E-Mails mit CDO verschicken

Wenn Sie aus der Access-Anwendung heraus E-Mails verschicken, nutzen Sie üblicherweise die Automation von Outlook. Allerdings ist Outlook nicht immer verfügbar. Gibt es denn keine andere Möglichkeit – außer ein paar veralteten SMTP-DLLs, die unter Visual Studio 6 erstellt wurden und mittlerweile nicht mehr die Sicherheitsstandards erfüllen und schon gar nicht unter der 64-Bit-Version von Access arbeiten? Doch: Es gibt die gute, alte CDO-Bibliothek. Was das ist, erfahren Sie in dieser Ausgabe.

Die CDO-Bibliothek ist standardmäßig in Windows 10 und auch im neuen Windows 11 enthalten und funktioniert in beiden Systemen reibungslos. Wir widmen dem Thema gleich drei Beiträge.

Der erste Beitrag heißt **E-Mails versenden mit CDO** (ab Seite 43) und zeigt, wie Sie einfache E-Mails mit der CDO-Bibliothek verschicken können. Hier lernen Sie auch die Grundlagen dieser Bibliothek und erfahren, wie Sie moderne Verschlüsselungstechniken für den Mailversand unterstützen können.

Im zweiten Beitrag zum Thema namens **Serienmails versenden mit CDO** erfahren Sie ab Seite 50, wie Sie SMTP-Konfigurationen für unterschiedliche Absenderkonten einrichten können und damit Serienmails an Empfänger beispielsweise aus einer Kundentabelle verschicken können. Dabei schauen wir uns auch an, wie Sie die E-Mail-Adressen der Empfänger aus der jeweiligen Tabelle auslesen.

In den bisherigen Beiträgen zum Thema CDO haben wir nur mit VBA-Code gearbeitet, im dritten mit dem Namen **Benutzeroberfläche für CDO-Serienmails** erfahren Sie, wie Sie eine praktische Benutzerüberfläche für die bisher erlernten Techniken hinzufügen können. Damit verwalten Sie die Konfigurationen, die Mailings und auch noch die einzelnen Adressaten der Mailings per Formular – dies alles ab Seite 59.

Wenn Sie schon immer einmal wissen wollten, wie Sie beim Start einer Access-Datenbank nicht nur ein Formular anzeigen, sondern sogar noch VBA-Code ausführen können, haben Sie ebenfalls gleich drei Beiträge für Sie:

- Der Beitrag **Code beim Öffnen der Anwendung: AutoExec** zeigt, wie Sie VBA-Code per Autostart-Makro aufrufen können (ab Seite 5).
- Unter **Code beim Öffnen der Anwendung: Formular** lernen Sie, wie Sie das Gleiche durch ein automatisch geöffnetes Formular erreichen können (ab Seite 7).
- Und **Code beim Öffnen der Anwendung: Ribbon** zeigt eine sicher für die meisten Entwickler neue Möglichkeit, gleich beim Öffnen Code auszuführen - mehr dazu ab Seite 9.

Und falls Sie auch noch wissen möchten, wie Sie Code beim Schließen der Anwendung ausführen – alles dazu lesen Sie unter **Code beim Schließen der Anwendung ausführen** ab Seite 11.

Spannend sind auch die beiden Beiträge zum Thema **Anzeige zuletzt verwendeter Datensätze**. Diese lassen sich etwa im Ribbon auflisten oder auch direkt in einem Listenfeld – und bieten so die Möglichkeit, die Datensätze schnell wieder anzuzeigen. Mehr dazu lesen Sie unter **Zuletzt verwendete Datensätze im Ribbon** (ab Seite 14) und **Zuletzt verwendete Datensätze per Listenfeld** (ab Seite 23).

Und nun wie immer: Viel Spaß beim Ausprobieren!

Ihr André Minhorst

Emulation im Webbrowser-Steuerelement einstellen

Wenn Sie das Webbrowser-Steuerelement nutzen, wird üblicherweise der Internet Explorer emuliert. Für viele Anwendungen ist das jedoch wenig hilfreich, denn sie funktionieren nur mit neueren Versionen des Internet Explorers oder auch nur noch mit Microsoft Edge. Dieser Beitrag zeigt, wie Sie einstellen, welche Version des Microsoft-Browsers im Webbrowser-Steuerelement verwendet wird – und Sie erfahren auch, wie Sie herausfinden, welchen Browser das Steuerelement gerade anzeigt.

Vorbereitung

Bevor wir uns anschauen, wie Sie das **Webbrowser**-Steuerelement zum Emulieren einer der Versionen des Internet Explorers oder von Microsoft Edge bewegen, legen wir als Erstes in einer Beispieldatenbank ein neues Formular namens **frmWebbrowser** an und fügen diesem ein **Webbrowser**-Steuerelement namens **ctlWebbrowser** hinzu. Um den Aufbau so praktisch wie möglich zu gestalten und unsere Versuche schnell durchführen zu können, wollen wir dafür sorgen, dass das **Webbrowser**-Steuerelement immer die folgende Webseite anzeigt:

```
https://www.whatsmybrowser.org/
```

Diese Seite liefert uns einen Hinweis darauf, welcher Browser im Webbrowser gerade emuliert wird. Damit diese direkt beim Öffnen des Formulars mit dem Webbrowser erscheint, fügen wir dem Formular außerdem ein Textfeld namens **txtURL** hinzu.

Dieses erhält die folgende Ereignisprozedur für die Ereigniseigenschaft **AfterUpdate**:

```
Private Sub txtURL_AfterUpdate()
    Me!ctlWebbrowser.ControlSource = "=""" & Me!txtURL _
        & """"
End Sub
```

Diese stellt die Eigenschaft **Steuerelementinhalt** auf die im Textfeld **txtURL** angegebene Internetadresse ein.

Nun wollen wir noch beim Laden die Adresse **https://www.whatsmybrowser.org/** einstellen und öffnen. Dazu fügen wir für die Ereigniseigenschaft **Beim Laden** noch die folgende Prozedur hinzu, welche die Adresse in das Textfeld **txtURL** schreibt und die Prozedur **txtURL_AfterUpdate** aufruft:

```
Private Sub Form_Load()
    Me!txtURL = "https://www.whatsmybrowser.org/"
    txtURL_AfterUpdate
End Sub
```

Damit das Formular direkt beim Öffnen der Datenbankdatei angezeigt wird, stellen wir in den Access-Optionen noch die Option **Formular anzeigen** im Bereich **Aktuelle Datenbank** unter **Anwendungsoptionen** auf den Namen des Formulars ein, hier **frmWebbrowser**.

Öffnen Sie die Anwendung nun, erscheint direkt das Formular und zeigt die oben genannte Webseite an.

Anzeige des verwendeten Browsers

Dies sorgt bei den Standardeinstellungen von Windows üblicherweise für die Anzeige aus Bild 1. Das ist natürlich nicht das, was wir uns wünschen – wir hätten lieber aktuellere Funktionen wie etwa von Microsoft Edge.

Anderen Browser emulieren

Wie also erreichen wir die Emulation eines anderen Browsers? Dazu sind folgende Schritte nötig:

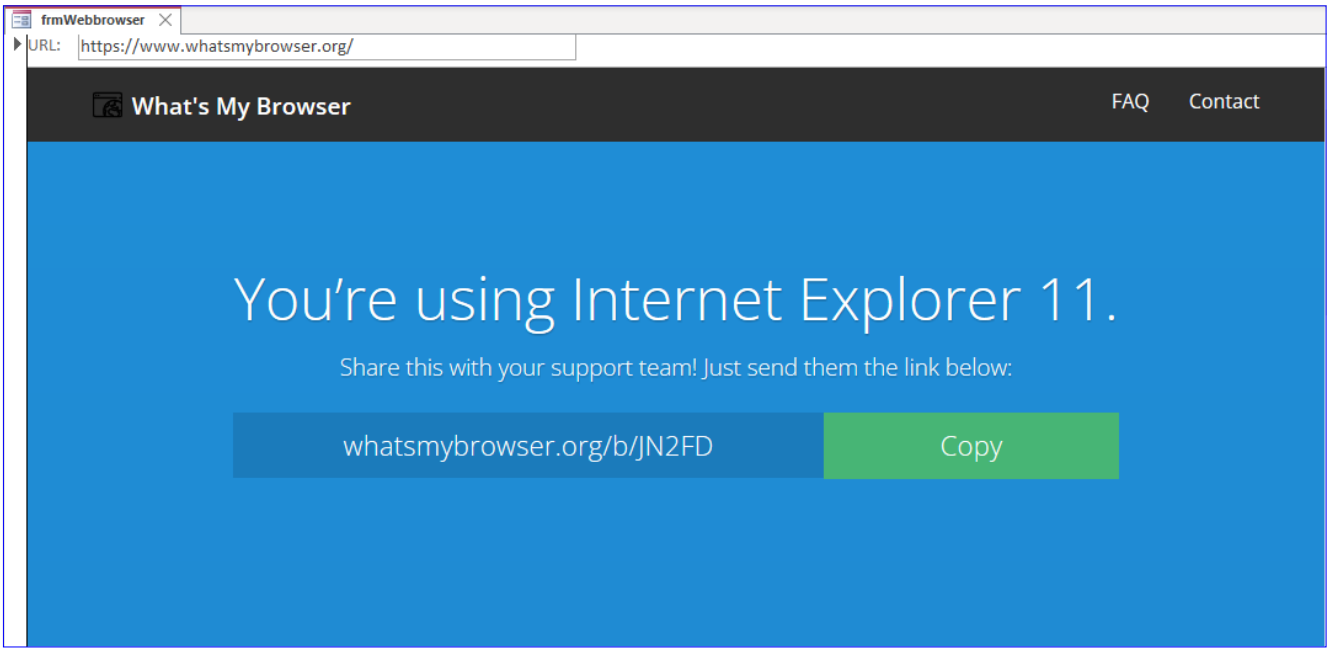

Bild 1: Anzeige des verwendeten beziehungsweise emulierten Browsers im **Webbrowser**-Steuerelement

- Alle (!) Instanzen von **MsAccess.exe** schließen. Also auch die, die gegebenenfalls unsichtbar im Hintergrund laufen. Ob das der Fall ist, erfahren Sie im Task Manager von Windows.

- In der Windows Registry einen bestimmten Wert einstellen.

- Die Beispieldatenbank erneut öffnen und schauen, ob sich der für die Emulation verwendete Browser geändert hat.

Im Detail sieht es so aus, dass Sie herausfinden müssen, welches Registry-Element den notwendigen Wert enthält. Warum das? Weil es unter Umständen mehrere gibt, welche diesen möglicherweise enthalten – mehr dazu weiter unten.

Richtigen Registry-Eintrag finden

Der Registry-Eintrag, der für die gewählte Emulation im **Webbrowser**-Steuerelement verantwortlich ist, befindet sich in einem neuen Windows 11-System an der folgenden Stelle:

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

Um diesen anzupassen, öffnen Sie zunächst die Registry. Dazu geben Sie in die Windows-Suche den Suchbegriff **RegEdit** ein und betätigen die Eingabetaste. Es erscheint der Dialog **Registrierungs-Editor**.

Hier können Sie direkt den obigen Pfad in das Textfeld im oberen Bereich eingeben und diesen mit der Eingabetaste übernehmen. Es erscheint dann das Element mit den entsprechenden Registry-Einträgen (siehe Bild 2).

Hier befindet sich ein Registry-Eintrag namens **msaccess.exe**, welcher den dezimalen Wert **11.001** enthält. Um diesen Wert zu ändern, klicken Sie doppelt auf den Eintrag **MsAccess.exe**. Es erscheint ein neuer Dialog, in dem Sie zunächst die Anzeige des Wertes von **Hexadezimal** auf **Dezimal** ändern. Dann geben Sie dort einmal den Wert **99.999** ein (siehe Bild 3).

Wenn Sie das Eingabefenster für den Wert nun schließen, können Sie die Access-Anwendung erneut öffnen. Wir

Code beim Öffnen der Anwendung: AutoExec

Wenn Sie Code beim Öffnen einer Access-Datenbank ausführen wollen, gibt es zwar keine direkte Möglichkeit wie etwa ein Ereignis beim Öffnen eines Formulars. Es gibt allerdings verschiedene Techniken, mit denen Sie dennoch den Zeitpunkt des Öffnens der Anwendung abfangen und dabei VBA-Code ausführen können. Bisher nutzte man hier vornehmlich die Möglichkeiten über das Makro namens AutoExec und das Startformular, aber es gibt noch eine weitere Möglichkeiten: nämlich über ein benutzerdefiniertes Ribbon. Im vorliegenden Beitrag schauen wir uns das AutoExec-Makro an und zeigen, wie Sie damit VBA-Code aufrufen können.

Microsoft Access bietet zwei Möglichkeiten an, um Abläufe zu programmieren und auf Ereignisse zu reagieren. Die erste und wohl eher von Einsteigern genutzte Technik sind die sogenannten Makros. Diese sind ein eigener Objekttyp und nicht mit den Makros unter Word und Excel zu verwechseln: Unter Word und Excel heißen nämlich VBA-Prozeduren, die das Wiederholen von bestimmten Abläufen ermöglichen und die auch aufgezeichnet werden können, ebenfalls Makros.

Die zweite Möglichkeit, unter Microsoft Access Abläufe zu programmieren, ist VBA. Diese Programmiersprache ist zwar komplizierter, aber wesentlich flexibler als die Makros. Wenn Sie beim Start einer Access-Anwendung jedoch VBA-Code ausführen wollen, gibt es keinen eigens dafür vorgesehenen Weg. Stattdessen ist einer der Wege eben eines der zuvor beschriebenen Makros.

Genau genommen benötigen Sie sogar ein spezielles Makro dafür, das sich durch seinen Namen von anderen Makros unterscheidet: Es muss auf jeden Fall **AutoExec** heißen. Access prüft nämlich beim Öffnen einer Datenbank, ob diese ein Makro namens **AutoExec** enthält. Falls ja, wird dieses gleich beim Starten ausgeführt. Nun haben wir ja schon beschrieben, dass die Programmierung mit Makros an sich weniger Möglichkeiten bietet und so möchten Sie vielleicht auch schon beim Start der Anwendung Prozeduren aufrufen, die in der Programmiersprache VBA geschrieben sind.

Das ist kein Problem, denn Makros bieten eine ganze Reihe verschiedener Befehle an, unter anderem auch einen, mit dem Sie eine VBA-Funktion aufrufen können. Sie finden hier also die perfekte Symbiose aus einem Makro und einer VBA-Routine.

Beim Start auszuführende Funktion definieren

Bevor wir das Makro anlegen, das beim Öffnen der Anwendung gestartet werden und unsere VBA-Funktion ausführen soll, legen wir erst einmal die VBA-Funktion an. In unserem Fall wollen wir einfach eine Meldung ausgeben, die angibt, dass sie von einer VBA-Funktion aufgerufen wurde.

In einer neuen, leeren Access-Datenbank nutzen Sie den Ribbonbefehl **Erstellen|Makros und Code|Modul**, um ein neues Standardmodul anzulegen und im VBA-Editor zu öffnen. Hier fügen wir nun die folgende Funktion ein:

```
Public Function Startup()
    MsgBox "Meldung per VBA"
End Function
```

Diese Funktion können Sie durch Platzieren der Einfügemarke innerhalb des Funktionscodes und Betätigen der Taste **F5** starten und sich von der Funktion überzeugen. Sollte dies nicht klappen, ist die Anwendung vielleicht noch nicht als vertrauenswürdig eingestuft – das müssten Sie dann prüfen und korrigieren.

Code beim Öffnen der Anwendung: Formular

Wenn Sie Code beim Öffnen einer Access-Datenbank ausführen wollen, gibt es zwar keine direkte Möglichkeit wie etwa ein Ereignis beim Öffnen eines Formulars. Es gibt allerdings verschiedene Techniken, mit denen Sie dennoch den Zeitpunkt des Öffnens der Anwendung abfangen und dabei VBA-Code ausführen können. Bisher nutzte man hier vornehmlich die Möglichkeiten über das Makro namens AutoExec und das Startformular, aber es gibt noch eine weitere Option: nämlich über ein benutzerdefiniertes Ribbon. Im vorliegenden Beitrag zeigen wir zunächst, wie Sie über das als Startformular definierte Formular VBA-Code ausführen können.

Neben dem **AutoExec**-Makro, dessen Einsatz wir im Beitrag **Code beim Öffnen der Anwendung: AutoExec** (**www.access-im-unternehmen.de/1367**) beschreiben, gibt es mit dem Startformular noch eine weitere Option.

Dabei gehen wir in dem oben angegebenen Beitrag davon aus, dass wie eine Funktion wie die folgende beim Start der Anwendung aufrufen wollen:

```
Public Function Startup()
    MsgBox "Meldung per VBA"
End Function
```

Statt der **MsgBox**-Anweisung, die hier angegeben ist, können Sie die beim Start Ihrer Anwendung notwendigen Anweisungen einfügen.

Startformular erstellen

Wie aber können wir diese VBA-Funktion nun mithilfe eines Formulars aufrufen? Dazu erstellen wir als Erstes einmal das benötigte Formular und speichern es unter dem Namen **frmStartup**.

Dann legen Sie für das Ereignis **Bei Laden** des Formulars den Wert **[Ereignisprozedur]** fest und klicken auf die Schaltfläche mit den drei Punkten (**...**), um die Ereignisprozedur im Klassenmodul des Formulars anzulegen (siehe Bild 1).

Im VBA-Editor erscheint die Prozedur, die wir wie folgt ergänzen:

```
Private Sub Form_Load()
    Call Startup
End Sub
```

Diese Prozedur ruft nun beim Laden des Formulars unsere Startfunktion auf. Das testen wir, indem Sie das Formular in der Formularansicht anzeigen. Dies sollte zuerst die

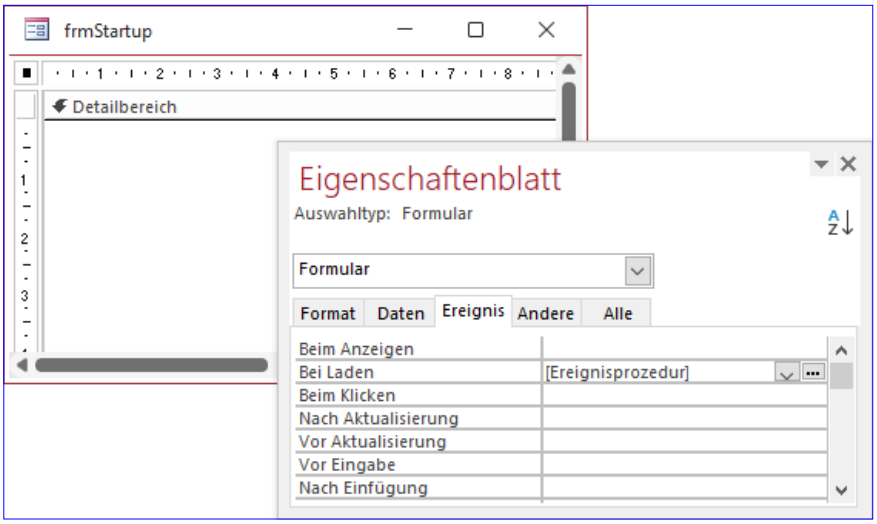

Bild 1: Das Startformular

Code beim Öffnen der Anwendung: Ribbon

Wenn Sie Code beim Öffnen einer Access-Datenbank ausführen wollen, gibt es zwar keine direkte Möglichkeit wie etwa ein Ereignis beim Öffnen eines Formulars. Es gibt allerdings verschiedene Techniken, mit denen Sie dennoch den Zeitpunkt des Öffnens der Anwendung abfangen und dabei VBA-Code ausführen können. Bisher nutzte man hier vornehmlich die Möglichkeiten über das Makro namens AutoExec und das Startformular, aber es gibt noch eine weitere Option: nämlich über ein benutzerdefiniertes Ribbon. Im vorliegenden Beitrag zeigen wir, wie Sie eine VBA-Funktion beim Starten unter Verwendung des Ribbons aufrufen können.

Neben dem **AutoExec**-Makro, dessen Einsatz wir im Beitrag **Code beim Öffnen der Anwendung: AutoExec** (**www.access-im-unternehmen.de/1367**) beschreiben, gibt es mit dem Startformular noch eine weitere Option – siehe **Code beim Öffnen der Anwendung: Formular** (**www.access-im-unternehmen.de/1368**).

Es gibt seit der Einführung des Ribbons jedoch noch eine weitere Möglichkeit, gleich beim Öffnen der Datenbank VBA-Code auszuführen. Diese hier ist zwar etwas aufwendiger, aber wenn Sie ohnehin ein Ribbon beim Start der Datenbankanwendung anzeigen, ist nur eine einzige zusätzliche Zeile zum Aufrufen der Startfunktion nötig. Außerdem sparen Sie so ein **AutoExec**-Makro beziehungsweise ein beim Starten der Anwendung zu öffnendes Formular ein.

Startfunktion

Wir gehen in den beiden oben angegebenen Beiträgen davon aus, dass wir eine Funktion wie die folgende beim Start der Anwendung aufrufen wollen:

```
Public Function Startup()
    MsgBox "Meldung per VBA"
End Function
```

Um diese automatisch beim Start aufzurufen und dabei ein Ribbon statt des **AutoExec**-Makros oder eines Startformulars zu nutzen, benötigen Sie zunächst eine minimale Ribbondefinition. Diese sieht wie folgt aus:

```
?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_Startup"/>
```

Diese Definition fügen Sie einer Tabelle namens **USysRibbons** hinzu, deren Entwurf wie in Bild 1 aussieht. Für das Feld **RibbonName** stellen wir einen eindeutigen Index ein. Wenn Sie diese Tabelle so angelegt und unter dem Namen **USysRibbons** gespeichert haben, wird diese möglicherweise nicht im Navigationsbereich angezeigt – das liegt daran, dass die Tabelle mit diesem Namen als Systemtabelle erkannt und ausgeblendet wird. Sie können diese dann entweder mit dem folgenden Befehl öffnen oder die Anzeige der Systemobjekte in den Optionen aktivieren:

```
DoCmd.OpenTable "USysRibbons"
```

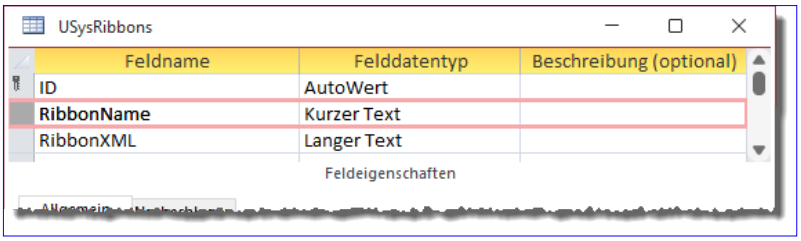

Bild 1: Entwurf der Ribbontabelle **USysRibbons**

Code beim Schließen der Anwendung ausführen

Beim Öffnen einer Access-Anwendung haben Sie verschiedene Möglichkeiten, Code auszuführen – Sie können ein Formular anzeigen, das beim Öffnen Code ausführt, das AutoExec-Makro nutzen, um eine Prozedur aufzurufen oder sogar das Ribbon dafür instrumentalisieren. Wenn es jedoch um Code geht, der beim Beenden der Anwendung ausgeführt werden soll, finden wir keine offiziell dafür vorgesehene Technik. Allerdings gibt es einen Trick, um doch noch VBA-Code auszuführen, wenn die Access-Anwendung durch den Benutzer geschlossen wird.

Code beim Starten der Anwendung ausführen

Wie Sie Code ausführen, wenn der Benutzer die Datenbank öffnet, haben wir bereits in drei Beiträgen gezeigt. Das gelingt auf folgende Arten:

- Mit dem Autostart-Makro: **Code beim Öffnen der Anwendung: AutoExec** (**www.access-im-unternehmen.de/1367**)
- Mit dem Startformular: **Code beim Öffnen der Anwendung: Formular** (**www.access-im-unternehmen.de/1368**)
- Und es gelingt sogar von einem Ribbon aus, das beim Start angezeigt wird: **Code beim Öffnen der Anwendung: Ribbon** (**www.access-im-unternehmen.de/1369**)

Code beim Schließen der Anwendung ausführen

Wenn es um das Ausführen von Code beim Schließen einer Anwendung geht, finden wir allerdings wesentlich weniger Möglichkeiten.

Auch hier können wir schon vorher sagen: Es gibt kein spezielles Ereignis etwa für die Datenbank, das Sie nutzen können, um Code beim Schließen der Datenbank auszuführen.

Stattdessen behelfen wir uns mit einem Trick. Dieser sieht kurzgefasst folgendes vor:

- Wir öffnen beim Starten ein Formular, das wir aber gleich wieder ausblenden.
- Diesem Formular fügen wir eine Ereignisprozedur für das Ereignis **Beim Entladen** hinzu.
- Wenn der Benutzer nun die Anwendung schließen will, muss zuvor das noch offene, aber ausgeblendete Formular geschlossen werden.
- Das Schließen des Formulars erfolgt automatisch, wenn die Datenbank geschlossen wird, was nach sich zieht, dass auch die **Beim Entladen**-Prozedur des Formulars noch ausgeführt wird.

Beim Schließen-Code Schritt für Schritt

Schauen wir uns im Detail an, was nötig ist, um beim Schließen der Datenbank VBA-Code auszuführen.

Die erste Aktion ist das Erstellen eines Formulars, das direkt beim Start der Anwendung geöffnet und ausgeblendet wird. Dieses Formular nennen wir **frmStart** und es enthält keinerlei Elemente – das ist auch nicht nötig, da der Benutzer dieses nie zu Gesicht bekommt.

Die erste Frage ist: Wie sorgen wir dafür, dass das Formular direkt nach dem Start ausgeblendet wird? Im Gegensatz zu Steuerelementen finden wir in den Formulareigenschaften keine Eigenschaft namens **Sichtbar**, also müssen wir das Formular per VBA ausblenden. Das

erledigen wir so früh wie möglich, damit das Formular noch nicht einmal aufblitzt. Welche ist die erste Ereignisprozedur, wo wir diesen Schritt durchführen können?

Um das herauszufinden, haben wir für die in Frage kommenden Ereigniseigenschaften jeweils eine leere Ereignisprozedur angelegt und diese mit einem Haltepunkt versehen. Das Ergebnis ist, dass das Ereignis **Beim Öffnen** als erstes ausgelöst wird (siehe Bild 1).

Also fügen wir hier die Anweisung zum Ausblenden des Formulars ein, was wie folgt aussieht:

```
Private Sub Form_Open(Cancel As Integer)
    Me.Visible = False
End Sub
```

Allerdings gelingt das nicht auf diese Weise, und auch nicht durch Einfügen der Prozedur in einer der übrigen im Bild zu findenden Ereignisprozeduren. Der Grund ist einfach und wir bestätigen diesen, indem wir uns jeweils den Wert der Eigenschaft **Visible** ausgeben lassen – zum Beispiel so:

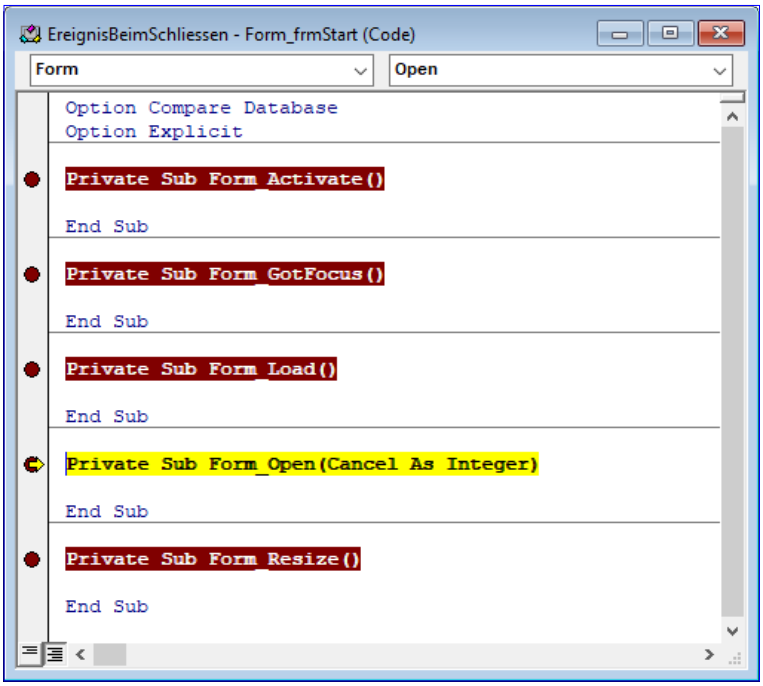

Bild 1: Ermitteln der ersten Prozedur beim Öffnen eines Formulars

```
Private Sub Form_Load()
    Debug.Print "Load: " & Me.Visible
End Sub
```

Hiermit stellen wir schnell fest, dass die **Visible**-Eigenschaft hier ohnehin noch den Wert **False** aufweist, das Formular also noch gar nicht sichtbar ist. Eine Idee wäre noch, das **Timer**-Ereignis zu nutzen, aber bis dieses erstmalig aufgerufen wird, wurde auch das Formular bereits einmal eingeblendet.

Wir können das Formular also für diesen Zweck nicht für die Option **Startformular** in den Optionen der Datenbank festlegen.

Öffnen des Formulars im ausgeblendeten Modus

Wenn wir das Formular schon nicht direkt beim Öffnen unsichtbar machen können, dann müssen wir einen anderen Weg finden, das Formular direkt unsichtbar zu machen. Die einfachste Variante ist, die **DoCmd.OpenForm**-Methode mit einem speziellen Parameter zu nutzen. Dieser heißt **WindowMode**.

Normalerweise nutzen wir diesen mit dem Wert **acDialog**, um das Formular als modalen Dialog zu öffnen. Nun aber wollen wir das Formular ausgeblendet öffnen und dazu nutzen wir den Wert **acHidden**. Der Aufruf sieht wie folgt aus:

```
DoCmd.OpenForm "frmStart", WindowMode:=acHidden
```

Wenn Sie diesen Aufruf im Direktbereich absetzen, haben Sie ein Problem: Wie finden Sie nun heraus, ob das Formular geöffnet wurde? Das fragen wir über die Auflistung **Forms** ab. Diese sollte für die Eigenschaft **Count** den Wert **1** liefern, sofern keine anderen Formulare geöffnet sind. Und das ist hier der Fall:

```
? Forms.Count
 1
```

Zuletzt verwendete Datensätze im Ribbon

Im Beitrag "Zuletzt verwendete Datensätze per Listenfeld" zeige wir, wie Sie die zuletzt in einem Formular angezeigten Kundendatensätze in einem Listenfeld aufführen können, um diese schnell wieder zu öffnen. Vielleicht haben Sie im Formular zur Kundenverwaltung aber keinen Platz für diese Liste oder Sie möchten diese einfach immer verfügbar haben. Dann bietet sich das Ribbon als Ort für diese Liste an. Im vorliegenden Beitrag zeigen wir, wie Sie das Ribbon um ein Steuerelement zur Anzeige und Auswahl der zuletzt verwendeten Datensätze erweitern.

Als Formular zur Anzeige der Kundendaten verwenden wir das gleiche Formular, das wir im Artikel **Zuletzt verwendete Datensätze per Listenfeld** (**www.access-im-unternehmen.de/1365**) erstellt haben. Auf diese Weise können wir gleich abgleichen, ob das Ribbonsteuerelement die richtigen Datensätze anzeigt. Das Formular sieht wie in Bild 1 aus.

Welches Ribbonsteuerelement für die zuletzt verwendeten Datensätze?

Als Erstes stellt sich die Frage, welches der Ribbonsteuerelemente am besten für unsere Aufgabe geeignet ist. Eigentlich sind das alle Steuerelemente, die Listen von Einträgen anzeigen können, und dafür kommen die folgenden in Frage:

- **comboBox**
- **dropDown**
- **menu**
- **dynamicMenu**
- **gallery**
- **splitButton**

Und Sie können sogar, wenn Sie eine begrenzte Menge von zuletzt verwendeten Datensätzen anzeigen wollen, für jeden eine Schaltfläche anzeigen und so alle interessanten Datensätze gleichzeitig bereitstellen.

Wichtig ist nur, dass wir die Anzeige mit jedem neu im Formular verwendeten Datensatz aktualisieren, und das ist mit all diesen Elementen möglich. Also schauen wir uns anhand verschiedener Beispielen an, wie das gelingt.

comboBox oder dropDown?

Die beiden Steuerelemente **comboBox** und **dropDown** haben verschiedene Eigenschaften, die sie für verschiedene Aufgaben prädestinieren. Wenn es darum geht, nicht nur den im jeweiligen Element ausgewählten Text zu ermitteln, sondern auch noch einen weiteren Wert wie beispielsweise einen Index oder eine Id für diesen Wert, dann ist das **dropDown**-Element die erste Wahl.

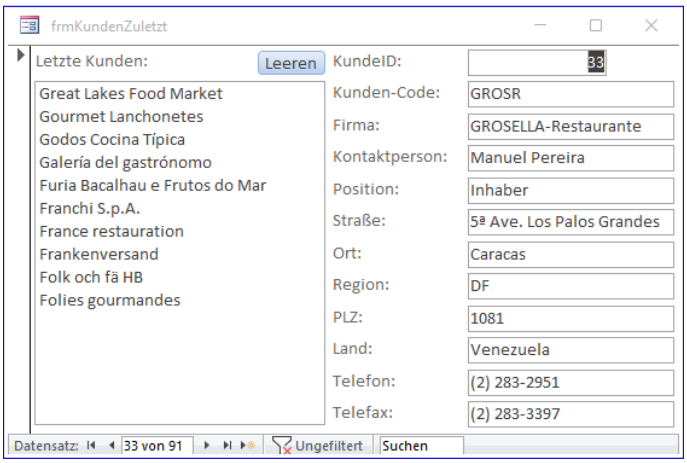

Bild 1: Die in diesem Formular zuletzt angezeigten Datensätze sollen auch per Ribbon angezeigt werden.

Also verwenden wir dieses als Erstes. Bevor wir es programmieren, benötigen wir jedoch noch eine Datensatzherkunft für die anzuzeigenden Elemente.

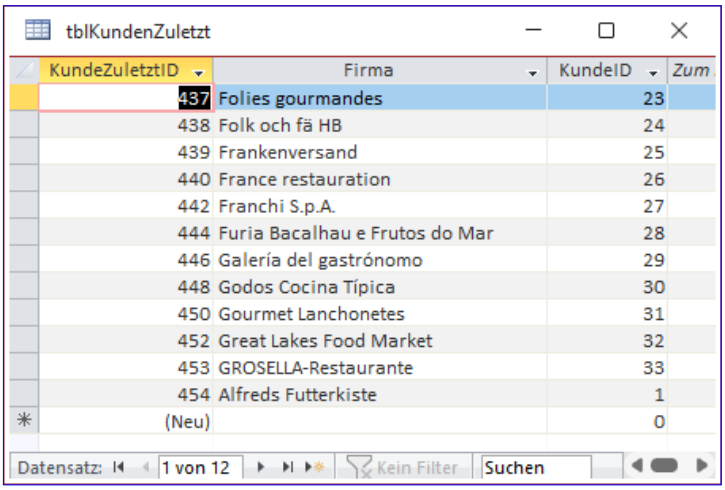

Bild 2: Tabelle mit den zuletzt angezeigten Datensätzen

Abfrage für die zuletzt angezeigten Datensätze

In der oben beschriebenen Lösung haben wir alle Datensätze der Tabelle **tblKunden**, die im Formular **frmKundenZuletzt** angezeigt wurden, in die Tabelle **tblKundenZuletzt** eingetragen. Diese sah dann nach dem Durchlaufen einiger Datensätze wie in Bild 2 aus.

Das Listenfeld, das die zehn zuletzt verwendeten Datensätze angezeigt hat, nutzt eine Abfrage, um diese in der richtigen Reihenfolge aus dieser Tabelle auszulesen, also in umgekehrter Reihenfolge zum Anlegezeitpunkt.

Außerdem hatte diese Abfrage ein Kriterium, das dafür gesorgt hat, dass der aktuell im Formular angezeigte Datensatz nicht in der Liste erscheint, denn dieser Datensatz ist ja schon sichtbar.

Diese Abfrage, die direkt für das Listenfeld hinterlegt war, kopieren wir nun in eine neue Abfrage namens **qryKundeZuletzt** (siehe Bild 3). Danach entfernen wir das dort markierte Kriterium.

Warum das? Weil es sein kann, dass das Ribbon auch angezeigt wird, obwohl das Formular gar nicht sichtbar ist.

Dann ist es natürlich wichtig, dass auch der zuletzt angezeigte Eintrag zur Auswahl steht.

Anlegen des Ribbons zur Anzeige des dropDowns

Um das Ribbon anzulegen, sind einige Schritte notwendig:

- Hinzufügen einer Tabelle namens **USysRibbons**
- Eintragen der Ribbondefinition in diese Tabelle
- Hinzufügen der Callbackprozeduren für die Funktionalität des Ribbons
- Festlegen der Ribbondefinition als Startribbon

Ribbontabelle mit Inhalt anlegen

Die Ribbontabelle ist eine Tabelle namens **USysRibbons**. Sie enthält die drei Felder **RibbonID** (Primärschlüsselfeld

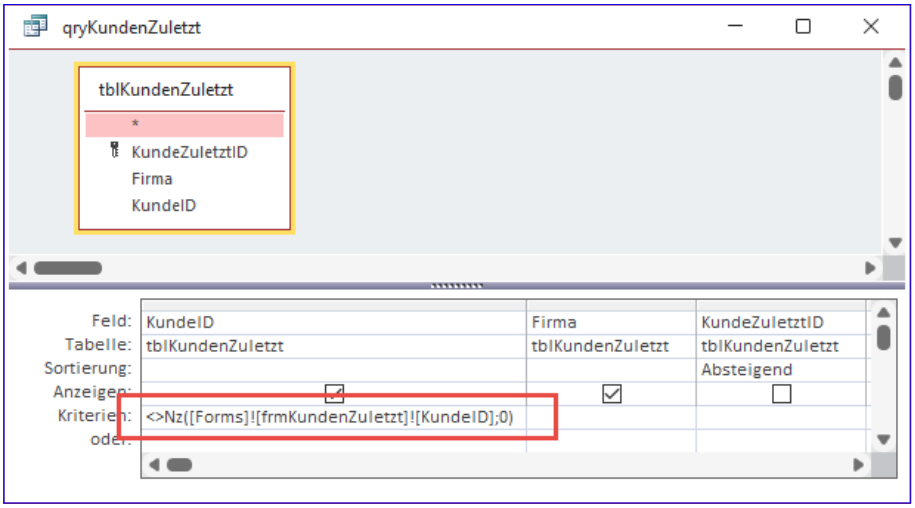

Bild 3: Abfrage zum Ermitteln der Datensätze in der richtigen Reihenfolge

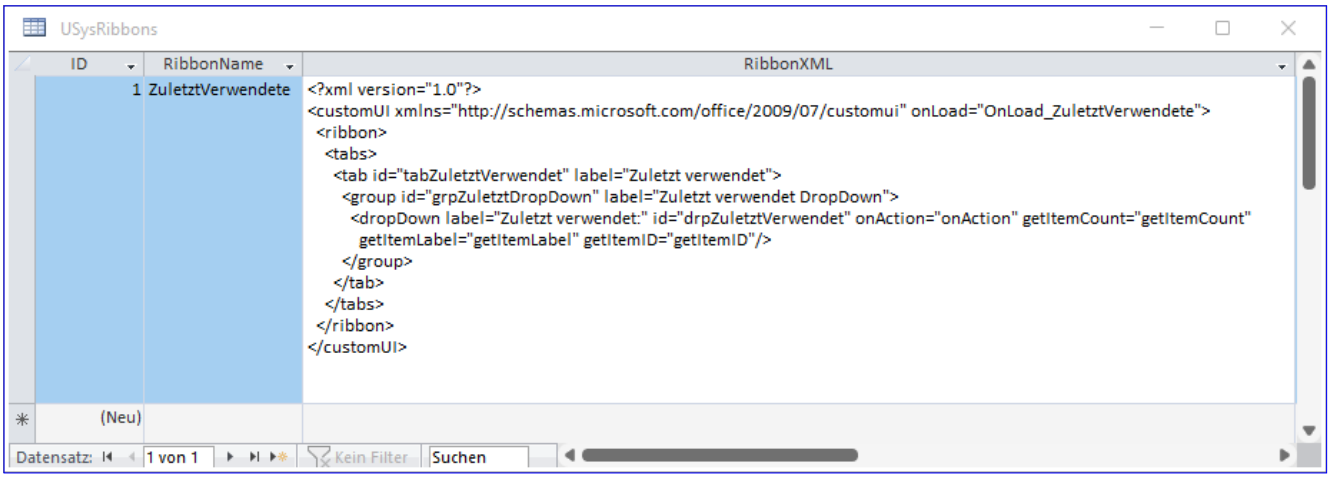

Bild 4: Die Tabelle **USysRibbons** mit der Ribbondefinition für die Anzeige der zuletzt angezeigten Datensätze im **dropDown**-Element

mit Autowert), **RibbonName** (Datentyp **Kurzer Text**) und **RibbonXML** (Datentyp **Langer Text**).

Ribbondefinition für das dropDown-Element erstellen

Die Ribbondefinition fügen wir in einen neuen Datensatz dieser Tabelle ein, für den wir außerdem den Wert **ZuletztVerwendete** im Feld **RibbonName** festlegen. Die Tabelle sieht dann wie in Bild 4 aus.

Die Ribbondefinition sehen Sie im Detail in Listing 1. Die meisten Elemente dienen der Herstellung der Struktur aus **tab**- und **group**-Element. Interessant ist das **dropDown**-Element mit den angegebenen Ereignisattributen. Hier legen wir die Prozeduren fest, die zum Füllen des **dropDown**-Elements nötig sind.

Ribbon-Verweis beim Laden speichern

Beim erstmaligen Anzeigen des Ribbons können wir das Ereignisattribut **onLoad** nutzen, um eine Prozedur anzugeben, die beim Laden des Ribbons ausgelöst wird. Hier geben wir den Wert **onLoad_ZuletztVerwendete** an.

Den Verweis auf die Ribbondefinition wollen wir in einer Variablen namens **objRibbon_ZuletztVerwendete** mit dem Datentyp **IRibbonUI** speichern. Dieser Datentyp ist

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_ZuletztVerwendete">
  <ribbon>
    <tabs>
      <tab id="tabZuletztVerwendet" label="Zuletzt verwendet">
        <group id="grpZuletztDropDown" label="Zuletzt verwendet DropDown">
          <dropDown label="Zuletzt verwendet:" id="drpZuletztVerwendet" onAction="onAction" getItemCount="getItemCount"
            getItemLabel="getItemLabel" getItemID="getItemID"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 1: Die Definition des **dropDown**-Elements zur Anzeige der zuletzt verwendeten Datensätze

in der Bibliothek **Microsoft Office x.0 Object Library** enthalten, auf die wir daher einen Verweis zum VBA-Projekt hinzufügen. Dazu wählen Sie im VBA-Editor den Menübefehl **Extras|Verweise** aus. Hier fügen Sie einen Verweis auf die genannte Bibliothek hinzu, sodass der Dialog anschließend wie in Bild 5 aussieht.

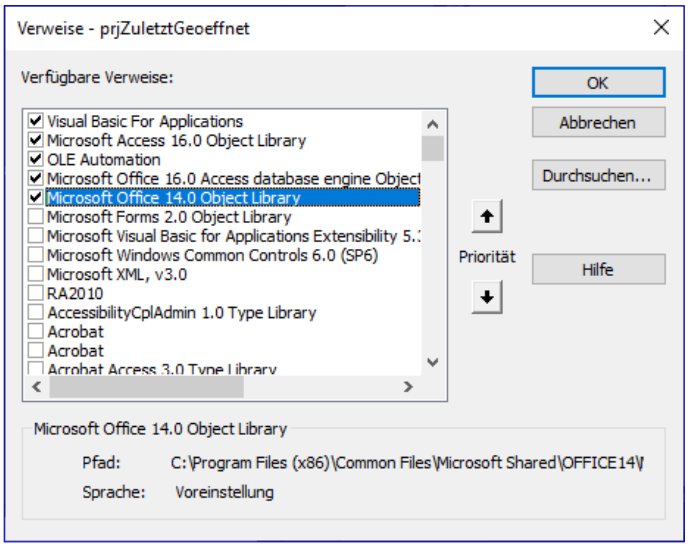

Bild 5: Hinzufügen eines Verweises auf die **Office**-Bibliothek

Nun können wir die Variable zum Referenzieren der Ribbondefinition zu einem neuen, leeren Modul namens **mdlRibbons** hinzufügen:

```
Public objRibbon_ZuletztVerwendete As IRibbonUI
```

Die beim Laden des Ribbons ausgeführte und in der Ribbondefinition für das Attribut **onLoad** des Elements **customUI** angegebene Prozedur definieren wir wie folgt:

```
Sub onLoad_ZuletztVerwendete(ribbon As IRibbonUI)
    Set objRibbon_ZuletztVerwendete = ribbon
End Sub
```

Die Prozedur liefert mit dem Parameter **ribbon** einen Verweis auf die Ribbondefinition, welche die Prozedur in **objRibbon_ZuletztVerwendete** speichert.

Hinzufügen der Callbackprozeduren für die Funktionalität des Ribbons

Nun wird es interessant, denn wir wollen das **dropDown**-Element mit den Daten aus der Abfrage **qryKundenZuletzt** füllen. Dazu haben wir im **dropDown**-Element drei Ereignisattribute mit entsprechenden VBA-Prozeduren definiert. In der ersten zählen wir die einzulesenden Elemente und tragen diese in ein Array ein, in der zweiten und dritten, die jeweils einmal für die ermittelte Anzahl aufgerufen werden, lesen wir die Daten aus dem Array aus und fügen die IDs und die Texte zum **dropDown**-Element hinzu. Als Erstes benötigen wir eine Variable, in der wir die anzuzeigenden Informationen speichern:

```
Private strZuletzt() As String
```

Danach folgt die Callbackprozedur, welche die Elemente zählt und im Array speichert. Diese füllt eine Recordset-Variable mit den Daten aus der Abfrage **qryKundenZuletzt** und durchläuft die einzelnen Datensätze in einer **Do While**-Schleife.

Dabei zählt sie die Elemente in der Variablen **i** und trägt gleichzeitig die Inhalte der Felder **KundeID** und **Firma** in das Array **strZuletzt** ein:

```
Sub getItemCount(control As IRibbonControl, ByRef count)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM µ
                        qryKundenZuletzt", dbOpenDynaset)
    Do While Not rst.EOF
        ReDim Preserve strZuletzt(1, i) As String
        strZuletzt(0, i) = rst!KundeID
        strZuletzt(1, i) = rst!Firma
        i = i + 1
        rst.MoveNext
    Loop
    count = i
End Sub
```

Zuletzt verwendete Datensätze per Listenfeld

Eine sehr praktische Funktion findet sich in den Office-Anwendungen und auch in vielen anderen Produkten. Wo auch immer Dateien verwendet werden, finden Sie beim Öffnen der jeweiligen Anwendung eine Liste der zuletzt verwendeten Dateien vor. Warum sollte man dies nicht auch in einer Access-Datenbank nutzen, um die zuletzt angezeigten Datensätze in Formularen zur Auswahl anzubieten? Wenn Sie zum Beispiel eine Bestellverwaltung nutzen, könnte es sehr sinnvoll sein, die zuletzt angezeigten oder bearbeiteten Kunden in einer Schnellauswahl zum erneuten Aufrufen vorzufinden. Wie das gelingt und welche Erweiterungen dazu notwendig sind, zeigt der vorliegende Beitrag.

Feature wie unter Office

Wenn Sie mit Anwendungen wie Word, Excel oder Access arbeiten, kennen Sie die Anzeige der zuletzt verwendeten Dateien, die beim Starten der Anwendung über die jeweilige Verknüpfung erscheint.

Hier wählen Sie immer aus den zuletzt verwendeten Dateien aus oder alternativ aus einer Liste von angehefteten Dateien, die unabhängig davon, wann sie zuletzt benutzt wurden, angezeigt werden sollen.

Vorbereitungen

Wir wollen zunächst nur den ersten Teil auch für die Datensätze einer Tabelle realisieren, in diesem Fall für die Kunden einer Bestellverwaltung.

Wir nutzen dazu wieder die Tabellen der Beispieldatenbank **Südsturm**, wobei für uns die Tabelle **tblKunden** interessant ist.

Als Basis dient das Formular **frmKundenZuletzt**, für dessen Eigenschaft **Datensatzquelle** wir die Tabelle **tblKunden** angeben.

Aus der Feldliste ziehen wir alle Felder der Tabelle in den Detailbereich des Entwurfs des Formulars (siehe Bild 1).

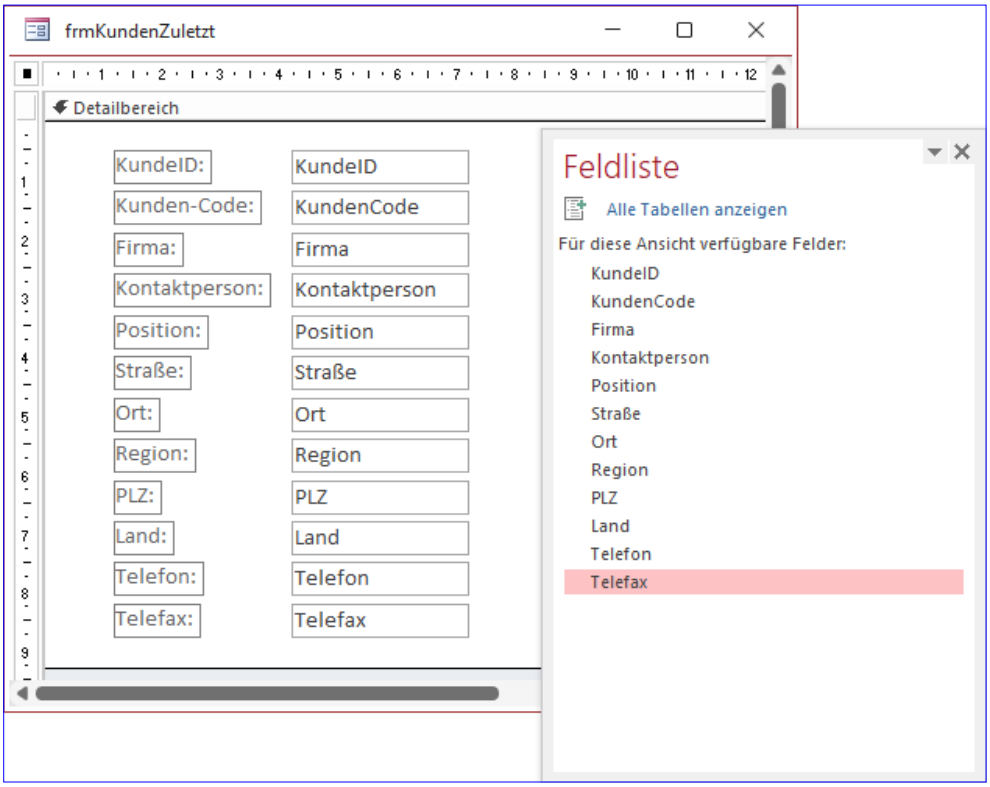

Bild 1: Dieses Formular wollen wir mit einer Liste der zuletzt verwendeten Kunden ausstatten.

Konzept für die Liste zuletzt verwendeter Datensätze

Bevor wir loslegen und dem Formular ein Listenfeld zur Anzeige der zuletzt verwendeten Datensätze hinzufügen, stellen wir ein paar grundlegende Überlegungen an.

Welche Datensätze speichern wir in der Liste der zuletzt verwendeten Datensätze?

Die erste Frage, die sich uns stellt, ist folgende: Zu welchen Datensätzen beziehungsweise wann soll ein Datensatz überhaupt gespeichert und in der Liste der zuletzt verwendeten Datensätze angezeigt werden?

Es gibt verschiedene Möglichkeiten:

- Jeder angezeigte Datensatz wird in der Liste gespeichert.
- Jeder bearbeitete Datensatz wird gespeichert.
- Jeder gezielt angezeigte Datensatz, beispielsweise durch das Anzeigen von einem Übersichtsformular aus, wird gespeichert.
- Nur Datensätze, die explizit gespeichert werden sollen – beispielsweise über einen Klick auf eine entsprechende Schaltfläche – werden in die Liste übernommen.

Wir wollen im ersten Entwurf den ersten Ansatz umsetzen und alle Datensätze in der Liste der zuletzt angezeigten Datensätze speichern, die überhaupt im Formular **frmKundenZuletzt** angezeigt wurden.

Wo sollen die Informationen über die zuletzt verwendeten Datensätze gespeichert werden?

Die zweite Frage lautet: Wo wollen wir überhaupt die Information speichern, welche Datensätze der Tabelle **tblKunden** zuletzt verwendet wurden?

Hier gibt es wiederum mindestens zwei Möglichkeiten:

- In einer eigenen Tabelle, die wir beispielsweise **tblKundenZuletzt** nennen. Hier speichern wir Informationen wie den Namen und/oder den Primärschlüsselwert des jeweiligen Datensatzes. Die Liste zur Anzeige der zuletzt verwendeten Datensätze kann ihre Daten direkt aus dieser Tabelle beziehen.
- Direkt in der Tabelle **tblKunden**. Wir könnten ein eigenes Feld mit dem Datentyp **Datum** beispielsweise namens **ZuletztAngezeigt** anlegen, das wir immer aktualisieren, wenn der Datensatz angezeigt wurde. Die Liste soll dann die X zuletzt geöffneten Datensätze anhand dieses Feldes ermitteln.

Hier wählen wir ebenfalls den ersten Ansatz. Dies ist praktikabler, weil wir dann nicht den Entwurf der Tabelle **tblKunden** anpassen müssen. Gegebenenfalls handelt es sich bei dieser Tabelle um eine per ODBC eingebundene SQL Server-Tabelle und es ist gar nicht möglich, diese Tabelle anzupassen. Außerdem spricht für die Speicherung in einer eigenen Tabelle, dass diese im Frontend gespeichert werden kann und somit die zuletzt verwendeten Datensätze für den jeweiligen Benutzer sichert.

Anlegen der Tabelle zum Speichern der zuletzt angezeigten Datensätze

Die Tabelle wollen wir **tblKundenZuletzt** nennen. Wir könnten auch den Namen **tblZuletztGeoeffnet** oder ähnlich verwenden, aber vielleicht wollen wir die gleiche Funktion auch noch für die Artikel hinzufügen – daher ist es sinnvoller, den Namen der betroffenen Elemente direkt im Tabellennamen anzugeben.

Welche Felder soll die Tabelle für die zuletzt angezeigten Datensätze enthalten?

Nun wird es interessanter. Wenn es nur um die Anzeige der zuletzt verwendeten Einträge der Tabelle **tblKunden** geht, könnte man neben dem Primärschlüsselfeld nur noch ein Feld mit der Bezeichnung des jeweiligen Elements, in diesem Falle der Kunden, anlegen. Allerdings wollen Sie den jeweiligen Kunden ja auch über das Listen-

feld öffnen, und dazu müssen wir genau wissen, um welchen Datensatz es sich handelt. Also benötigen wir auch noch den Primärschlüsselwert aus der Tabelle **tblKunden**.

Wenn wir aber den Primärschlüsselwert und auch den Namen des Kunden aus der Tabelle **tblKunden** in der Tabelle **tblKundenZuletzt** speichern, laufen wir Gefahr, inkonsistente Daten anzuzeigen. Es kann nämlich sein, dass wir im Formular **frmKundenLetzte** den Namen eines Kunden ändern, nachdem wir diesen angezeigt haben.

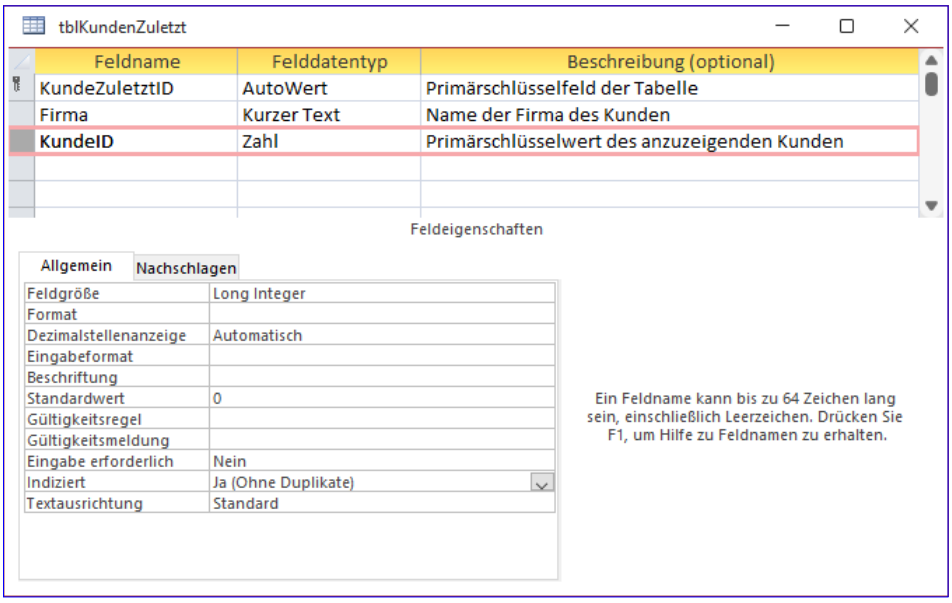

Bild 2: Entwurf der Tabelle **tblKundenZuletzt**

Dem können wir allerdings vorbeugen, indem wir im Formular **frmKundenLetzte** nicht nur eine Ereignisprozedur anlegen, die beim Anzeigen eines Kunden ausgelöst wird und dessen Daten in der Tabelle **tblKundenZuletzt** speichert, sondern auch noch eine, die beim Ändern eines Datensatzes ausgelöst wird und den aktuellen Kunden dann in der Tabelle **tblKundenZuletzt** überschreibt.

Und bei der Gelegenheit sollten wir auch im Hinterkopf behalten, dass der Benutzer ja auch einmal einen Kundendatensatz löschen kann. Dieser soll dann selbstverständlich auch aus der Tabelle **tblKundenZuletzt** entfernt werden.

Entwurf der Tabelle tblKundenZuletzt

Also gestalten wir die Tabelle **tblKundenZuletzt** wie in Bild 2. Neben dem Primärschlüsselfeld legen wir ein Feld an, das den gleichen Namen erhält wie das Feld, aus dem die in der Liste der zuletzt geöffneten Einträge anzuzeigenden Werte stammen. Für das Feld **KundeID**, welches den Primärschlüsselwert der anzuzeigenden Einträge aus der Tabelle **tblKunden** aufnehmen soll, legen wir einen eindeutigen Index fest. Dieser sorgt dafür, dass jeder Datensatz der Tabelle **tblKunden** nur einmal in der Tabelle **tblKundenZuletzt** referenziert werden kann. Später nutzen wir das Auslösen eines Fehlers beim Hinzufügen eines Datensatzes mit einem bereits vorhandenem Wert in diesem Feld für unsere Zwecke – mehr dazu weiter unten.

Benötigen wir kein Feld in tblKundenZuletzt, das die Reihenfolge angibt?

Wir könnten die Funktion mit noch mehr Features ausstatten und beispielsweise die Zeit anzeigen, wann der Datensatz zuletzt aufgerufen wurde. Fürs erste reicht es uns jedoch, einfach nur die zuletzt verwendeten Kunden so anzuzeigen, dass der zuletzt verwendete Kunde ganz oben erscheint. Benötigen wir dazu ein spezielles Feld? Nein, wir können dazu das Primärschlüsselfeld **KundeZuletztID** nutzen und einfach in absteigender Reihenfolge nach dem Inhalt dieses Feldes sortieren.

Prozedur erstellen, die den aktuellen Kunden zur Tabelle hinzufügt

Nachdem wir diese Tabelle erstellt haben, wollen wir dem Formular **frmKundeZuletzt** zuerst einmal eine Ereignisprozedur hinzufügen, die bei Anzeigen eines Kunden ausgelöst wird und welche die Daten für diesen Kunden in der

```
Private Sub Form_Current()
    Dim db As DAO.Database
    Dim strINSERT As String
    Dim strDELETE As String
    If Not IsNull(Me!KundeID) Then
        Set db = CurrentDb
        On Error Resume Next
        strINSERT = "INSERT INTO tblKundenZuletzt(Firma, KundeID) VALUES('" & Replace(Me!Firma, "'", "''") & "', " _
            & Me!KundeID & ")"
        db.Execute strINSERT, dbFailOnError
        Select Case Err.Number
            Case Is = 3022
                strDELETE = "DELETE FROM tblKundenZuletzt WHERE KundeID = " & Me!KundeID
                db.Execute strDELETE, dbFailOnError
                db.Execute strINSERT, dbFailOnError
            Case 0
            Case Else
                MsgBox "Fehler " & Err.Number & vbCrLf & vbCrLf & Err.Description
        End Select
    End If
End Sub
```

Listing 1: Prozedur zum Speichern der Daten für den aktuellen Datensatz in der Tabelle **tblKundenZuletzt**

Tabelle **tblKundenZuletzt** speichert. Erst danach fügen wir das Listenfeld mit den Daten der Tabelle **tblKunden-Zuletzt** hinzu.

Das passende Ereignis für unsere Zwecke heißt **Beim Anzeigen**. Es wird sowohl beim Anzeigen des ersten Datensatzes beim Öffnen des Formulars als auch beim Wechseln zu einem anderen Datensatz ausgelöst.

Diese Prozedur füllen wir mit dem Code aus Listing 1. Die Prozedur prüft zuerst, ob **KundeID** nicht den Wert **Null** hat, was der Fall ist, wenn der Datensatzzeiger auf einen neuen, leeren Datensatz verschoben wird. Diesen Fall müssten wir später berücksichtigen, wenn wir uns um das Speichern beim Ändern eines Datensatzes kümmern.

Danach deaktivieren wir die eingebaute Fehlerbehandlung mit **On Error Resume Next**. Das ist nötig, damit wir versuchen können, den Datensatz mit dem entsprechenden Wert im Feld **KundeID** zur Tabelle **tblKundenZuletzt** hinzuzufügen. Ist bereits ein Datensatz mit diesem Wert für das Feld **KundeID** vorhanden, wird nämlich der Fehler mit der Nummer **3022** ausgelöst. In diesem Fall wollen wir den vorhandenen Datensatz mit dem Wert aus **KundeID** aus der Tabelle **tblKundenZuletzt** löschen und diesen erneut anlegen. Warum belassen wir diesen dann nicht einfach in der Tabelle? Weil der Datensatz nun der zuletzt verwendeten Datensatz ist und da wir absteigend nach dem Inhalt des Feldes **KundeZuletztID** sortieren, müssen wir den Datensatz löschen und wieder anlegen, um dieses entsprechend zu aktualisieren.

In dem Fall, dass der Kunde noch nicht in der Liste enthalten ist, fügen wir diesen einfach mit der Anweisung aus **strINSERT** zur Tabelle **tblKundenZuletzt** hinzu. Für einen Beispieldatensatz sieht diese Anweisung etwa wie folgt aus:

```
INSERT INTO tblKundenZuletzt(Firma, KundeID) VALUES('Centro comercial Moctezuma', 13)
```

Falls tatsächlich der Fehler **3022** auftritt, den wir in einer **Select Case**-Bedingung über den Wert von **Err.Number** prüfen, stellen wir eine **DELETE**-Anweisung zusammen, um den betroffenen Datensatz zu löschen und diesen anschließend mit der Anweisung aus **strINSERT** erneut anzulegen. Die **DELETE**-Anweisung sieht für diese Fall wie folgt aus:

```
DELETE FROM tblKundenZuletzt WHERE KundeID = 13
```

Die **Select Case**-Bedingung enthält noch zwei weitere Zweige. Der mit dem Wert **0** ist für einen fehlerfreien Aufruf der **INSERT INTO**-Anweisung vorgesehen und erledigt nichts. Der **Else**-Zweig zeigt für alle anderen Fehler eine entsprechende Meldung an.

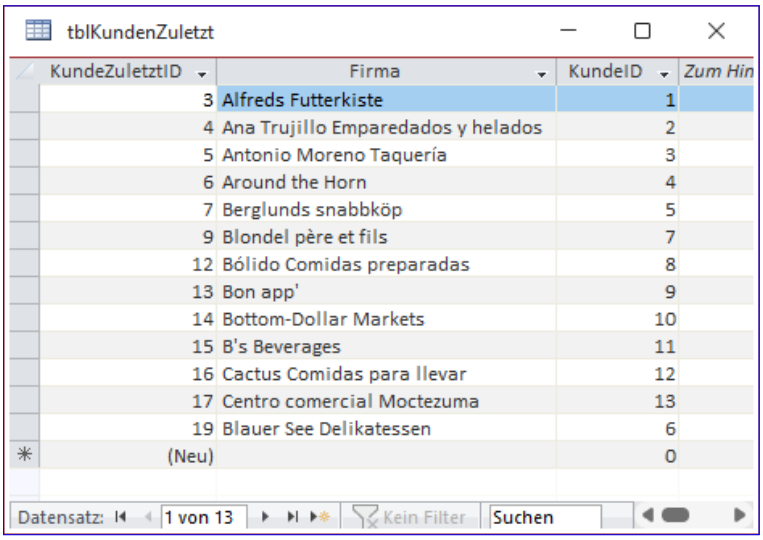

Bild 3: Einige zur Tabelle **tblKundenZuletzt** hinzugefügte Datensätze

Nachdem wir in die Formularansicht gewechselt sind und einige Datensätze durchlaufen haben – inklusive eines Sprungs zum Datensatz mit der Nummer 6 am Ende – sieht die Tabelle **tblKundenZuletzt** wie in Bild 3 aus.

Zuletzt verwendete Datensätze in Listenfeld anzeigen

Nun fügen wir dem Formular **frmKundenZuletzt** auf der linken Seite ein Listenfeld namens **lstKundenZuletzt** hinzu. Zuvor verschieben wir die Textfelder des Formulars weiter nach rechts, damit genug Platz für das Listenfeld vorhanden ist. Das Ergebnis sehen Sie in Bild 4.

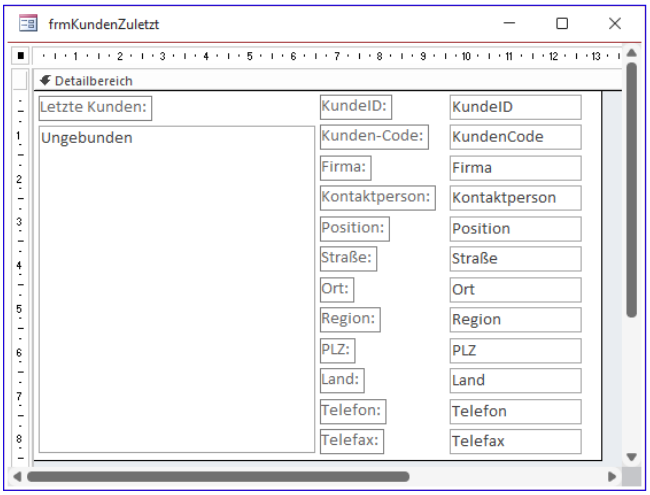

Bild 4: Entwurf des Formulars **frmKundenZuletzt** mit Listenfeld

Das Listenfeld füllen wir über die Eigenschaft **Datensatzherkunft** mit einer Abfrage, die auf der Tabelle **tblKundenZuletzt** basiert.

Warum benötigen wir dazu eine Abfrage? Weil wir die Datensätze absteigend nach dem Primärschlüsselfeld **KundeZuletztID** sortieren wollen. Die Abfrage sieht in der Entwurfsansicht wie in Bild 5 aus.

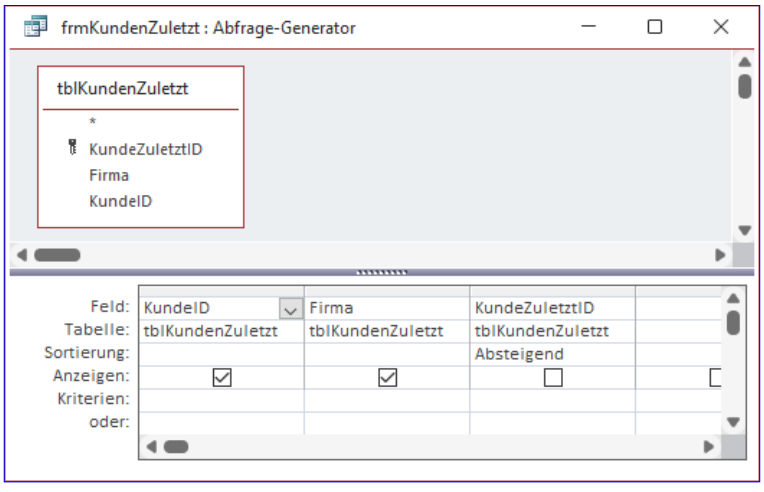

Bild 5: Abfrage mit den Daten für das Listenfeld **lstKundenZuletzt**

Sie enthält alle Felder der Tabelle **tblKundenZuletzt** und sortiert die Datensätze absteigend nach dem Feld **KundeZuletztID**. Das Feld **KundeZuletztID** benötigen wir allerdings nur zum Sortieren, aber nicht für die Anzeige im Listenfeld. Also blenden wir es durch Entfernen des Hakens im Feld **Anzeigen** aus. Außerdem wollen wir das Feld **KundeID** als erstes Feld zurückliefern und als zweites das Feld **Firma**. Warum, erläutern wir gleich im Anschluss.

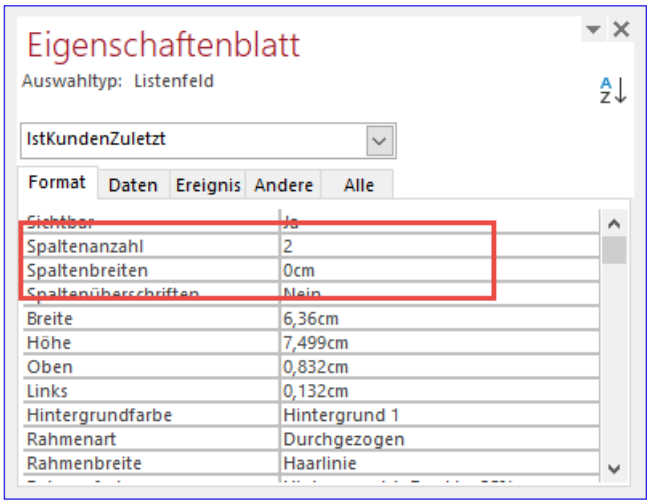

Bild 6: Einstellungen für die Spalten im Listenfeld

Definition der Anzeige im Listenfeld

Die Abfrage liefert die beiden Felder **KundeID** und **Firma** zurück. Wir möchten aber nur das Feld **Firma** anzeigen. Das Feld **KundeID** möchten wir als gebundenes Feld nutzen. Deshalb stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein. Dadurch wird die erste Spalte praktisch ausgeblendet und nur die zweite Spalte angezeigt. Das Ergebnis sehen Sie in Bild 7.

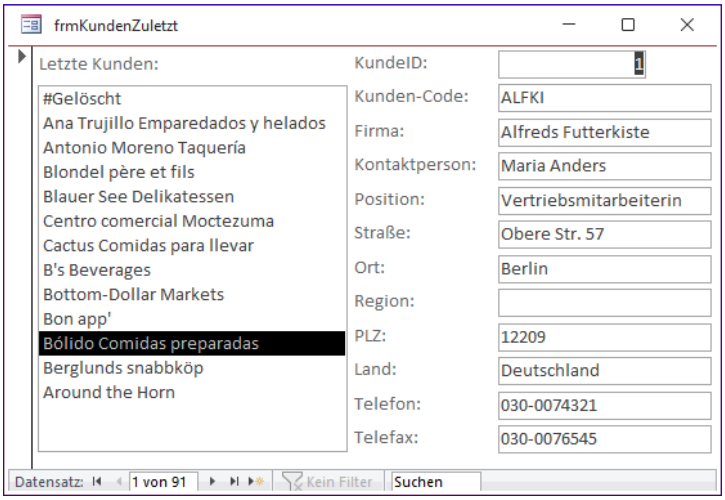

Bild 7: Die Anzeige der zuletzt geöffneten Datensätze

Hier sehen wir allerdings ein kleines Problem: Der erste Datensatz im Listenfeld erscheint gleich als **#Gelöscht**. Der Effekt tritt auf, weil dieser Datensatz derjenige ist, der gerade oben stand und auch als erster angezeigt wird. Diese Aktualisierung erledigt Access im Gegensatz zu den anderen, wie das Neuanlegen dieses Datensatzes, automatisch.

Also fügen wir der **Beim Anzeigen**-Prozedur noch eine Zeile hinzu, die das Listenfeld aktualisiert:

```
Private Sub Form_Current()
    ...
    If Not IsNull(Me!KundeID) Then
        ...
        Me!lstKundenZuletzt.Requery
    End If
End Sub
```

Damit können Sie nun durch die Datensätze navigieren und das Listenfeld nimmt immer den aktuellen Datensatz als ersten Datensatz in die Liste auf.

Den aktuellen Datensatz sofort anzeigen?

Hier stellt sich die Frage, ob der aktuelle Datensatz immer gleich beim Anzeigen im Formular auch an die erste Stelle in die Liste der zuletzt angezeigten Datensätze aufgenommen werden soll. Immerhin wird er ja gerade erst im Formular angezeigt, warum sollte man ihn dann direkt als ersten Datensatz in der Liste anzeigen?

Also prüfen wir, ob wir das auch so umsetzen können, dass der Datensatz immer erst beim Anzeigen des nächsten Datensatzes in die Liste aufgenommen werden kann. Das gelingt zumindest nicht so einfach, denn im Gegen-

Löschereignisse und -optionen im Zusammenspiel

Formulare haben gleich drei Ereignisse, die sich rund um das Thema Löschen drehen. Welche davon ausgelöst werden, hängt auch von der Einstellung einer Access-Option ab. Um sicherzugehen, dass Aktionen, die nach dem Löschen eines Datensatzes über das Formular ausgeführt werden sollen, tatsächlich stattfinden, müssen Sie einige Dinge beachten. Dieser Beitrag stellt die drei betroffenen Ereignisprozeduren vor, erläutert die Access-Option, die sich auf die Ausführung dieser Ereignisprozeduren auswirkt und zeigt, wie Sie das alles so zusammenbringen, dass die gewünschten Folgeaktionen zuverlässig ausgeführt werden.

Formularereignisse

Formulare bieten so einige Ereignisse, die durch verschiedene Aktionen des Benutzers ausgelöst werden. Dazu gehören das Öffnen und Laden des Formulars sowie das Schließen und Entladen, außerdem gibt es Ereignisse, die beim Anzeigen, vor der Aktualisierung oder nach der Aktualisierung eines Datensatzes feuern.

Löschen-Ereignisse

In manchen Fällen kann es auch wichtig sein, die Ereignisse beim Löschen eines Datensatzes abzufangen. Und derer gibt es gleich drei.

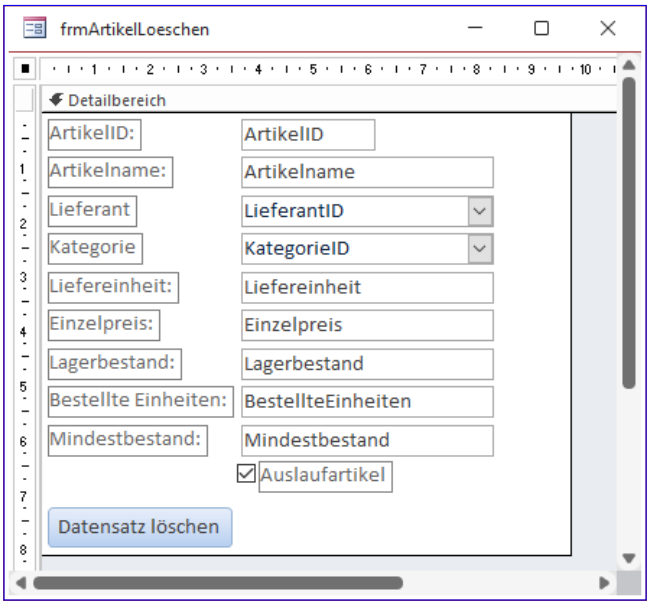

Bild 1: Beispielformular mit Löschen-Schaltfläche

Diese heißen folgendermaßen:

- **Beim Löschen**
- **Vor Löschbestätigung**
- **Nach Löschbestätigung**

Beispielformular erstellen

Um mit diesen Ereignissen experimentieren zu können, haben wir ein Formular namens **frmArtikelLoeschen** erstellt, das die Tabelle **tblArtikel** als Datensatzquelle verwendet und der wir eine Schaltfläche namens **cmdDatensatzLoeschen** hinzugefügt haben (siehe Bild 1).

Die Schaltfläche löst die folgende Ereignisprozedur aus:

```
Private Sub cmdDatensatzLoeschen_Click()
    RunCommand acCmdDeleteRecord
End Sub
```

Der Befehl **RunCommand acCmdDeleteRecord** entspricht dem Löschen eines Datensatzes durch Markieren des Datensatzes über den Datensatzmarkierer auf der linken Seite und anschließendes Betätigen der **Löschen**- oder der **Entf**-Taste. Alternativ können Sie auch die Tastenkombination **Strg + -** nutzen. Später werden wir sehen, dass hier noch eine Fehlerbehandlung notwendig ist.

Löschen-Ereignisse implementieren

Als Nächstes wollen wir die drei Ereignisprozeduren für die Ereignisse **Beim Löschen**, **Vor Löschbestätigung** und **Nach Löschbestätigung** implementieren.

Das erledigen wir wie üblich über das Eigenschaftenblatt des Formulars im Bereich **Ereignis**. Hier wählen Sie für die drei Ereignisse jeweils den Wert **[Ereignisprozedur]** aus und klicken auf die Schaltfläche mit den drei Punkten (**...**) – siehe Bild 2.

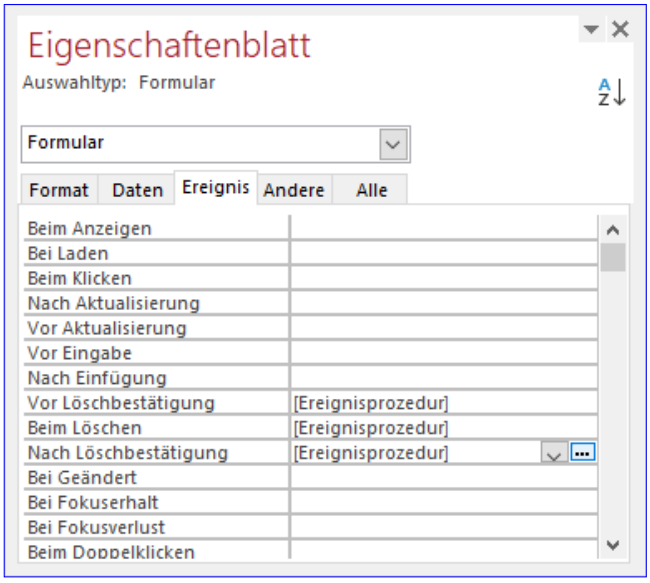

Bild 2: Implementieren der Löschen-Ereignisse

Damit legen Sie im Klassenmodul **Form_frmArtikel-Loeschen** drei Ereignisprozeduren an. Um diese analysieren zu können, fügen wir nun jeweils einen Haltepunkt hinzu und/oder eine **Debug.Print**-Anweisung, die den Namen der jeweiligen Prozedur im Direktbereich ausgibt – je nachdem, was für Sie praktischer erscheint (siehe Bild 3).

Reihenfolge der Ereignisprozeduren

Danach wechseln wir zur Formularansicht des Formulars **frmArtikelLoeschen** und löschen einen der Datensätze. Wie wir nun sehen, wird als Erstes die Ereignisprozedur **Form_Delete** (**Beim Löschen**) ausgelöst.

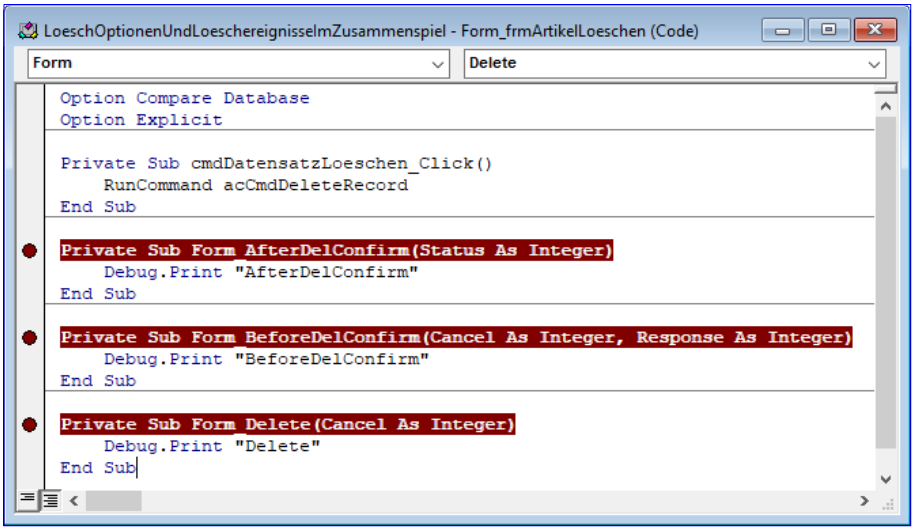

Bild 3: Die Löschen-Ereignisse im VBA-Editor

Zu diesem Zeitpunkt ist der Datensatz bereits aus dem Formular verschwunden. Dann wird das Ereignis **Form_BeforeDelConfirm** (**Vor Löschbestätigung**) ausgelöst. Danach gibt es einen Unterbrechung, in der die Meldung aus Bild 4 erscheint.

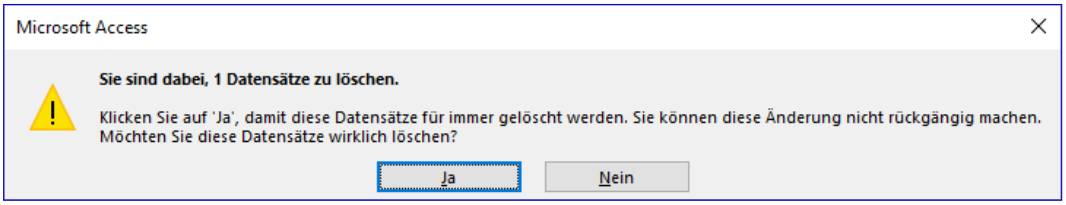

Bild 4: Meldung nach dem Ereignis **Vor Löschbestätigung**

Unabhängig davon, ob Sie die Schaltfläche **Ja** oder **Nein** betätigen, wird im Anschluss noch die Prozedur **Form_AfterDelConfirm** (**Nach Löschbestätigung**) aufgerufen. Allerdings wird im Falle von **Ja** der

aktuell angezeigte Datensatz gelöscht und im Falle von **Nein** wird dieser beibehalten und auch im Formular wieder angezeigt. Und auch beim Aufruf des Ereignisses **Nach Löschbestätigung** gibt es einen Unterschied, wie wir gleich sehen werden.

Parameter der Ereignisprozeduren

Hier wird es interessant, denn die Parameter von Ereignisprozeduren liefern entweder wichtige Informationen oder Sie können diese nutzen, um bestimmte Informationen zu übergeben.

Beginnen wir mit der Prozedur **Form_Delete**:

```
Private Sub Form_Delete(Cancel As Integer)
    Debug.Print "Delete", Cancel
End Sub
```

Diese enthält einen Parameter namens **Cancel**, der standardmäßig den Wert **0** enthält, was **False** entspricht. Das bedeutet: Standardmäßig hat **Cancel** den Wert **False**, was dazu führt, dass der Löschvorgang nicht abgebrochen wird.

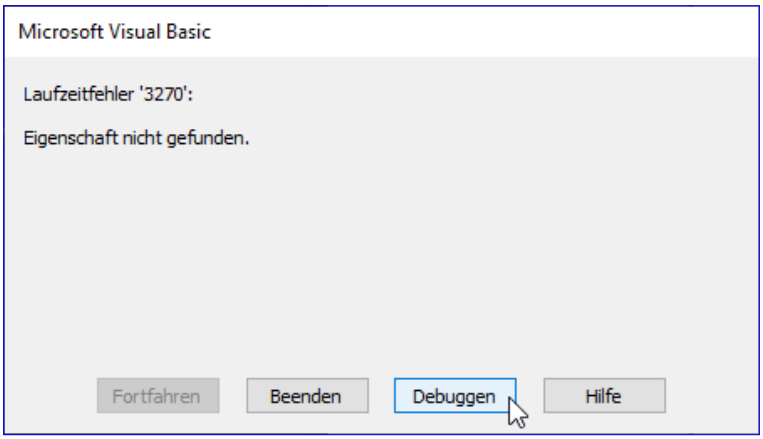

Bild 5: Fehler beim Abbruch von **RunCommand acCmdDeleteRecord**

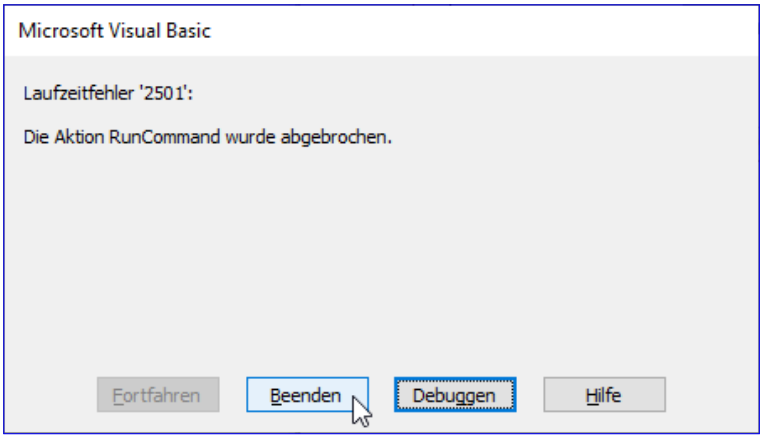

Bild 6: Fehler beim Abbruch von **RunCommand acCmdDeleteRecord** über die Standardrückfrage.

Individuelle Rückfrage zum Löschvorgang

Um das auszuprobieren, erweitern wir die Prozedur wie folgt:

```
Private Sub Form_Delete(Cancel As Integer)
    Debug.Print "Delete", Cancel
    If MsgBox("Löschen abbrechen?", vbYesNo) = vbYes Then
        Cancel = True
    End If
End Sub
```

Dies führt nun dazu, dass beim Löschen eine Meldung mit dem Text **Löschen abbrechen?** angezeigt wird. Klicken Sie hier auf **Ja**, stellt die Prozedur den Parameter **Cancel** auf den Wert **True** ein. Das funktioniert ohne Probleme, wenn Sie den Datensatz mit der Tastenkombination **Strg + -** löschen oder durch Anklicken des Datensatzmarkierers und anschließendes Betätigen der **Löschen**- oder der **Entf**-Taste.

Wenn Sie allerdings die Schaltfläche **cmdDatensatz-Loeschen** verwendet haben, löst dies einen Fehler aus, dessen Meldung wir uns in diesem Zusammenhang nicht erklären können (siehe Bild 5).

Bei der Gelegenheit zeigen wir auch direkt die Fehlermeldung, die erscheint, wenn Sie das Löschen mit **RunCommand acCmdDeleteRecord** initiieren und dann in der standardmäßigen Rückfrage den Löschvorgang abbrechen (siehe Bild 6). Das ist noch nachvollziehbar und dieser

E-Mails versenden mit CDO

Für das Versenden von E-Mails von einer Access-Anwendung aus gibt es verschiedene Möglichkeiten. Die naheliegendste ist der Versand unter Verwendung von Outlook, da dieses üblicherweise auf Rechnern mit Microsoft Access installiert ist. Es gibt jedoch auch Fälle, bei denen kein Office-Paket vorliegt und daher eine alternative Lösung gefragt ist. Früher gab es die Bibliothek vbSendmail, die auch heute noch eingeschränkt funktioniert. Eingeschränkt deshalb, weil beispielsweise SSL nicht unterstützt wird. Also haben wir nach einer Alternative gesucht, die auch moderne, sichere Versender unterstützt und sind dabei auf eine eher betagte Lösung gestoßen: die Bibliothek CDO, die auf jedem Windows-System installiert ist.

CDO-Bibliothek einbinden

Um die Elemente der CDO-Bibliothek zu nutzen, haben Sie wie üblich zwei Möglichkeiten – Late Binding und Early Binding. Da wir immer gern mit IntelliSense arbeiten, nutzen wir auch in diesem Fall wieder Early Binding. Das zieht nach sich, dass wir einen Verweis auf die Bibliothek zum VBA-Projekt der Zielanwendung hinzufügen müssen.

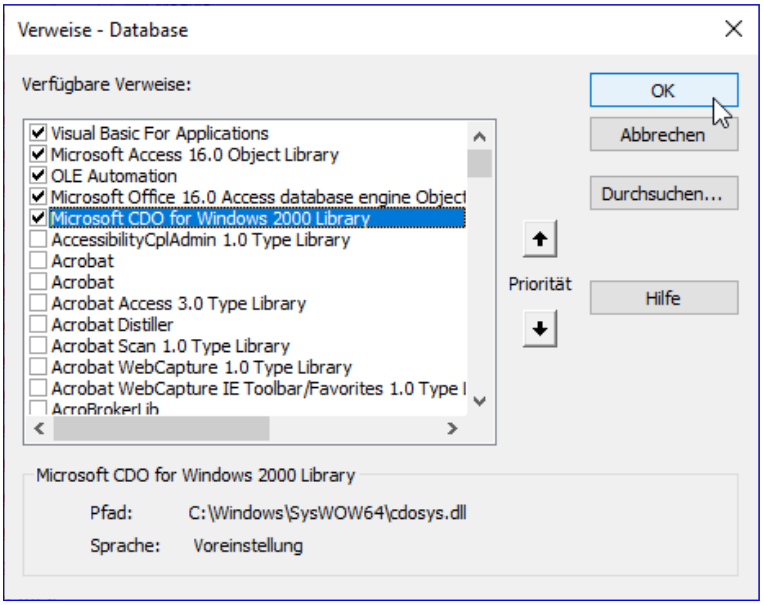

Bild 1: Verweis auf die CDO-Bibliothek

Dazu öffnen Sie den VBA-Editor mit der Tastenkombination **Alt + F11** und betätigen dann den Menübefehl **Extras|Verweise**. Im nun erscheinenden Verweise-Dialog für das aktuelle VBA-Projekt suchen Sie nach einem Eintrag namens **Microsoft CDO for Windows 2000 Library** und markieren diesen (siehe Bild 1). Dem Namen können Sie bereits entnehmen, warum wir die Bibliothek in der Einführung als »betagt« bezeichnet haben.

Nach dem Hinzufügen der Bibliothek schauen wir zuerst einmal in den Objektkatalog und wählen dort ganz oben den Eintrag **CDO** aus.

Hier finden wir schnell eine interessante Klasse namens **Message**, die alle notwendigen Eigenschaften und Methoden zu bieten scheint – inklusive **To**, **From**, **Subject**, **TextBody** und **Send** (siehe Bild 2).

Allerdings vermissen wir hier Eigenschaften, mit denen wir festlegen können, über welchen SMTP-Server wir die E-Mails versenden und wie wir uns dort authentifizieren. Die Bibliothek bietet neben einigen Interface-Klassen nur noch zwei weitere Klassen – nämlich **Configuration** und **DropDirectory**.

Darüber hinaus bietet die Klasse **Message** auch noch eine Eigenschaft namens **Configuration**, sodass wir wohl die notwendigen Eigenschaften über die **Configuration**-Klasse vornehmen werden.

Schauen wir allerdings im Objektkatalog in die Elemente der Klasse **Configuration**, finden wir dort nicht viel, was uns direkt weiterbringt – hier sehen wir nämlich nur die drei Elemente **Fields**, **GetInterface** und **Load**.

Auf der Suche nach dem Schlüsselwort **SMTP** landen wir jedoch schnell bei der Klasse **CdoConfiguration**, die einige Elemente bereitstellt, die hilfreich aussehen. Diese sehen Sie in Bild 3.

Eine weitere Recherche im Internet lieferte den Hinweis, dass wir Eigenschaften wie etwa den SMTP-Server über die **Fields**-Auflistung des **Configuration**-Objekts festlegen und dieses dann über die Eigenschaft **Configuration** dem **Message**-Objekt zuweisen.

Grundlegende Anwendung

Wie sich zeigen wird, benötigen wir für verschiedene Provider unterschiedliche Konfigurationen. Wir schauen uns erst einmal die einfachste Variante an, bei der wir von einem E-Mail-Provider mit einem einfachen SMTP-Server ausgehen, der keine zusätzlichen Funktionen wie SSL oder dergleichen bietet.

Um die zu verwendenden Parameter herauszufinden, starten wir erst einmal ohne Parameter mit der folgenden Prozedur.

Hier deklarieren wir zuerst einmal ein **Message**- und ein **Configuration**-Objekt:

```
Public Sub MailSenden()
    Dim objMessage As CDO.Message
    Dim objConfiguration As CDO.Configuration
```

Dann erstellen wir das **Configuration**-Objekt, mit dem wir erst einmal nichts konfigurieren, sowie ein **Message**-Objekt:

```
    Set objConfiguration = New CDO.Configuration
    Set objMessage = New CDO.Message
```

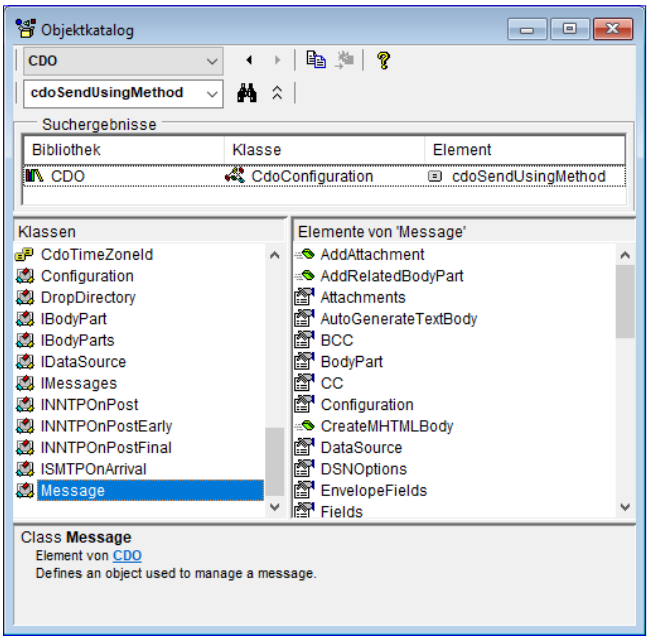

Bild 2: Eigenschaften und Methoden der **Message**-Klasse

Danach weisen wir dem **Message**-Objekt das leere **Configuration**-Objekt zu und legen einige andere, offensicht-

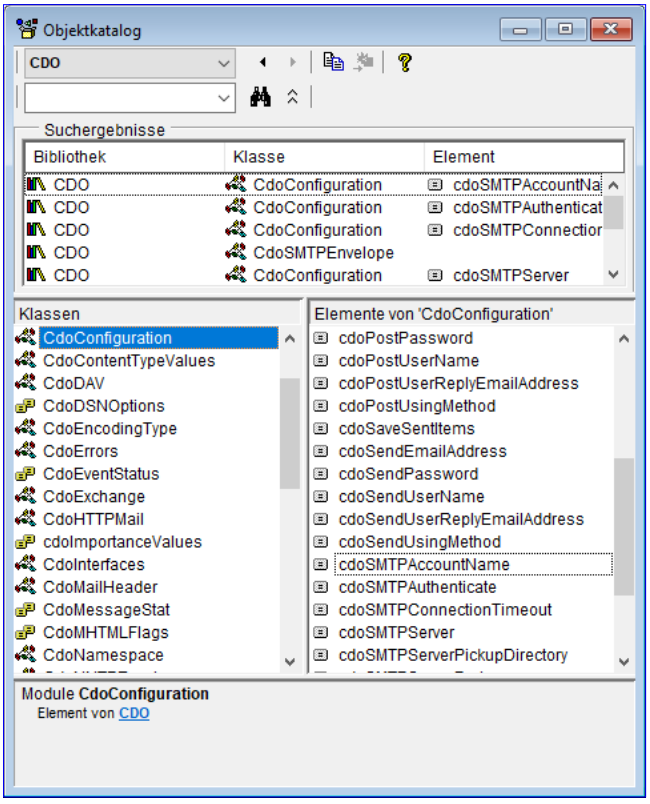

Bild 3: Konstanten für die **Fields**-Eigenschaft

lich benötigte Informationen fest – nämlich den Adressat, den Absender, den Betreff und den Inhalt der E-Mail:

```
    With objMessage
        Set .Configuration = objConfiguration
        .To = "andre@minhorst.com"
        .From = "info@amvshop.de"
        .Subject = "Beispielbetreff"
        .TextBody = "Dies ist ein Beispieltext."
```

Danach rufen wir bei deaktivierter eingebauter Fehlerbehandlung die **Send**-Methode auf und lassen uns im Falle eines Fehlers die Fehlernummer und den Fehlertext ausgeben:

```
        On Error Resume Next
        .Send
        If Not Err.Number = 0 Then
            Debug.Print Err.Number, Err.Description
        End If
    End With
    Set objConfiguration = Nothing
    Set objMessage = Nothing
End Sub
```

Erster Versuch des Mailversands

Rufen wir die Prozedur so auf, erhalten wir den folgenden Fehler:

```
-2147220960
Der "SendUsing"-Konfigurationswert ist ungültig.
```

Mit **SendUsing** landen wir im Objektkatalog schnell bei der Klasse **CdoSendUsing** mit der Eigenschaften **SendUsingPort**.

Senden mit Port

Also fügen wir dem **Configuration**-Objekt ein **Fields**-Element mit dieser Eigenschaft hinzu:

```
With objConfiguration
    .Fields(cdoSendUsingMethod).Value = cdoSendUsingPort
    .Fields.Update
End With
```

Dies liefert den folgenden Fehler:

```
-2147220982
Der erforderliche Name des SMTP-Servers wurde in der Konfigurationsquelle nicht gefunden.
```

Also geben wir auch noch einen SMTP-Server an:

```
...
.Fields(cdoSMTPServer).Value = "smtp.minhorst.com"
...
```

Damit versenden wir tatsächlich schon die erste E-Mail mit einem einfachen SMTP-Server ohne besondere Absicherung. Das funktionierte in unserem Fall aber auch nur, weil die Absender- und die Empfängeradresse beide mit dem angegebenen SMTP-Server arbeiten.

E-Mail-Server mit Verschlüsselung nutzen

Je nachdem, welchen Server Sie verwenden, können die notwendigen Angaben variieren. In den folgenden Abschnitten schauen wir uns die notwendigen Einstellungen für das **Configuration**-Objekt an.

Dazu haben wir eine neue, parametrisierte Version der obigen Prozedur namens **SendMail** geschaffen, mit der Sie alle notwendigen Einstellungen inklusive der Informationen für die E-Mails übergeben können (siehe Listing 1).

Die Prozedur erwartet die folgenden Parameter:

- **strServername**: Name des Servers, zum Beispiel **smtp.mailserver.de**
- **strUsername**: Benutzername für das Mailkonto, in der Regel die E-Mail-Adresse

Serienmails versenden mit CDO

Zum Verwenden von Serienmails nutzt man meist Outlook und schreibt eine Mail an sich selbst, während man die Empfänger dann möglichst dem Feld BCC hinzufügt. Auf diese Weise bleibt der Datenschutz gewahrt, denn Sie wollen ja nicht jedem Empfänger die E-Mail-Adressen aller anderen Empfänger der Serienmail mitteilen. Der Nachteil ist, dass Sie so noch nicht einmal einfache Individualisierungen realisieren können wie etwa eine persönliche Anrede. Wenn Sie das erledigen wollen, versenden Sie per Automation aus einer Datenbank Nachrichten über Outlook. Falls das nicht in Frage kommt, weil beispielsweise Outlook nicht auf dem Rechner installiert ist, können Sie noch eine Alternative nutzen, nämlich die Bibliothek CDO. Im Beitrag »E-Mails versenden mit CDO« haben wir bereits die grundlegenden Techniken zum E-Mail-Versand mit dieser Bibliothek vorgestellt. Nun gehen wir einen Schritt weiter und zeigen, wie das auch noch für Serienmails gelingt.

Vorbereitung für den Einsatz der CDO-Bibliothek

Damit Sie die CDO-Bibliothek mit den Objekten und Methoden zum Versenden von E-Mails nutzen können, benötigen Sie einen Verweis auf diese Bibliothek. Diesen fügen Sie über den **Verweise**-Dialog hinzu, den Sie vom VBA-Editor aus öffnen. Hier wählen Sie den Menüeintrag **Extras|Verweise** und wählen dann im erscheinenden Dialog **Verweise** den Eintrag **Microsoft CDO for Windows 2000 Library** aus (siehe Bild 1).

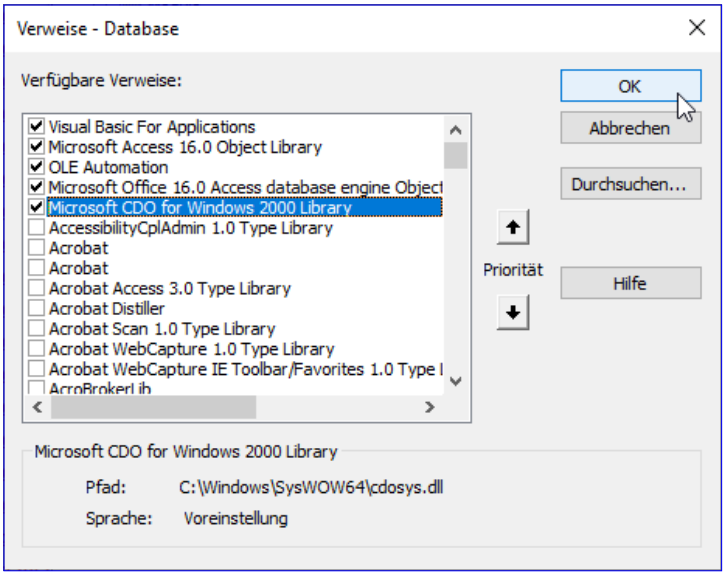

Bild 1: Verweis auf die CDO-Bibliothek

Vorbereiten der Datenbank

Um die hier vorgestellten Techniken nutzen zu können, benötigen Sie zumindest eine Tabelle mit den Daten der E-Mail-Empfänger. Diese sollte im besten Fall die Felder **Anrede**, **Vorname**, **Nachname** und **E-Mail-Adresse** enthalten.

Wir haben eine solche Tabelle wie in Bild 2 definiert. Das dortige Feld **AnredeID** ist ein Fremdschlüsselfeld, mit dem Sie die Datensätze der Tabelle **tblAnreden** auswählen können. Diese enthält neben dem Primärschlüsselfeld **AnredeID** noch das Feld **Anrede** mit der

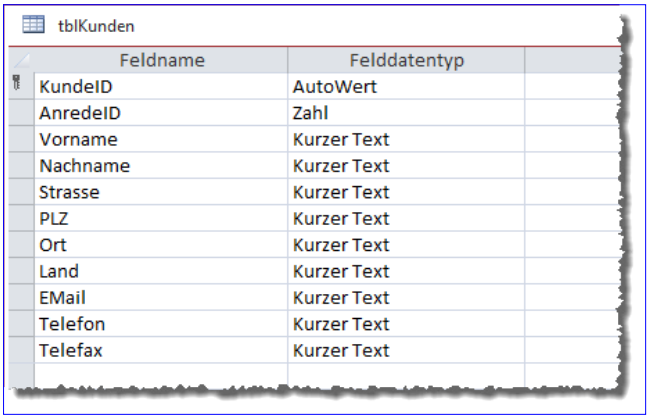

Bild 2: Beispieltabelle für Kundendaten im Entwurf

Bezeichnung der Anrede sowie ein weiteres Feld namens Mailanrede, das wir mit einer Anrede speziell für die E-Mails gefüllt haben.

Diese Tabelle sieht mit den Daten für die Anreden **Herr** und **Frau** wie in Bild 3 aus.

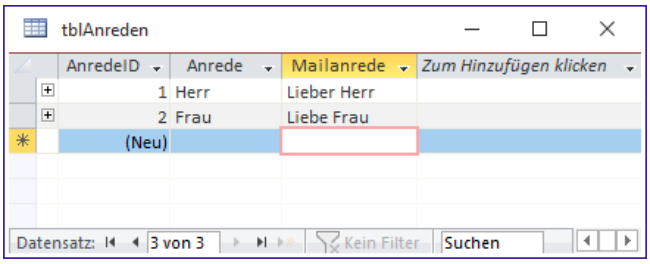

Bild 3: Tabelle mit den Anreden

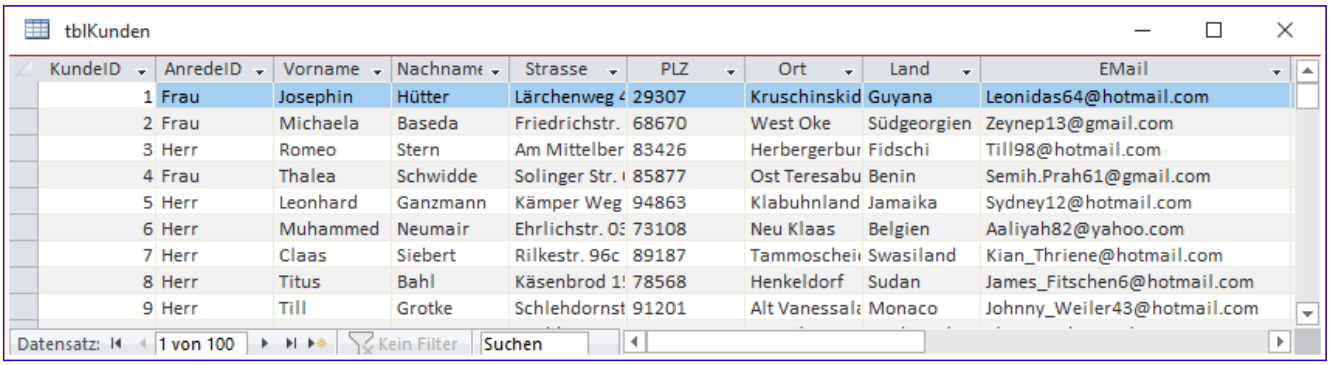

Bild 4: Tabelle mit den aufgefüllten Datensätzen

Die Tabelle **tblKunden** haben wir mit den Techniken aus dem Beitrag **Beispieldaten generieren mit .NET und Bogus** (**www.access-im-unternehmen.de/1359**) gefüllt. Die Tabelle sieht anschließend wie in Bild 4 aus. Wir werden später nur die ersten zwei, drei Datensätze, um testweise E-Mails zu versenden. Diesen fügen wir daher tatsächlich vorhandene E-Mail-Adressen hinzu, bei denen wir auch prüfen können, ob diese angekommen sind.

Speichern der SMTP-Konfiguration

Im Beitrag **E-Mails versenden mit CDO** (**www.access-im-unternehmen.de/1363**) haben wir die Daten für den Zugriff auf den SMTP-Server, der für den Mailversand verwendet werden soll, direkt in den Code geschrieben.

Das wollen wir in diesem Beitrag direkt ein wenig professioneller gestalten und die Daten in einer Tabelle unterbringen, deren Inhalte wir dann über ein Formular anpassen können.

Die Tabelle finden Sie in der Entwurfsansicht in Bild 5. Sie nimmt neben dem Primärschlüsselfeld zunächst ein Feld zum Speichern der Bezeichnung eines Satzes von SMTP-Server-Daten auf. Danach folgen die

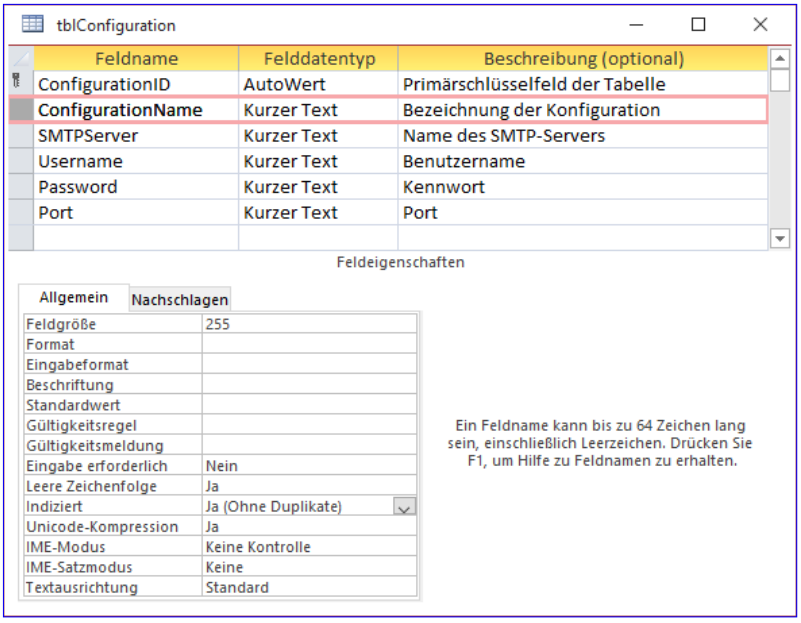

Bild 5: Tabelle zum Speichern der SMTP-Konfiguration

Felder für die Adresse des SMTP-Servers, für den Benutzernamen und das Kennwort, unter dem die Anmeldung erfolgen soll, sowie gegebenenfalls der Port.

Auslesen der Konfiguration und Erstellen eines Configuration-Objekts

Zusätzlich programmieren wir eine Funktion, der wir nur noch den Primärschlüsselwert der zu liefernden Konfiguration übergeben. Diese soll dann ein fertiges **Configuration**-Objekt auf Basis der Daten aus dem entsprechenden Datensatz der Tabelle zurückliefern.

Diese Funktion sieht wie in Listing 1 aus. Sie erwartet den Parameter **lngConfigurationID** und liefert ein Objekt des Typs **Configuration** zurück. Sie erzeugt ein **Database**-Objekt auf Basis der aktuellen Datenbank und ein Recordset mit dem Datensatz der Tabelle **tblConfigurations**, dessen Wert im Feld **ConfigurationID** dem mit **lngConfigurationID** übergebenen Parameter entspricht.

Aus diesem Recordset liest die Funktion die einzelnen Feldwerte aus und schreibt diese in die Variablen **strSMTPServer**, **strPort**, **strUsername** und **strPassword**.

```
Public Function GetConfiguration(lngConfigurationID As Long) As CDO.Configuration
    Dim objConfiguration As CDO.Configuration
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strSMTPServer As String
    Dim strPort As String
    Dim strUsername As String
    Dim strPassword As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblConfigurations WHERE ConfigurationID = " & lngConfigurationID, _
        dbOpenDynaset)
    strSMTPServer = rst!SMTPServer
    strPort = Nz(rst!Port, "")
    strUsername = rst!UserName
    strPassword = rst!Password
    Set objConfiguration = New CDO.Configuration
    With objConfiguration
        .Fields(cdoSendUsingMethod).Value = cdoSendUsingPort
        .Fields(cdoSMTPServer).Value = strSMTPServer
        Debug.Print .Fields(cdoSMTPServerPort)
        If Not Len(strPort) = 0 Then
            .Fields(cdoSMTPServerPort).Value = strPort
        End If
        .Fields(cdoSMTPUseSSL).Value = "true"
        .Fields(cdoSMTPAuthenticate).Value = cdoBasic
        .Fields(cdoSendUserName).Value = strUsername
        .Fields(cdoSendPassword).Value = strPassword
        .Fields.Update
    End With
    Set GetConfiguration = objConfiguration
End Function
```

Listing 1: Funktion zum Ermitteln eines **Configuration**-Objekts

Dann erstellt sie ein neues Objekt des Typs **CDO.Configuration** und weist den Eigenschaften dieses Objekts die Werte aus den Textfeldern zu – neben einigen Werten für Eigenschaften, die immer gleich sind wie etwa für **cboSendUsingMethode**, **cdoSMTPUserSSL** oder **cdoSMTPAuthenticate**. Nachdem alle relevanten Eigenschaften gefüllt sind, gibt die Funktion das mit **objConfiguration** referenzierte Objekt als Ergebnis zurück.

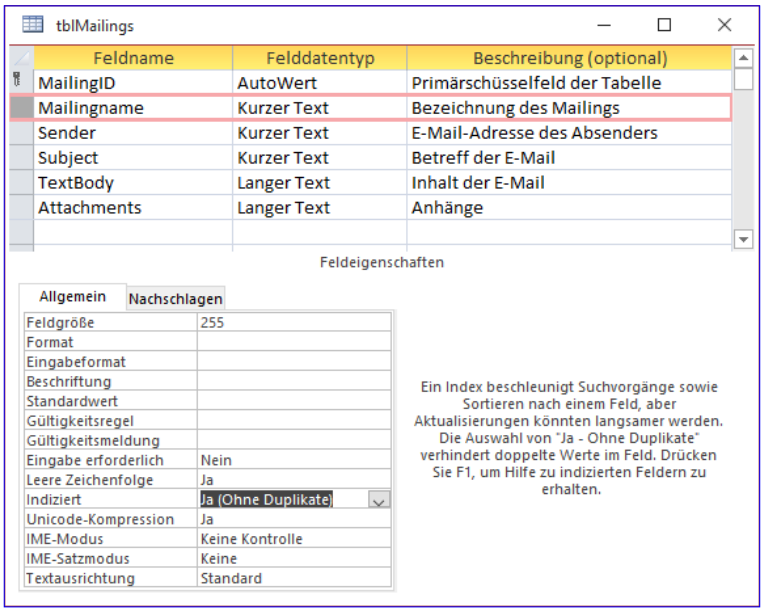

Bild 6: Tabelle zum Speichern der Daten eines Mailings

Serien-E-Mails verschicken

In der Lösung aus dem Beitrag **E-Mails senden mit CDO** (**www.access-im-unternehmen.de/1363**) haben wir das Zusammenstellen des **Configuration**-Objekts und des **Message**-Objekts, das zum Versenden einer E-Mail dient, in einer Prozedur erstellt. Das ist beim Versand einer Serienmail nicht sinnvoll – wenn immer der gleiche SMTP-Server mit den gleichen Anmeldedaten zum Einsatz kommt, brauchen wir nicht für jede E-Mail erneut ein **Configuration**-Objekt zu erstellen. Wenn wir also gleich eine Prozedur programmieren, mit der wir die Empfänger der E-Mail durchlaufen, wird dies berücksichtigen werden.

Für das Versenden einer Serienmail benötigen wir noch einige Daten, die wir am Besten auch direkt in einer eigenen Tabelle speichern. Dabei handelt es sich um die Daten, die für jede Mail eines Mailings gleich sind, also:

- E-Mail-Adresse des Absenders
- Betreff der E-Mail
- Inhalt der E-Mail, gegebenenfalls mit Platzhaltern
- Anhänge für die E-Mail, gegebenenfalls mit Platzhaltern

Diese Daten speichern wir in der Tabelle **tblMailings** aus Bild 6. Neben den genannten Daten nimmt diese Tabelle nur noch ein Primärschlüsselfeld auf sowie ein Feld für die Bezeichnung des Mailings. Für das Feld **Mailingname**

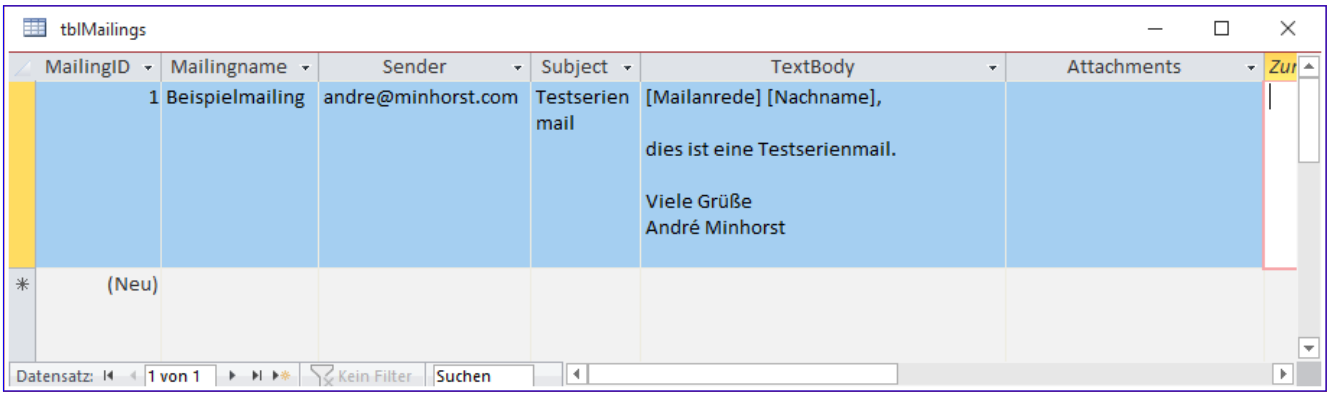

Bild 7: Tabelle zum Speichern der Daten eines Mailings mit Beispieldaten

Benutzeroberfläche für CDO-Serienmails

Im Beitrag »Serienmails versenden mit CDO« haben wir einige Prozeduren und Funktionen vorgestellt, mit denen Sie Serien-E-Mails über die CDO-Bibliothek von Windows versenden können. Das macht natürlich nur halb soviel Spaß, wenn nur die nackten Routinen vorliegen. Also zeigen wir im vorliegenden Beitrag auch noch, wie Sie eine praktische Benutzeroberfläche zum Verwalten der für den Versand einer Serienmail benötigten Daten programmieren.

Datenmodell der Anwendung

Die Anwendung zum Versenden von Serienmails enthält die Tabellen aus dem **Beziehungen**-Fenster aus Bild 1. Hier finden Sie die beiden Tabellen **tblKunden** sowie **tblAnreden**, welche die Informationen zu den Empfängern liefern. Die Tabelle **tblConfigurations** enthält die Daten für die Konfiguration des Mailservers und die Tabelle **tblMailings** die Daten zum Mailing selbst, also eine Bezeichnung, die Absenderadresse, den Betreff, den Inhalt und die Anlagen. Der Tabelle **tblMailings** haben wir ein Fremdschlüsselfeld namens **ConfigurationID** hinzugefügt, mit dem wir die für das jeweilige Mailing verwendete Konfiguration speichern können. Außerdem enthält die Tabelle **tblMailings** zwei Felder, mit denen die Adressaten des Mailings definiert werden. **SQLSource** nimmt den SQL-Ausdruck auf, der die Empfängeradressen liefert und **Mailfield** den Namen des Feldes aus diesem Ausdruck, der festlegt, welches Feld die Mailadresse enthält.

Zu erstellende Formulare

Wir wollen der Lösung einige Formulare hinzufügen, mit der Sie diese komfortabel steuern können. Dazu gehören die folgenden:

- Formular zum Verwalten der Konfigurationen
- Formular zum Verwalten der Mailings
- Formular zum Auswählen der Empfänger des Mailings

Formular zum Verwalten der Konfigurationen

Wir beginnen mit dem Formular **frmConfigurations**. Das Formular verwendet die Tabelle **tblConfigurations** als Datensatzquelle. Ziehen Sie alle Felder aus der Feldliste in das Formular und ordnen Sie diese so an wie in Bild 2. Dann fügen Sie noch einige weitere Steuerelemente hinzu:

- Kombinationsfeld **cboSchnellauswahl** (in den Formularkopf)

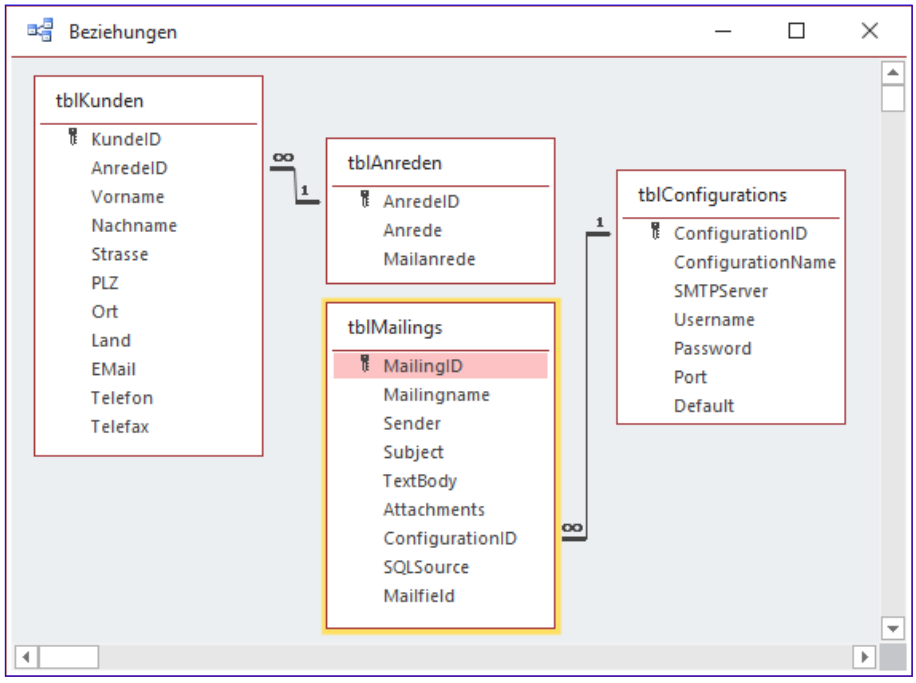

Bild 1: Datenmodell der Anwendung

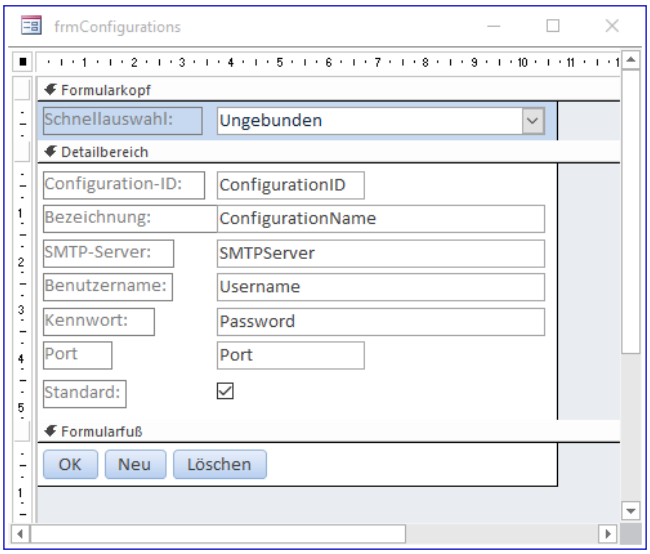

Bild 2: Entwurf des Formulars **frmConfigurations**

- Schaltfläche **cmdOK** (mit den anderen Schaltflächen in den Formularfuß)
- Schaltfläche **cmdNeu**
- Schaltfläche **cmdLoeschen**

Außerdem passen wir den Namen des Kontrollkästchens für das Feld **Standard** auf **chkDefault** an.

Schnellauswahl für die Konfiguration

Das Kombinationsfeld **cboSchnellauswahl** verwendet eine Abfrage auf Basis der Tabelle **tblConfigurations** als Datensatzherkunft. Dabei verwenden wir nur die ersten beiden Felder **ConfigurationID** und **ConfigurationName**, wobei wir diese nach dem Feld **ConfigurationName** sortieren wollen:

```
SELECT ConfigurationID, ConfigurationName
FROM tblConfigurations
ORDER BY ConfigurationName;
```

Damit das Kombinationsfeld nur die Bezeichnung der Konfiguration anzeigt, aber nicht den Inhalt des Autowertfeldes, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein. Damit das Kombinationsfeld nach dem Wechsel zu einem anderen Datensatz im Formular den Namen der aktuellen Konfiguration anzeigt, fügen wir die folgende Prozedur hinzu, die durch das Ereignis **Beim Anzeigen** ausgelöst wird:

```
Private Sub Form_Current()
    Me!cboSchnellauswahl = Me!ConfigurationID
End Sub
```

Gegebenenfalls ändert der Benutzer die Bezeichnung der Konfiguration. Damit sich dies direkt im Kombinationsfeld zur Schnellauswahl wiederspiegelt, aktualisieren wir dieses, wenn der Benutzer Daten im Formular aktualisiert und gespeichert hat:

```
Private Sub Form_AfterUpdate()
    Me!cboSchnellauswahl.Requery
End Sub
```

Aktionen beim Laden des Formulars

Beim Öffnen des Formulars soll dieses den ersten Datensatz anzeigen, der im Feld **Default** den Wert **True** enthält – der also als Standardkonfiguration definiert ist. Damit dies geschieht, suchen wir direkt in der Prozedur, die durch das Ereignis **Beim Laden** ausgelöst wird, mit der **FindFirst**-Methode des Recordsets des Formulars nach diesem Datensatz. Unabhängig davon, ob dies einen Datensatz findet, soll das Kombinationsfeld **cboSchnellauswahl** auf den gleichen Datensatz eingestellt werden, den auch das Formular anzeigt. Dafür stellen wir den Wert von **Me!cboSchnellauswahl** anschließend auf **Me!ConfigurationID** ein:

```
Private Sub Form_Load()
    Me.Recordset.FindFirst "Default = True"
    Me!cboSchnellauswahl = Me!ConfigurationID
End Sub
```

Auswahl einer anderen Konfiguration per Kombinationsfeld

Damit das Formular nach der Auswahl eines anderen Datensatzes über das Kombinationsfeld **cboSchnellaus-**

wahl den gewünschte Datensatz anzeigt, fügen wir dem Kombinationsfeld eine Prozedur für das Ereignis **Nach Aktualisierung** hinzu. Diese sieht wie folgt aus und prüft zunächst, ob im Kombinationsfeld überhaupt ein Datensatz ausgewählt ist. Falls ja, sucht die Prozedur mit der **FindFirst**-Methode des Recordsets des Formulars nach dem betroffenen Datensatz und stellt diesen ein:

```
Private Sub cboSchnellauswahl_AfterUpdate()
    If Not Nz(Me!cboSchnellauswahl, 0) = 0 Then
        Me.Recordset.FindFirst "ConfigurationID = " µ
                                & Me!cboSchnellauswahl
    End If
End Sub
```

Einstellen der Standardkonfiguration

Das Feld **Default** der Tabelle **tblConfigurations** legt fest, welche Konfiguration standardmäßig verwendet werden soll. Das bedeutet schlicht und einfach, dass diese Konfiguration beim Anlegen neuer Mailings für das Fremdschlüsselfeld **ConfigurationID** der Tabelle **tblMailings** festgelegt wird. Außerdem soll diese Konfiguration beim Öffnen des Formulars **frmConfigurations** standardmäßig angezeigt werden, was wir weiter oben bereits realisiert haben.

Mit dem Kontrollkästchen wollen wir dem Benutzer die Möglichkeit bieten, eine andere Konfiguration als Standardkonfiguration festzulegen. Dabei nutzen wir als Erstes die Prozedur, die durch das Ereignis **Vor Aktualisierung** des Kontrollkästchens **chkDefault** ausgelöst wird. Diese prüft, ob der Benutzer soeben den Haken aus dem Kontrollkästchen entfernt hat. Falls ja, soll eine Meldung erscheinen, die den Benutzer darauf hinweist, dass er zum Ändern der Standardkonfiguration erst zu der gewünschten Standardkonfiguration wechseln muss und diese dann mit einem Klick auf das Kontrollkästchen mit dem Text **Standard** festlegen kann. Das Ereignis **Vor Aktualisierung** nutzen wir deshalb, weil wir hier mit dem **Cancel**-Parameter einstellen können, dass die Änderung rückgängig gemacht wird, wenn der Benutzer das Kontrollkästchen deaktiviert hat:

```
Private Sub chkDefault_BeforeUpdate(Cancel As Integer)
    If Not Me!chkDefault Then
        MsgBox "Um eine andere Konfiguration zur µ
         Standardkonfiguration zu machen, aktivieren Sie µ
           diese Option für die gewünschte Konfiguration."
        Cancel = True
    End If
End Sub
```

Wenn der Benutzer hingegen das Kontrollkästchen **chkDefault** aktiviert hat, soll die aktuell angezeigte Konfiguration als Standardkonfiguration eingestellt werden. In diesem Fall feuert auch noch das Ereignis **Nach Aktualisierung** des Kontrollkästchens, was die folgende Prozedur auslöst:

```
Private Sub chkDefault_AfterUpdate()
    Dim db As DAO.Database
    If Me!chkDefault Then
        Set db = CurrentDb
        db.Execute "UPDATE tblConfigurations µ
           SET Default = 0 WHERE NOT ConfigurationID = " µ
                    & Me!ConfigurationID, dbFailOnError
    End If
End Sub
```

Diese Prozedur prüft, ob **chkDefault** den Wert **True** hat, also ob der Benutzer das Kontrollkästchen für diesen Datensatz aktiviert hat. Falls ja, führt die Prozedur mit der **Execute**-Methode des **Database**-Objekts eine Abfrage aus, die den Wert des Feldes **Default** für alle anderen Datensätze der Tabelle **tblConfigurations** auf **0** einstellt. Damit ist sichergestellt, dass das Feld **Default** nur für den aktuell angezeigten Datensatz den Wert **True** aufweist.

Löschen einer Konfiguration

Mit einem Klick auf die Schaltfläche **cmdLoeschen** kann der Benutzer die aktuelle Konfiguration löschen. Dazu löst die Schaltfläche die folgende Prozedur aus. Diese löscht den aktuellen Datensatz und versucht danach, den Datensatzzeiger auf den ersten Datensatz einzustellen, dessen Feld **Default** den Wert **True** aufweist:

```
Private Sub cmdLoeschen_Click()
    RunCommand acCmdDeleteRecord
    Me.Recordset.FindFirst "Default = True"
End Sub
```

Anlegen einer neuen Konfiguration

Um eine neue Konfiguration anzulegen, muss der Benutzer nur auf die Schaltfläche **cmdNeu** klicken. Diese springt dann zu einem neuen, leeren Datensatz:

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

Schließen des Formulars

Ein Klick auf die Schaltfläche **cmdOK** ruft die **DoCmd.Close**-Methode auf und schließt das aktuelle Formular:

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

Wenn Sie in die Formularansicht wechseln und einen Datensatz eingegeben haben, sieht dieses wie in Bild 3 aus.

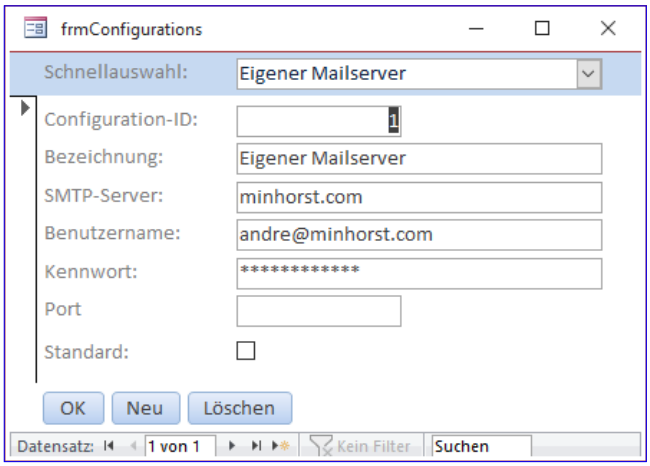

Bild 3: Das Formular **frmConfigurations** in der Formularansicht

Kennwort verbergen

Damit das Textfeld **Password** das Kennwort nicht direkt anzeigt, sondern Sternchen als Platzhalter verwendet, stellen wir die Eigenschaft **Eingabeformat** auf **Kennwort** ein.

Formular zum Verwalten der Mailings

Das Formular **frmMailings** dient zum Verwalten der Mailings. Es verwendet die Tabelle **tblMailings** als Datensatzquelle. Aus dieser Tabelle haben wir alle Felder in den Detailbereich der Entwurfsansicht gezogen.

Für die in Bild 4 markierten Felder stellen wir die Eigenschaft **Horizontaler Anker** auf den Wert **Beide** ein. So können wir die Felder vergrößern, indem wir die Breite des Formulars vergrößern. Anschließend müssen Sie die Bezeichnungsfelder dieser Steuerelemente markieren und für diese die Eigenschaft **Horizontaler Anker** auf **Links** einstellen, da diese durch die vorherige Einstellung auf **Rechts** festgelegt wurde.

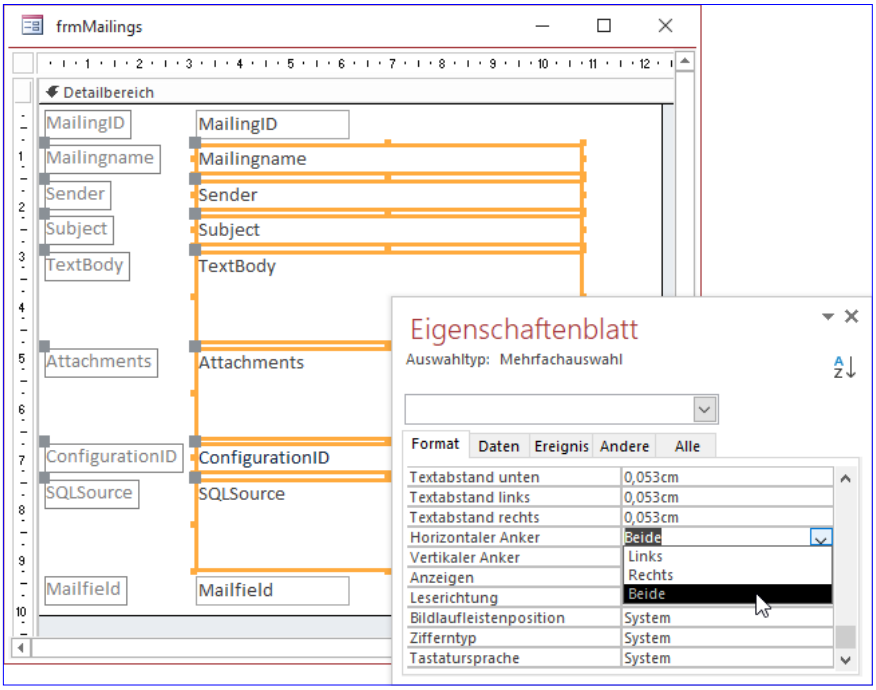

Bild 4: Das Formulars **frmMailings** in der Entwurfsansicht

Eines der Felder können wir außerdem in der Höhe anpassbar machen. Das ist für das Feld **TextBody** am sinnvollsten. Daher stellen wir seine Eigenschaft **Vertikaler Anker** auf **Beide** ein. Auch hier müssen wir die Eigenschaft **Vertikaler Anker** für das entsprechende Bezeichnungsfeld wieder auf **Oben** festlegen. Außerdem müssen wir die Eigenschaft **Vertikaler Anker** für alle Felder, die sich unterhalb des Textfeldes **TextBody** befinden, auf **Unten** einstellen, da diese sonst beim Vergrößern der Höhe des Formulars vom Textfeld **TextBody** überlappt werden.

Die Beschriftungen der Bezeichnungsfelder wurden bisher von den Feldnamen übernommen, diese passen wir jedoch auch noch an (noch nicht im Screenshot sichtbar).

Listenfeld für die Anhänge

Sie sehen die geänderten Bezeichnungen in Bild 5. Hier sehen Sie auch, wie Sie das Textfeld für die Anzeige der Daten des Feldes **Attachments** in ein Listenfeld umwandeln können – und zwar über das Kontextmenü. Anschließend ändern Sie noch den Namen des Listenfeldes in **lstAttachments**. Leider können wir die Bindung des Listenfeldes an das Feld **Attachments** danach nicht mehr nutzen, denn ein Listenfeld bindet man in der Regel an eine Datensatzherkunft wie eine Tabelle oder Abfrage. Deshalb bauen wir das Listenfeld ein wenig um. Als Erstes entfernen Sie den Inhalt der Eigenschaft **Steuerelementinhalt**. Dann stellen Sie die Eigenschaft **Herkunftstyp** auf **Wertliste** ein. Schließlich sorgen wir dafür, dass beim Wechseln des Datensatzes im Formular der Inhalt des Feldes **Attachments** als Wert des Listenfeldes eingestellt wird.

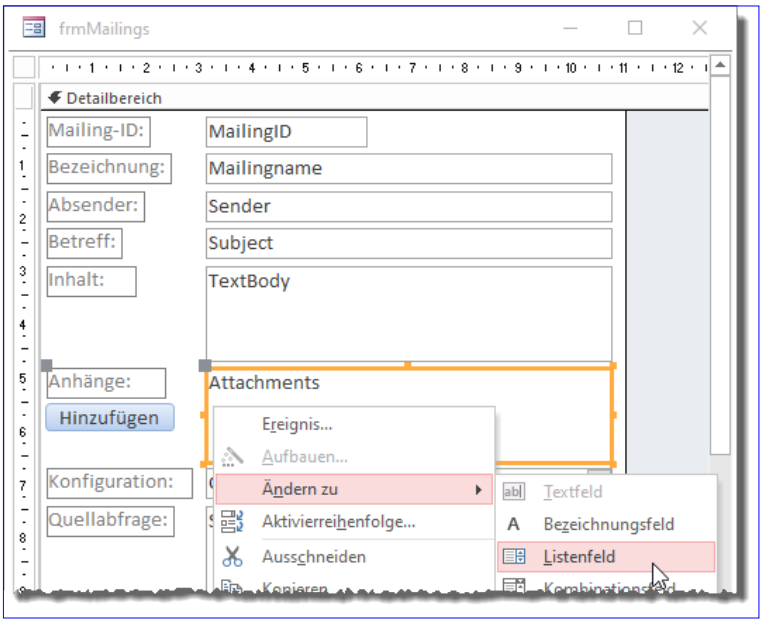

Bild 5: Ändern des Textfeldes für die Anhänge in ein Listenfeld

Dazu legen wir die Prozedur **Form_Current** mit der folgenden Anweisung an:

```
Private Sub Form_Current()
    Me!lstAttachments.RowSource = Me!Attachments
End Sub
```

Damit zeigt das Listenfeld bei jedem Datensatzwechsel automatisch die enthaltenen Attachments an.

Attachments hinzufügen

Dem Listenfeld **lstAttachments** stellen wir eine Schaltfläche namens **cmdHinzufuegen** zur Seite, welche die Ereignisprozedur aus Listing 1 auslöst. Diese stellen wir in ähnlicher Form im Beitrag **E-Mails mit Anlagen mit Outlook versenden** (**www.access-im-unternehmen.de/1357**) vor. Wir mussten allerdings einige Anpassungen vornehmen, da unser Listenfeld wie oben beschrieben nicht die Daten einer eigenen Tabelle oder Abfrage anzeigt, sondern den Inhalt des Feldes **Attachment** als Wertliste. Deshalb arbeiten wir beim Hinzufügen nicht mit der **AddItem**-Methode des Listenfeldes, sondern fügen die im Dateiauswahl-Dialog selektierten Daten dem Text aus dem Feld **Attachments** der Tabelle **tblMailings** hinzu, sofern die jeweilige Datei dort noch nicht enthalten ist. Für die Nutzung des Dateiauswahl-Dialogs fügen Sie dem VBA-Projekt noch einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** hinzu.

Attachments leeren

Im Gegensatz zu dem Listenfeld für Anlagen aus dem soeben genannten Beitrag wollen wir hier nur eine Schaltfläche zum Leeren des Listenfeldes hinzufügen und nicht das Selektieren einzelner zu entfernender Einträge erlauben. Dazu hinterlegen wir für die Schaltfläche **cmdLeeren** die folgende Ereignisprozedur:

```
Private Sub µ
          cmdLeeren_Click()
    Me!Attachments = Null
    Me!lstAttachments.µ
              RowSource = ""
End Sub
```

Diese leert sowohl das Feld **Attachments** der Datensatzquelle als auch das Listenfeld **lstAttachments**.

Konfiguration automatisch auswählen

Damit bei einem neuen Mailing direkt die als Standard markierte Konfiguration im Feld **cboConfigurationID** ausgewählt wird, legen wir die folgende Prozedur für das Ereignis **Beim Anzeigen** des Formulars fest:

```
Private Sub Form_Current()
    Dim lngConfigurationID As Long
    If Me.NewRecord Then
        lngConfigurationID = Nz(DLookup("ConfigurationID", µ
              "tblConfigurations", "Default = True"), 0)
        If Not lngConfigurationID = 0 Then
            Me!cboConfigurationID.DefaultValue = µ
                                    lngConfigurationID
        End If
```

```
Private Sub cmdHinzufuegen_Click()
    Dim objFileDialog As FileDialog
    Dim l As Long, Dim m As Long
    Dim bolVorhanden As Boolean
    Dim strAttachments As String
    Dim varAttachment As Variant
    strAttachments = Me!lstAttachments.RowSource
    Set objFileDialog = FileDialog(msoFileDialogFilePicker)
    With objFileDialog
        .AllowMultiSelect = True
        .Title = "Anlagen auswählen"
        .Filters.Clear
        .Filters.Add "Alle Dateien", "*.*"
        .ButtonName = "Hinzufügen"
        If .Show = True Then
            For l = 1 To .SelectedItems.Count
                If Not Len(Me!lstAttachments.RowSource + .SelectedItems(l)) > 32750 Then
                    bolVorhanden = False
                    For Each varAttachment In Split(strAttachments, ";")
                        If varAttachment = .SelectedItems(l) Then
                            bolVorhanden = True
                            Exit For
                        End If
                    Next varAttachment
                    If Not bolVorhanden Then
                        strAttachments = strAttachments & ";" & .SelectedItems(l)
                    End If
                    If Left(strAttachments, 1) = ";" Then
                        strAttachments = Mid(strAttachments, 2)
                    End If
                Else
                    MsgBox "Es können keine weiteren Dateien hinzugefügt werden."
                    Exit Sub
                End If
            Next l
        End If
    End With
    Me!lstAttachments.RowSource = strAttachments
    Me!Attachments = strAttachments
End Sub
```

Listing 1: Prozedur zum Hinzufügen von Anlagen

```
    End If
    Me!lstAttachments.RowSource = Me!Attachments
End Sub
```

Die Prozedur prüft zunächst, ob das Formular gerade einen neuen, leeren Datensatz anzeigt. Falls ja, ermittelt Sie den Primärschlüsselwert des Datensatzes der Tabelle **tblConfigurations**, dessen Feld **Default** den Wert **True** aufweist und schreibt diesen in die Variable **lngConfigurationID**. Ist dieser nicht **0**, stellt die Prozedur den Wert des Kombinationsfeldes **cboConfigurationID** aus **lngConfigurationID** ein.

Aktuelle Konfiguration bearbeiten

Das Formular für die Verwaltung der Mailings soll neben einem Kombinationsfeld zur Auswahl der zu verwendenden Konfiguration auch noch die Möglichkeit bieten, die Konfiguration zu öffnen und zu bearbeiten beziehungsweise eine neue Konfiguration anzulegen. Deshalb fügen wir neben dem Kombinationsfeld **cboConfigurationID** noch eine Schaltfläche namens **cmdKonfigurationBearbeiten** hinzu.

Damit diese Schaltfläche beim Ändern der Größe des Formulars nicht hinter anderen Steuerelementen verschwindet, deren Größe ebenfalls angepasst wird, stellen Sie die Eigenschaft **Horizontaler Anker** auf **Rechts** und **Vertikaler Anker** auf **Unten** ein. Die Platzierung der Schaltfläche können Sie Bild 6 entnehmen.

Um die Konfiguration bearbeiten und eventuelle Änderungen direkt in das Kombinationsfeld übernehmen zu können, öffnen wir das Formular über die Schaltfläche mit dem Wert **acDialog** für den Parameter **WindowMode**. Außerdem übergeben wir mit dem Parameter **OpenArgs** den Primärschlüsselwert des aktuell ausgewählten Datensatzes der Tabelle **tblConfigurations**.

Durch das Öffnen des Formulars als modalen Dialog wird der aufrufende Code solange angehalten, bis der Benutzer das Formular schließt oder dieses unsichtbar geschaltet wird. Wir wollen dafür sorgen, dass das Formular dann unsichtbar wird, damit wir anschließend noch die **ConfigurationID** der aktuell im Formular selektierten Konfiguration ermitteln können. Danach wollen wir das Formular dann schließen. Dies alles erledigen wir mit der Prozedur aus Listing 2. Wichtig ist hier erst einmal, dass wir das Formular wie angegeben als modalen Dialog öffnen und mit dem aktuellen Wert für **ConfigurationID** als Öffnungsargument anzeigen – den Rest sehen wir uns gleich an.

Das Formular **frmConfigurations** soll beim Laden prüfen, ob ein Öffnungsargument übergeben wurde. Dazu prüfen wir den Wert der Eigenschaft **OpenArgs**, welche das Öffnungsargument gegebenenfalls enthält, auf eine leere Zeichenkette. Ist die Zeichenkette nicht leer, soll das Formular **frmConfigurations** beim Öffnen direkt auf den Datensatz mit der übergebenen **ConfigurationID** eingestellt werden. Anderenfalls verfahren wir so, wie wir es bereits mit der bisherigen Methode getan haben:

```
Private Sub Form_Load()
    If Not Nz(Me.OpenArgs, "") = "" Then
        Me.Recordset.FindFirst "ConfigurationID = " µ
            & Me.OpenArgs
    Else
        Me.Recordset.FindFirst "Default = True"
    End If
    Me!cboSchnellauswahl = Me!ConfigurationID
End Sub
```

Damit das Formular **frmConfigurations** bei unserem Plan mitspielt, müssen wir noch dafür sorgen, dass es beim Klick auf **OK** nicht geschlossen, sondern nur ausgeblendet wird. Auch dafür wollen wir herausfinden, ob es vom Formular **frmMailings** aus geöffnet wurde. Das ist wieder der Fall, wenn das Öffnungsargument in der Eigenschaft **OpenArgs** nicht leer ist. Klickt der Benutzer in diesem Fall auf **cmdOK**, soll aber nicht nur das Formular mit der Einstellung **Visible = False** ausgeblendet werden. In diesem Fall wäre ein eventuell neu angelegter Konfigurationsdatensatz nämlich noch nicht gespeichert. Das holen wir durch das Einstellen der **Dirty**-Eigenschaft auf den Wert False nach, bevor wir das Formular ausblenden: