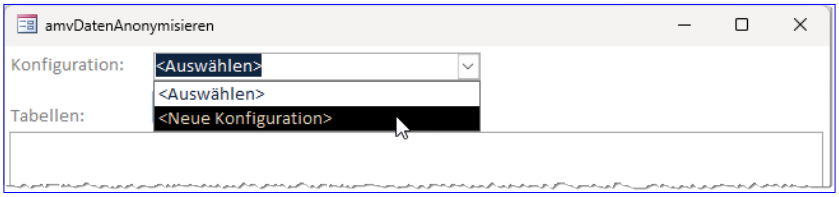


Bild 2: Anlegen einer neuen Konfiguration

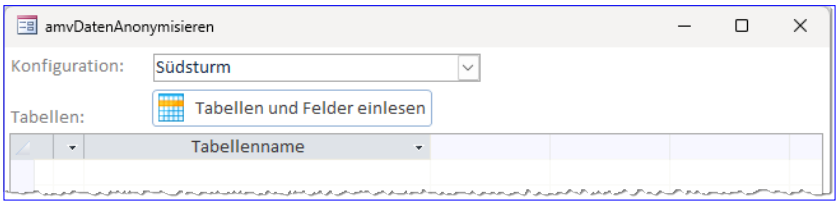


Bild 3: Die neue Konfiguration wird nun aktiviert.

amvDatenAnonymisieren installieren

Die Lösung finden Sie im Download unter dem Namen **amvDatenAnonymisieren.accda**. Die Dateiendung deutet bereits darauf hin, dass es sich um ein Access-Add-In handelt.

Dieses installieren wir, indem wir eine beliebige Access-Datenbank öffnen und den Ribbonbefehl **Datenbanktools|Add-Ins|Add-Ins|Add-In-Manager** aufrufen.

Hier klicken wir auf **Neues hinzufügen...** und wählen im folgenden Dateidialog die **.accda**-Datei aus. Danach schließen wir den Add-In-Manager. Beim erneuten Aufklappen des Add-In-Menüs sehen wir den neuen Eintrag **amvDatenAnonymisieren** und können das Add-In durch Anklicken öffnen.

Daten anonymisieren in der Praxis

Danach erscheint die Benutzeroberfläche des Add-Ins (siehe Bild 1). Hier sehen wir zunächst das Auswahlfeld **Konfiguration**.

Klicken wir es an, sehen wir einen weiteren Eintrag namens **Neue Konfiguration** (siehe Bild 2). Wählen wir diesen aus, erscheint eine Inputbox, der wir den Namen der zu erstellenden Konfiguration übergeben können.

Für unsere Beispieldatenbank **Südsturm.accda** geben wir hier beispielsweise den Namen **Südsturm** ein. So können wir später leicht erkennen, welche Konfiguration wir für welche Datenbank angelegt haben.

Nach dem Anlegen wird die neue Konfiguration automatisch selektiert und auch die Schaltfläche **Tabellen und Felder einlesen** ist nun aktiviert (siehe Bild 3).

Wenn wir diese betätigen, liest das Add-In alle Tabellen der aktuellen Datenbank mit Ausnahme derer, die mit **MSys...** beginnen ein. Zuvor erscheint der Hinweis, dass alle bestehenden Daten für diese Konfiguration gelöscht und überschrieben werden.

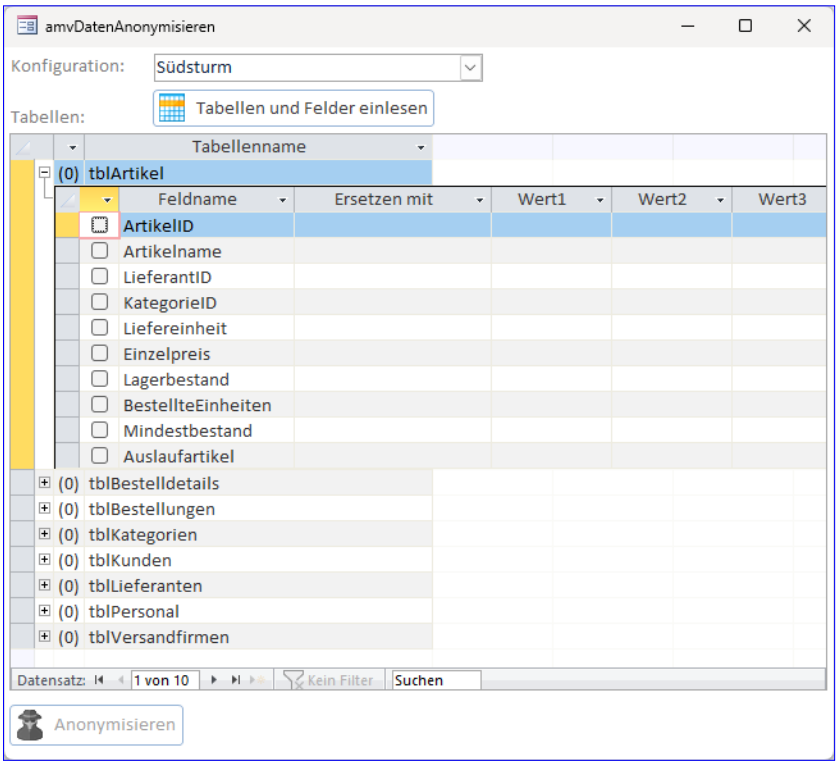


Bild 4: Die Tabellen und Felder der aktuellen Anwendung

Alle Tabellen der aktuellen Datenbank werden nun angezeigt. Klappen wir einen der Einträge auf, sehen wir die enthaltenen Felder (siehe Bild 4).

Hier können wir nun festlegen, welche Feldinhalte auf welche Art anonymisiert werden sollen.

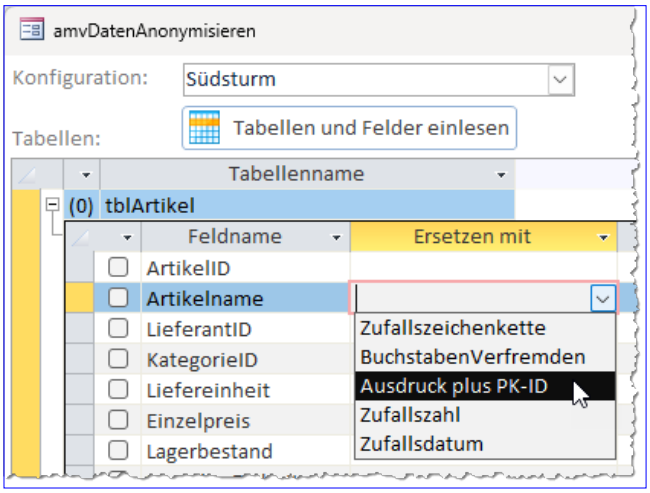


Bild 5: Auswahl einer der Anonymisierungsarten

Wenn wir das Feld **Artikelname** anonymisieren wollen, wählen wir beispielsweise den Eintrag **Ausdruck plus PK-ID** aus (siehe Bild 5).

Danach wird für diesen Eintrag das Feld **Wert1** mit dem Namen des Feldes vorbelegt. Außerdem sehen wir im Infobereich unten im Formular Hinweise zu den Bedeutungen der Felder **Wert1**, **Wert2** und **Wert3** (siehe Bild 6).

Außerdem ändert sich auf der Ebene der Tabelle die Zahl in Klammern in der ersten Spalte. Diese gibt an, wie viele Felder mit Anonymisierungen versehen wurden. Auf diese Weise können wir auch bei zugeklappten Tabellen-Einträgen schnell erkennen, hinter welchem Eintrag sich Anonymisierungen befinden.

Anonymisierung durchführen

Nun wird auch die Schaltfläche **Anonymisieren** aktiviert. Wenn wir diese betätigen, erscheint eine Rückfrage, ob die Anonymisierung durchgeführt werden soll. Bestätigen

wir dies, führt das Add-In den Auftrag aus und trägt für alle Datensätze der Tabelle **tblArtikel** im Feld **Artikel** die entsprechenden Werte ein.

Anschließend können wir die Tabelle mit den anonymisierten Daten öffnen und sehen das Ergebnis aus Bild 7. Das Feld wurde mit den Werten **Artikelname1**, **Artikelname2** und so weiter gefüllt.

Anonymisieren von Zahlenfeldern

Als Nächstes wollen wir die Werte eines Zahlenfeldes anonymisieren. Hier bietet sich das Feld **Einzelpreis** der Tabelle **tblArtikel** an (siehe Bild 8).

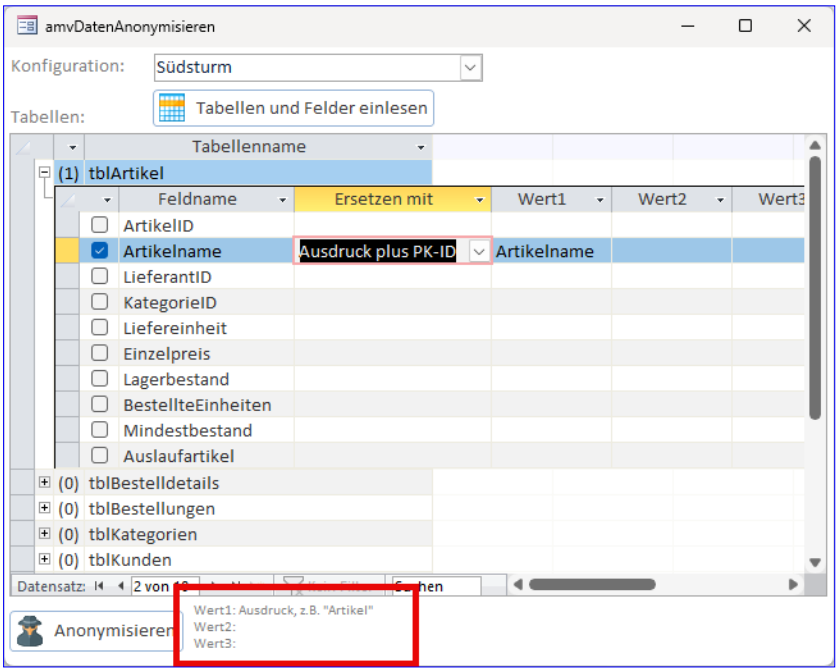


Bild 6: Informationen zur Anonymisierung

Dazu wählen wir für dieses Feld den Eintrag **Zufallszahl** aus. Die Werte sind wie folgt zu füllen:

Wert1: Kleinster Wert

Wert2: Größter Wert

Wert3: Nachkommastellen

Die entsprechenden Hinweise werden wieder im unteren Bereich des Formulars eingeblendet.

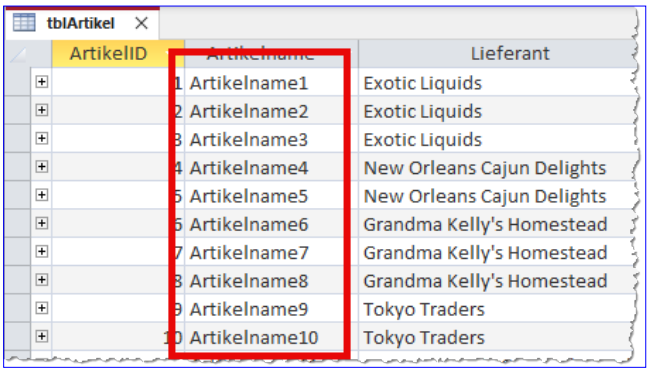


Bild 7: Tabelle mit anonymisierten Daten im Feld **Artikel**

Hier legen wir fest, dass wir Preise im Bereich von 5 bis 20 EUR mit einer Nachkommastelle eintragen wollen.

Wenn wir nun auf die Schaltfläche **Anonymisieren** klicken, wird das Feld **Artikelname** nochmals anonymisiert, allerdings mit den gleichen Werten wie zuvor. Wenn wir das nicht wollen, weil die Tabelle beispielsweise sehr viele Datensätze enthält und der Vorgang lange dauert, können wir den Haken vorn für

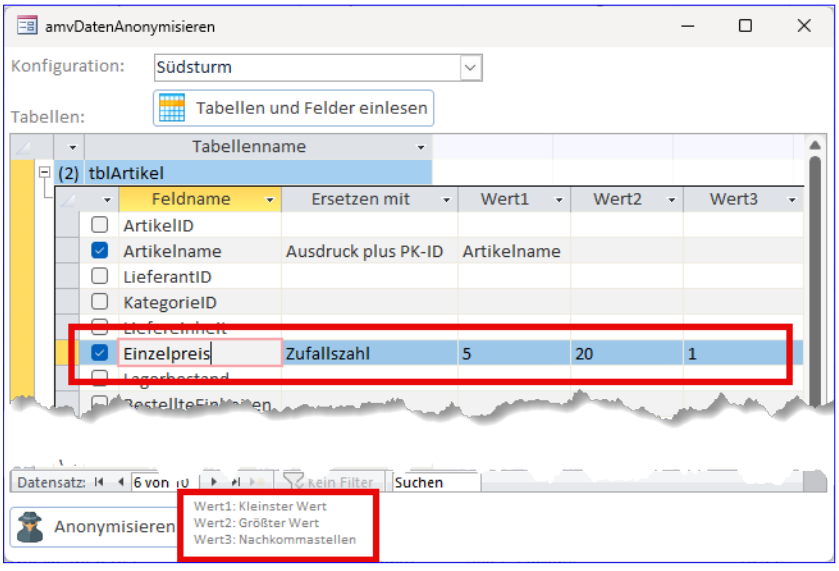
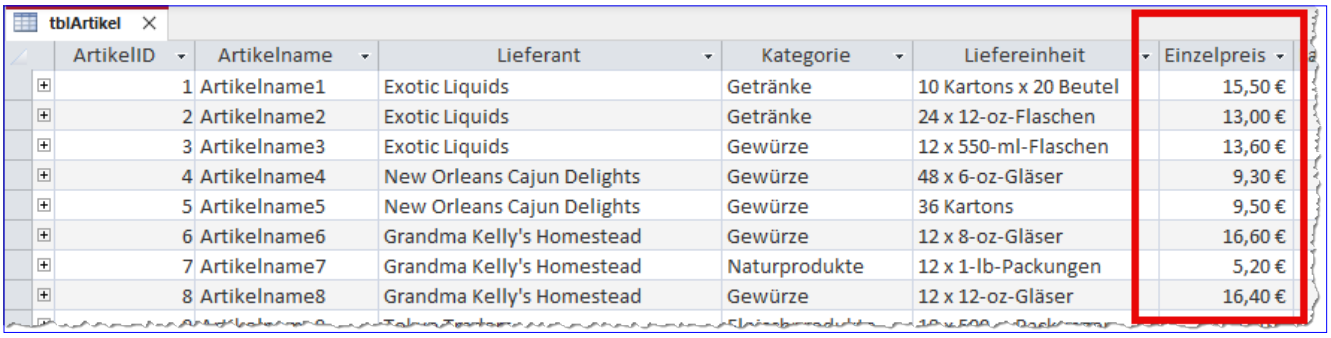


Bild 8: Anonymisieren eines Zahlenfeldes



tblArtikel

ArtikelID	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis
1	Artikelname1	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	15,50 €
2	Artikelname2	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	13,00 €
3	Artikelname3	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	13,60 €
4	Artikelname4	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	9,30 €
5	Artikelname5	New Orleans Cajun Delights	Gewürze	36 Kartons	9,50 €
6	Artikelname6	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	16,60 €
7	Artikelname7	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	5,20 €
8	Artikelname8	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	16,40 €

Bild 9: Tabelle mit anonymisierten Preisen

die Zeile mit dem Feld **Artikelname** entfernen.

Anschließend starten wir die Anonymisierung und finden das Ergebnis aus Bild 9 vor.

Datum anonymisieren

Schließlich wollen wir noch ein Datumsfeld anonymisieren. Dazu bietet sich die Tabelle **tblBestellungen** an.

Hier wählen wir für das Feld **Bestelldatum** den Eintrag **Zufallsdatum** aus und legen für die beiden Felder **Wert1** und **Wert2** das kleinste und das größte Datum fest (siehe Bild 10).

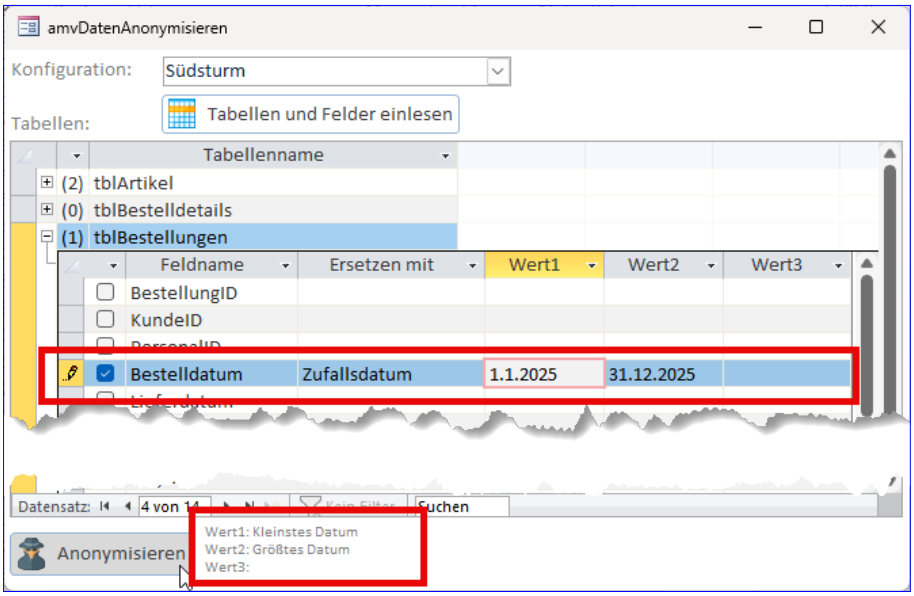


Bild 10: Anonymisieren von Datumsangaben

Klicken wir nun auf **Anonymisieren**, wird das Feld **Bestelldatum** mit zufälligen Datumsangaben gefüllt.

Beenden des Add-Ins

Wenn wir das Add-In nun schließen, bleibt die Konfiguration im Add-In gespeichert. Wir können es also später wieder aufrufen und die gespeicherte Konfiguration im Auswahlfeld oben selektieren.

Dies lädt die bereits angelegte Konfiguration mit allen Tabellen, Feldern und Anonymisierungsparametern wieder in die Liste (siehe Bild 11).

Erweitern des Add-Ins um eigene Anonymisierungsfunktionen

Wir haben nur fünf Basisfunktionen für die Anonymisierung hinterlegt. Es gibt sicher Anwendungsfälle, in denen andere Funktionen erforderlich sind. Wie die Regeln definiert sind und wie Sie neue Funktionen anlegen können,

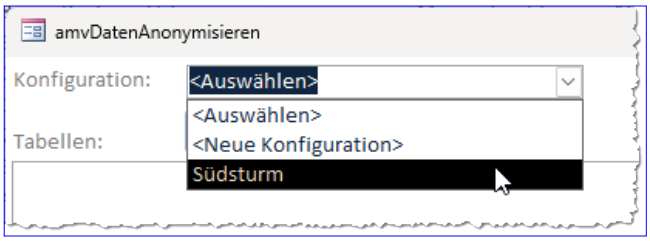


Bild 11: Wiederherstellen eines Konfiguration