

# ACCESS

## IM UNTERNEHMEN

### SOFTWARE FREISCHALTEN

Schützen Sie Ihre Software vor unerlaubter Benutzung (ab S. 46).



### In diesem Heft:

#### SCHNELLE LISTENFELDSUCHE

Durchsuchen Sie Listenfelder blitzschnell per Tasteneingabe.

#### PERFEKTE SCHRIFTART

Wählen Sie eine ergonomische Schriftart für die Arbeit im VBA-Editor aus.

#### OUTLOOK-MAILS: FLEXIBLER ABSENDER

Stellen Sie den Outlook-Absender per VBA-Code ein.

SEITE 2

SEITE 10

SEITE 40

## Software schützen

**Wenn Sie Ihre mit viel Mühe entwickelte Software zum Kunden geben, möchten Sie sicherstellen, dass diese auch dort bleibt und nicht über das Internet verteilt wird. Ein Weg ist, die Software nur dann betriebsbereit zu machen, wenn der Kunde diese mit einem Schlüssel freigeschaltet hat. Und damit er die freigeschaltete Datenbank dann nicht doch noch weitergibt, binden wir den Schlüssel auch noch an die E-Mail-Adresse des Kunden.**



Wenn der Kunde die Software dann doch weitergibt, sollte dies tunlichst nicht auffliegen: Wenn Sie die Datenbank dann nämlich auf einem File-Sharing-Portal finden, können Sie den Schuldigen schnell ermitteln. Die technischen Grundlagen zu dieser Vorgehensweise vermitteln wir in unserem Beitrag **Software freischalten per Schlüssel** ab Seite 46. Hier erfahren Sie, wie Sie einen solchen Schlüssel abhängig von der E-Mail-Adresse des Benutzers erzeugen. Außerdem können Sie die weiterzugebende Anwendung mit Funktionen ausstatten, die erst nach Eingabe des Schlüssels nutzbar sind. Und Sie können sogar den freigeschalteten Funktionsumfang definieren und festlegen, wie lange die Anwendung mit dem Schlüssel nutzbar ist. Damit können Sie zum Beispiel zeitlich limitierte Lizenzen für eine Software verteilen.

Passend zu diesen Techniken schauen wir uns im Beitrag **Ribbon-Elemente aktivieren und deaktivieren** (ab Seite 21) an, wie Sie die Elemente des Ribbons abhängig von bestimmten Variablen aktivieren oder deaktivieren können. Wenn Sie dem oben genannten Schlüssel für Ihre Anwendung den freizuschaltenden Funktionsumfang in codierter Form mitgegeben haben, erfahren Sie in diesem Beitrag, wie Sie die Steuerelemente im Ribbon abhängig von diesem Code aktivieren oder deaktivieren. So kann der Benutzer die freigeschalteten Funktionen nutzen und bekommt gleichzeitig vielleicht noch Interesse an den zwar deaktivierten, aber bereits sichtbaren weiteren Funktionen.

Der Beitrag **Daten in Properties verstecken** zeigt ab Seite 42, wie Sie Informationen, die nicht leicht zugänglich in Tabellen landen sollen, an anderer Stelle in der Anwendung verbergen können. Das wäre beispielsweise ein

Platz zum Ablegen etwa des Startdatums bei der Nutzung einer per Schlüssel freigeschalteten Anwendung.

Wer viel Code schreibt, sollte für eine perfekte Arbeitsumgebung sorgen. Dazu gehören nicht nur Schreibtisch, Tastatur und Monitor, sondern auch eine perfekt lesbare Schriftart. Die Basics zu diesem Thema hat Sascha Trowitzsch ab Seite 10 im Beitrag **Schriftarten im VBA-Editor** zusammengetragen.

Wenn Sie E-Mails per VBA über Outlook verschicken, haben Sie sich vielleicht schon gefragt, wie Sie den Absender einstellen können. Das gelingt nämlich nicht einfach durch Zuweisen der gewünschten E-Mail-Adresse. Wie Sie es dennoch hinbekommen, zeigt der Beitrag **Outlook-Absender einstellen** ab Seite 40.

Für alle Leser, die selbst Pakete mit DHL verschicken oder eine Funktion zum Erstellen geeigneter Etiketten in die Anwendungen ihrer Kunden einbauen wollen, haben wir im Beitrag **DHL-Versand vorbereiten** ab Seite 60 die richtige Lösung. DHL hat nämlich die bisher verwendete Schnittstelle Intraship eingestellt, die wir vor einiger Zeit bereits einmal beschrieben haben. Schließlich liefert der Beitrag **Dateieigenschaften ermitteln** ab Seite 38 viel Know-how rund um die Ermittlung der Eigenschaften der Dateien, die sich in Ihrem Dateisystem befinden.

Viel Spaß beim Lesen!

A handwritten signature in black ink, appearing to read 'A. Minhorst'.

Ihr André Minhorst

## Listenfeld mit Schnellsuche per Taste

Neulich fiel mir in einer Benutzerumgebung auf, dass ich dort im Listenfeld per Eingabe eines Zeichens direkt zu den Einträgen springen konnte, die mit diesem Zeichen beginnen. Das ist immer ein schöner Anlass, dies in Access nach zu programmieren. Dieser Beitrag liefert also eine Erweiterung für Listenfelder, mit der Sie dem Benutzer die Auswahl der enthaltenen Einträge noch leichter machen können.

Als Beispiel dient die Tabelle **tblArtikel** der Süd Sturm-Beispieldatenbank. Diese verwenden wir als Datensatzherkunft eines neuen Listenfeldes in einem Formular namens **IstArtikel** – und zwar über eine Abfrage, die nur die beiden Felder **ArtikelID** und **Artikelname** liefert. Dabei sortiert die Abfrage die Datensätze noch nach dem Artikelnamen (siehe Bild 1).

Mit dieser Abfrage statten wir dann das Listenfeld **IstArtikel** im Formular **frmListenfeld-Schnellsuche** aus. Damit das Listenfeld nur die Spalte mit den Artikelnamen anzeigt und nicht den Primärschlüsselwert der Datensätze, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein. Das Formular sieht dann im Entwurf wie in Bild 2 aus.

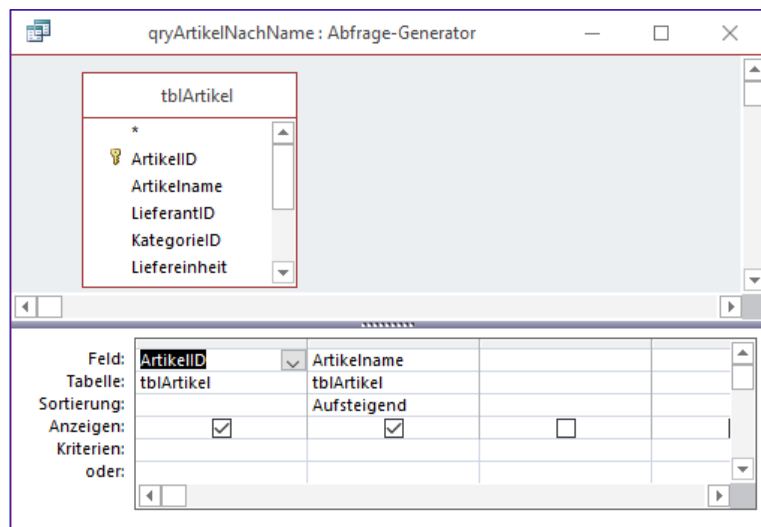


Bild 1: Abfrage als Datenherkunft des Listenfeldes

### Funktion schon da ...

Nun wollen wir eine Funktion zum Listenfeld hinzufügen, die dafür sorgt, dass die Eingabe eines Buchstabens direkt zum ersten Datensatz springt, der mit diesem Buchstaben anfängt.

Als ich dann allerdings testweise in die Formularansicht des Formulars wechselte und einen Buchstaben eingegeben habe, während das Listenfeld den Fokus hat, stellte ich erstaunt fest, dass diese Funktion bereits in das Listenfeld von Access integriert ist!

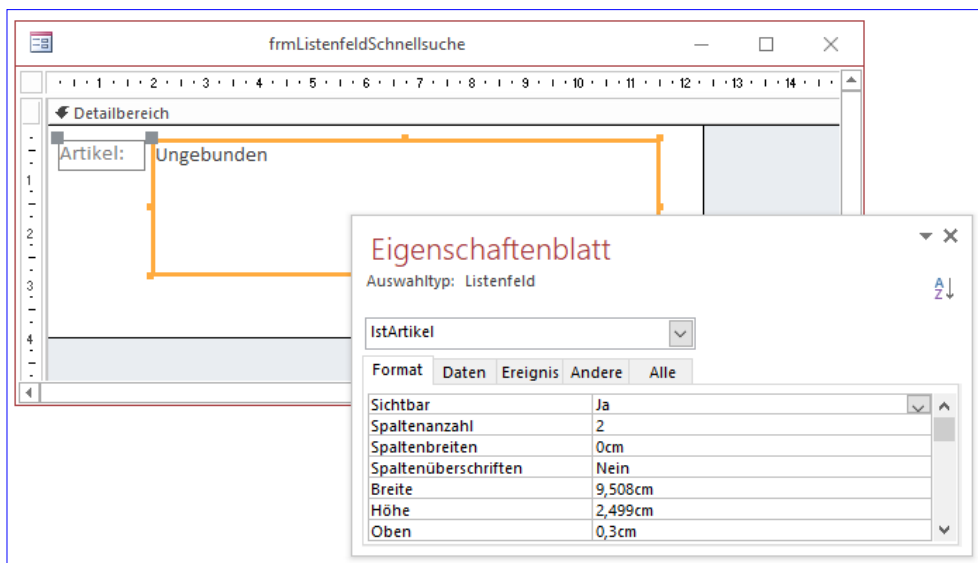
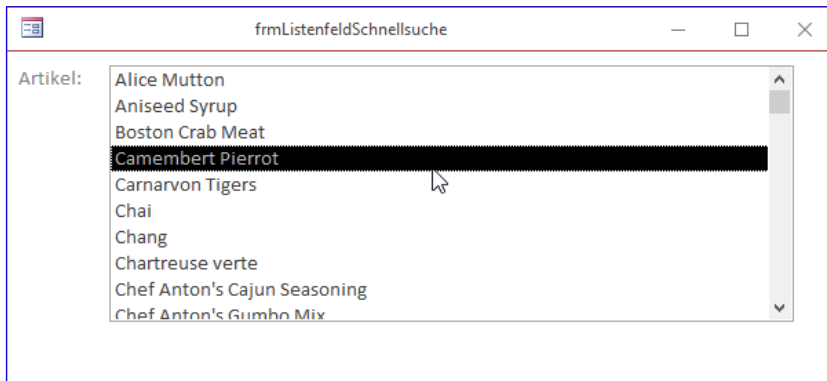


Bild 2: Entwurfsansicht des Beispielformulars



**Bild 3:** Formular in Aktion

Das Eintippen des Buchstabens C beispielsweise sorgte direkt zur Markierung des ersten Datensatzes, der mit dem Buchstaben C beginnt (siehe Bild 3).

Das zeigt mal wieder, dass auch eine Anwendung wie Access, die praktisch seit 2010 überhaupt nicht mehr weiterentwickelt wurde, noch Funktionen offenbart, die man bis dahin noch nie genutzt hat, geschweige denn erkannt hat. Nun: Ich gehe davon aus, dass der eine oder andere diese Funktion bereits kennt – an mir ist diese bisher jedoch vorübergegangen.

Nun denn: So einfach lassen wir uns nicht abspesen und bieten Varianten zum Standardverhalten des Listenfeldes bei Eingabe eines Buchstabens an. Das Standardverhalten sieht wie folgt aus:

Wenn Sie einen Buchstaben eingeben, zu dem das Listenfeld in der ersten Spalte einen Wert enthält, der mit diesem Buchstaben beginnt, wird dieser Datensatz markiert.

Geben Sie den gleichen Buchstaben nochmals ein, wird die Markierung zum nächsten Eintrag verschoben, dessen Wert in der ersten Spalte mit diesem Buchstaben beginnt. Ist nur ein pas-

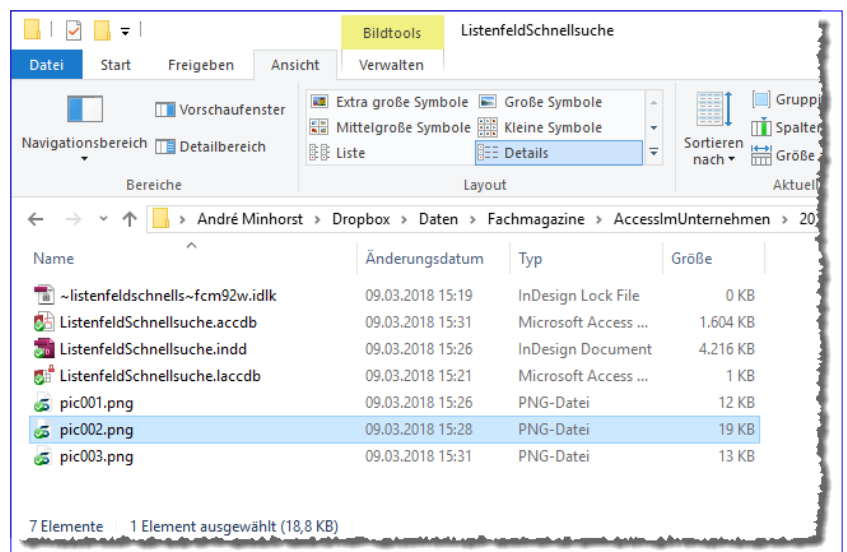
sender Eintrag vorhanden, bleibt dieser markiert. Liegen mehrere Einträge vor, durchläuft das Listenfeld diese Einträge immer wieder, wenn der Benutzer die gleiche Taste mehrfach betätigt. Ist kein passender Eintrag vorhanden, bleibt der aktuelle Eintrag markiert.

### Varianten

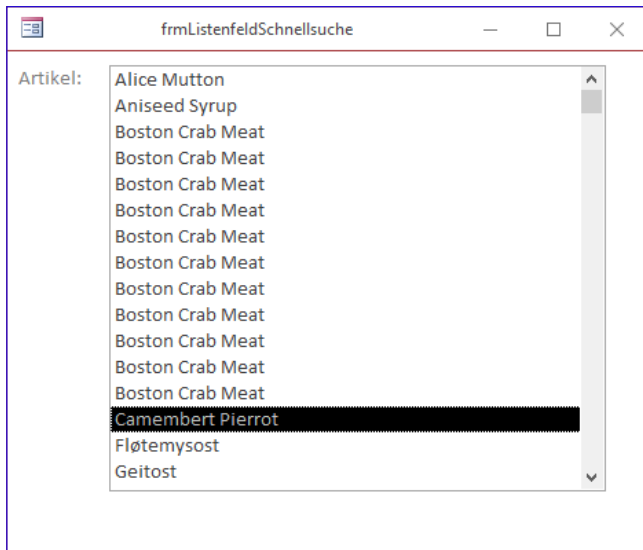
Die erste Variante, die wir uns ansehen wollen, entspricht dem Verhalten, das die Dateiliste im Windows Explorer bietet.

Wenn die Dateiliste den Fokus hat und Sie geben einen Buchstaben ein, verhält sich die Liste wie das Listenfeld in Access, das heißt, dass der erste Eintrag markiert wird, dessen Dateiname mit dem angegebenen Anfangsbuchstaben beginnt.

Das funktioniert auch noch genauso wie im Listenfeld, wenn Sie beispielsweise drei Einträge haben, die mit **p** beginnen und mehrfach die Taste **p** drücken. Wenn Sie aber bei der Konstellation von Bild 4 erst **p** und dann direkt **I** drücken, springt die Markierung nicht erst zum Eintrag **pic001** und dann zum ersten Eintrag, der mit **I** beginnt. Stattdessen tut sich nichts und die Markierung



**Bild 4:** Tastatureingabe im Windows Explorer



**Bild 5:** Die **FindFirst**-Methode funktioniert für unseren Anwendungszweck funktioniert.

den wir unternehmen, soll nur zum ersten Element mit dem Anfangsbuchstaben springen, den wir mit der Tastatur eingeben.

Dazu wollen wir es mit der **FindFirst**-Methode probieren, die wir in der Prozedur, die durch das Ereignis **Bei Taste ab** auslösen, wie folgt einsetzen:

```
Private Sub lstArtikel_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim strSQL As String
    strSQL = "Artikelname LIKE '" & Chr(KeyCode) & "*'"
    Debug.Print strSQL
    Me!lstArtikel.Recordset.FindFirst strSQL
End Sub
```

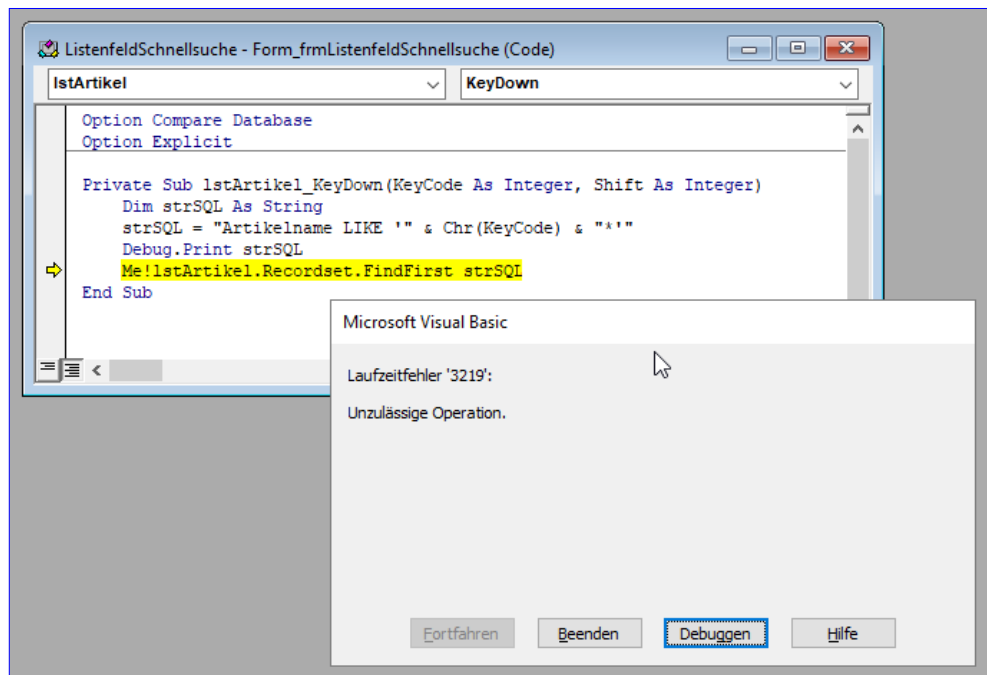
verharrt gefühlt eine Sekunde auf dem Eintrag mit dem Anfangsbuchstaben **p**. Der Grund ist einfach: Im Windows Explorer können Sie auch durch zügige Eingabe mehrerer aufeinanderfolgender Zeichen zu Einträgen springen, die mit diesen Zeichen beginnen.

Das Ergebnis ist allerdings ernüchternd, wie Bild 5 zeigt: Bei der Eingabe der Taste **C** beispielsweise springt die Markierung noch zum richtigen Datensatz, aber wenn wir diese Taste mehrfach betätigen, sieht das Resultat wie in der Abbildung aus.

Wenn Sie also wie im Beispiel in der Abbildung **pic003** eingeben, dann springt die Markierung erst zum ersten Eintrag mit **p** und dann weiter zum Eintrag **pic003.png**.

Der Explorer wartet also immer einige Augenblicke auf weitere schnell eingegebene Zeichen, bevor er davon ausgeht, dass nun doch ein anderer Eintrag angesteuert werden soll.

Diese Funktion wollen wir nun für das Listenfeld abbilden. Der erste Ansatz,



**Bild 6:** Fehlermeldung nach mehrmaligem Auslösen der Prozedur

Darüber hinaus gibt es nach ein paar weiteren Tastatureingaben noch eine nicht reproduzierbare Fehlermeldung (siehe Bild 6).

### Listeneinträge durchlaufen

Also müssen wir uns einen anderen Ansatz überlegen.

Dieser ist nur sinnvoll, wenn das Listenfeld nicht allzu viele Datensätze enthält, denn wir durchlaufen hier die Listeneinträge, bis wir den ersten passenden Eintrag gefunden haben.

Der erste Entwurf des Codes sieht wie in Listing 1 aus. In dieser Prozedur durchlaufen wir alle Datensätze vom ersten bis zu dem Datensatz, dessen erstes Zeichen in der Spalte mit dem Index **1, i** mit dem eingetippten Zeichen übereinstimmt.

Wenn wir diesen Code ausprobieren und etwa den Buchstaben **B** eingeben, funktioniert es. Wenn wir allerdings einen Anfangsbuchstaben eingeben, für den es mehrere Artikelnamen gibt – wie beim Buchstaben **C** –, erhalten wir das Ergebnis aus Bild 7. Wir landen also immer genau einen Eintrag unter dem erwarteten Ergebnis.

Warum das so ist, lässt sich nicht so einfach erklären.

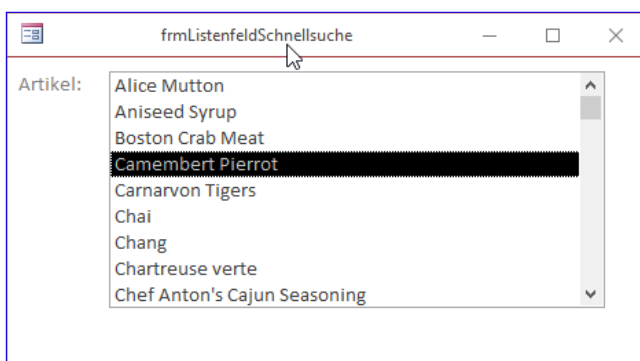


Bild 8: Markierung über den Direktbereich

```
Private Sub 1stArtikel_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim i As Integer
    For i = 0 To Me!1stArtikel.ListCount - 1
        Debug.Print i, Left(Me!1stArtikel.Column(1, i), 1), Me!1stArtikel.Column(1, i)
        If Left(Me!1stArtikel.Column(1, i), 1) = Chr(KeyCode) Then
            Me!1stArtikel.Selected(i) = True
            Exit For
        End If
    Next i
End Sub
```

Listing 1: Versuch, den ersten passenden Eintrag zu finden und zu markieren

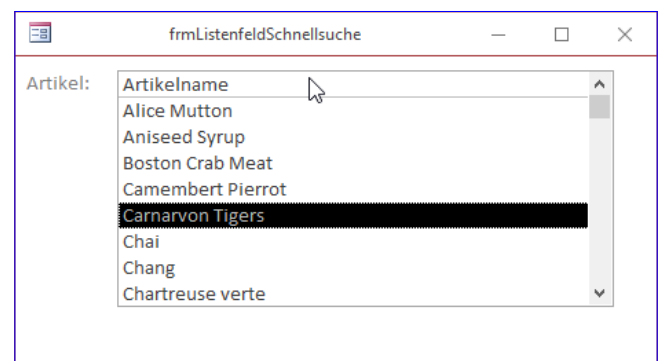


Bild 7: Falsche Selektion

Zu Analysezwecken haben wir per **Debug.Print**-Anweisung die jeweils durchlaufenen Werte für **i**, den ersten Buchstaben des Artikels und den kompletten Artikelnamen ausgeben lassen. Auch das lieferte keine Auffälligkeiten – wenn wir beispielsweise den Buchstaben **C** eingeben, erschien im Direktbereich die folgende Ausgabe:

0	A	Alice Mutton
1	A	Aniseed Syrup
2	B	Boston Crab Meat
3	C	Camembert Pierrot

Es wurde aber dennoch nicht der Eintrag **Camembert Pierrot**, sondern **Carnarvon Tigers** markiert – also genau ein Eintrag weiter unten. Wir gingen also davon aus, dass die Prozedur für Eintrag mit dem Index-Wert **3** die Eigenschaft **Selected** auf den Wert **True** einstellt. Den entsprechenden Befehl haben wir dann über den Direktbereich abgesetzt:

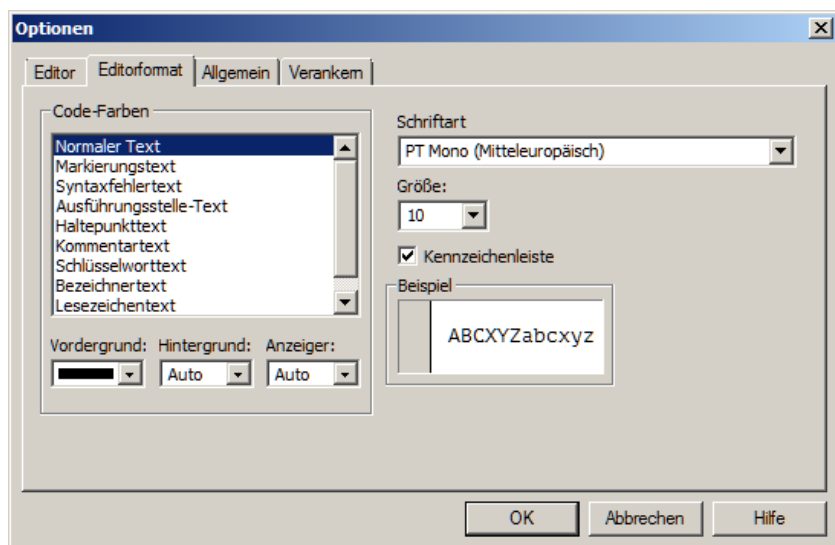
## Schriftarten im VBA-Editor

Sie programmieren viel unter VBA und scrollen fortwährend in den Modulfenstern? Ihre Augen werden immer schlechter und die Dioptrien der Lesebrille erhöhen sich? Dann wird es möglicherweise Zeit, sich mit der Schriftart zu beschäftigen, die Sie für Ihren Code verwenden. Denn Courier New ist keineswegs das Maß aller Dinge, und es gibt zahlreiche Alternativen, deren Erprobung sich lohnen könnte.

### Schriftarten einstellen

In der Regel macht man sich beim Programmieren wenig Gedanken über die Gestalt der Entwicklungsumgebung und nimmt die Vorgaben klaglos hin. Schreiben Sie Texte unter Word, so ist das Einstellen einer passenden Schriftart eine Selbstverständlichkeit. Beim Programmieren unter VBA hingegen wird die vorgegebene Schriftart meist unbesehen übernommen. Und die hört auf den schönen Namen **Courier New** in der Größe 10 Punkt. Nun existiert VBA schon seit zwei Jahrzehnten, ohne dass sich an dieser Tatsache etwas geändert hätte. Dabei verwenden andere Code-Fenster von Microsoft, wie die des aktuellen **Visual Studio**, die zweifellos günstigere Schriftart **Consolas**. Also könnte man einfach unter VBA ebenfalls diese Schriftart verwenden. Doch Gemach! Es gibt inzwischen noch viele weitere freie Schriftarten, die speziell für das Programmieren entwickelt wurden. Deshalb lohnt sich ein Blick auf diese unbedingt, und wir haben sie mithilfe eines Test-Sets, das Sie in der Beispieldatenbank finden, einer Prüfung unterzogen.

Manuell stellen Sie die Schriftart der Code-Fenster unter VBA über das Menü **Extras** | **Optionen...** | **Editorformat** ein. Im dabei erscheinenden Dialog können Sie für verschiedene Schriftstile, die auf der linken Seite aufgelistet sind (siehe Bild 1), die Vorder- und Hintergrundfarbe festlegen, aber auch die Schriftarten samt Schriftgröße. Zum Ausprobieren verschiedener Schriftarten ist das vor



**Bild 1:** Über die Optionen der VBA-Umgebung lassen sich mehrere Schriftarten und Stile einstellen

allem deshalb unpraktisch, weil das Kombinationsfeld zur Font-Auswahl nur wenige Zeilen ausklappt. Zudem unterscheidet VBA hier nicht zwischen geeigneten und ungeeigneten Schriftarten. Ungeeignet sind die sogenannten **Proportionalen Schriften**, bei denen jedes Zeichen eine unterschiedliche angepasste Breite einnimmt. Darunter würde die Lesbarkeit erheblich leiden. Der Name einer Schrift wiederum sagt nichts über deren Charakter aus, so dass Sie auf externe Anwendungen angewiesen sind, um überhaupt die proportionalen von den nichtproportionalen Schriften unterscheiden zu können.

Nun gibt es **API**-Routinen, über die man die Schriftarten des Systems durchlaufen und zusätzlich nach Eigenschaften filtern kann. Nur gibt es leider keine einfache Methode, um eine Schriftart dann auch per VBA zu set-

zen. Zumindest haben wir keine gefunden. Die Auswahl einer Schriftart über den **Optionen**-Dialog wirkt sich nach Bestätigung über **OK** sofort aus. Programmtechnisch können wir hierfür keinen Ersatz anbieten, sondern nur einen Workaround, der im Folgenden noch beschrieben wird.

### Einstellung per VBA selbst

Die Einstellung der Schriftart speichert VBA nämlich in der Registry ab. Zu finden ist der Eintrag im Zweig

HKEY\_CURRENT\_USER\Software\Microsoft\VBA\7.0\Common.

Dort steht für den Schriftartnamen der Schlüssel **FontFace** und für die Größe der Schlüssel **FontHeight**. Da sich der Zweig im User-Bereich befindet, können die Werte auch ohne administrative Rechte programmatisch geändert werden. Die schlechte Nachricht ist, dass VBA die Schlüssel zur Laufzeit ignoriert und es auch nicht so etwas, wie einen Reload gibt, um es zu zwingen, die Schlüssel neu einzulesen. Die einzige Möglichkeit scheint ein kompletter Neustart von Access oder einer anderen Office-Anwendung zu sein. Der allerdings kann sehr wohl von einer Routine ausgelöst werden.

Das Setzen der neuen Werte übernimmt eine ziemlich einfache Prozedur, die Listing 1 abbildet. Sie übergeben ihr als Parameter den Namen der Schriftart, sowie optional ihre Größe, die sonst auf 10 Punkt festgelegt wird. Zum Beschreiben der Registry benutzen wir ein Objekt **WshShell**, welches sich in der Bibliothek **Windows Script Host Model (wshom.ocx)** verbirgt, die unter Windows immer installiert ist, falls nicht ein allzu sicher-

heitsbeeinträchtigter Administrator sie deaktiviert hat. Diese Bibliothek nehmen Sie in die Verweise des VBA-Projekts auf. Die Methode ist übrigens unter VBA generell die mit Abstand einfachste, um in die Registry zu schreiben.

Einmal gesetzt, müssen Sie anschließend Access zu einem Neustart bewegen. Das erledigen Sie etwa mit den folgenden Zeilen:

```
Dim sApp As String
sApp = "" & SysCmd(acSysCmdAccessDir) & "msaccess.exe" & ""
Shell sApp & " " & "" & CurrentDb.Name & "" & " /CMD NoStartDlg"
Application.Quit
```

**SysCmd** ermittelt den Pfad der Access-Applikation. Den Namen der ausführenden Datei hängen Sie an. Der **Shell**-Anweisung übergeben Sie anschließend diesen Pfad und zusätzlich den Dateinamen der aktuellen Datenbank, die sich aus **CurrentDb.Name** ergibt. Den Sinn der angehängten **CMD**-Anweisung werden wir gleich erläutern. Die Pfade sollten Sie immer in Anführungszeichen einbinden, damit **Shell** sie etwa bei Leerzeichen im Pfad nicht fehlinterpretiert. Sie starten damit also eine neue Instanz der Datenbank und beenden die aktuelle über **Quit**. Das Ganze erfolgt in der Regel so schnell, dass es kaum wahrnehmbar ist. Allerdings sollten Sie dann in der neuen Instanz auch den gleichen Zustand wiederherstellen, der vor dem Beenden der aktuellen zu sehen war. In der Demo-Anwendung geschieht das über einen Code des Startformulars **frmIntro**. Dort finden sich im Ereignis **Beim Laden** die folgenden Zeilen:

```
Private Sub ChangeVBAFont(ByVal FontName As String, Optional FontSize As Long = 10)
    Dim oReg As WshShell
    Set oReg = New WshShell
    oReg.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\VBA\7.0\Common\FontFace", FontName
    oReg.RegWrite "HKEY_CURRENT_USER\Software\Microsoft\VBA\7.0\Common\FontHeight", FontSize
End Sub
```

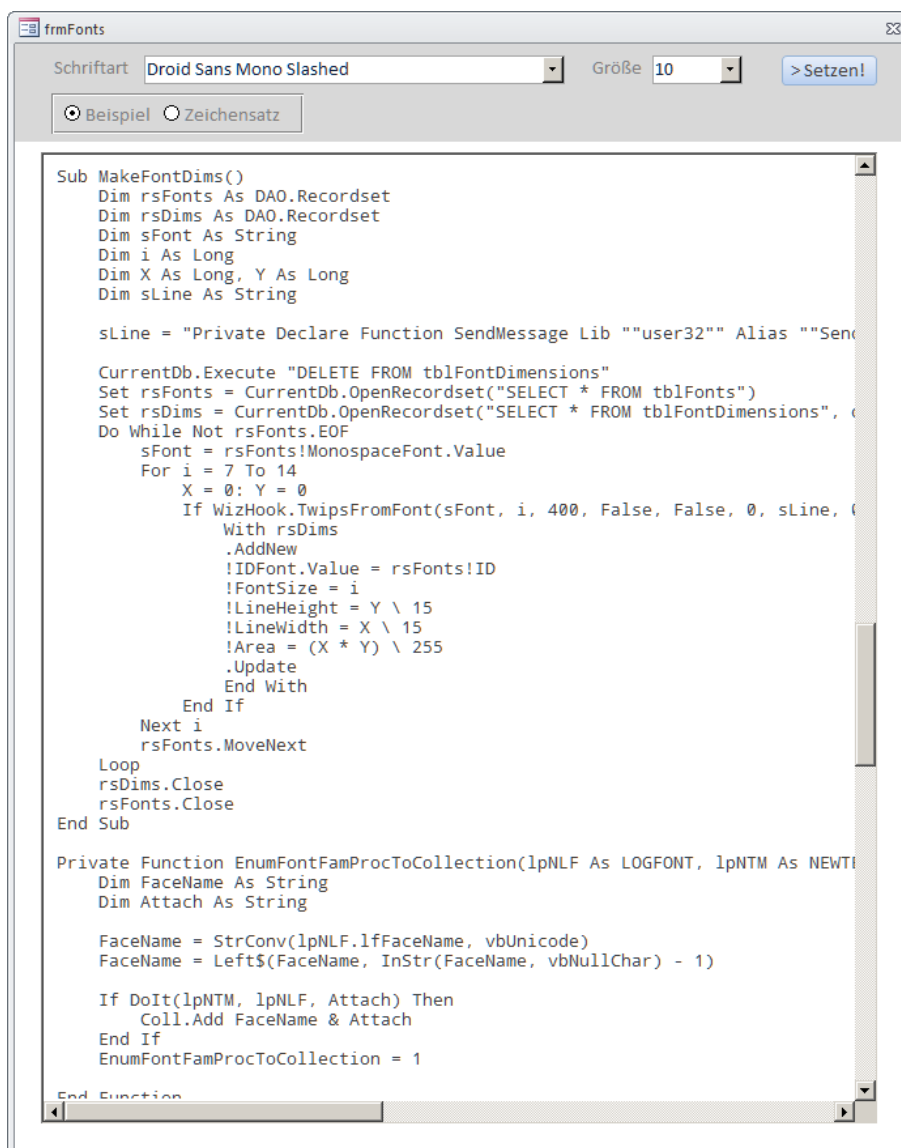
**Listing 1:** Über diese Prozedur lässt sich eine neue Schriftart für den VBA-Editor einstellen



```
If Command = "NoStartDlg" Then
    DoCmd.Close acForm, Me.Name
    DoCmd.OpenForm "frmFonts"
    DoCmd.OpenModule "mdlFonts", "GetFontsAsCollection"
End If
```

Ausgewertet wird hier zunächst die **Command**-Funktion von VBA. Sie gibt einen String aus, der über den Kommandozeilenparameter **CMD** gesetzt werden kann.

Normalerweise ist das ein Leer-String. Wir aber haben diesen zuvor in der Prozedur mit dem Wert **NoStartDlg** versehen. Also werden die weiteren Zeilen ausgeführt, das Intro-Formular geschlossen und stattdessen das Formular **frmFonts** geladen und zusätzlich das Modul **mdlFonts** geöffnet, in dem außerdem die Prozedur **GetFontsAsCollection** angesprungen wird. Dies ist bereits ein Vorgriff auf die Funktionalität des Formulars **frmFonts**.



**Bild 2:** Das Formular **frmFonts** zeigt einen Pseudo-Editor mit einstellbarer Schriftart mit Beispielcode an

### Einstellen der VBA-Schriftart über ein Formular

Das Formular **frmFonts** öffnet sich in der Demo automatisch, nachdem Sie das Intro-Formular schließen. Oder eben, nachdem die vorher beschriebene Prozedur aufgerufen wurde. Dann präsentiert es sich wie in Bild 2.

Den größten Teil nimmt die Textbox ein, in der ein Beispielcode abgebildet wird. Es handelt sich dabei übrigens nicht um eine normale Access-Textbox, sondern um ein ActiveX-Steuerelement, die **MSForms**-Variante, welche mehr Steuerungsmöglichkeiten zulässt. Die Schriftart können Sie über das Kombinationsfeld oben ebenso ändern, wie die Schriftgröße. Statt des Beispiel-Codes können Sie über die Optionsgruppe auch alle Zeichen des Zeichensatzes als Block in die Textbox holen (siehe Bild 3).

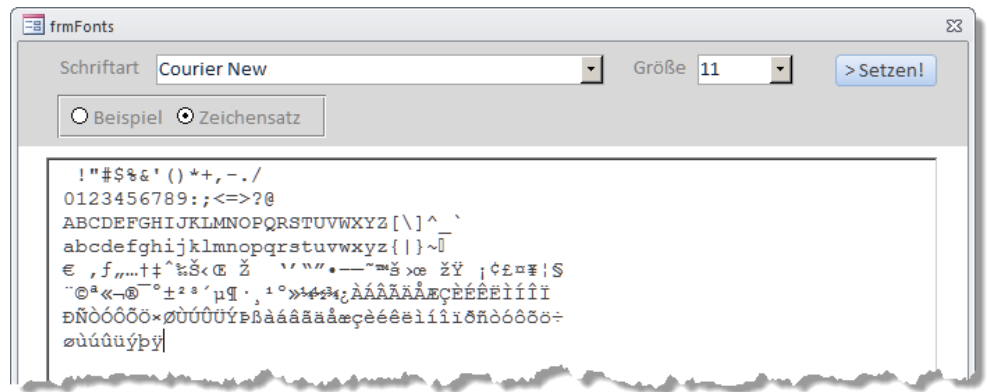
Sagt Ihnen der **Font** zu, so legen Sie ihn in der Registry über Klick auf den Button **Setzen!** fest. Access wird dann, wie beschrie-

ben, neu gestartet und das Formular erneut zur Ansicht gebracht. Gleichzeitig wird der VBA-Editor aktiviert und ebenfalls angezeigt, so dass Kontrolle über die neu gesetzte Schriftart besteht. Es hängt von Ihrem System ab, wie das abläuft: Entweder Sie haben zwei Monitore, wobei Access auf dem einen, die VBA-Umgebung auf dem anderen dargestellt wird. Oder Sie haben einen ziemlich großen Monitor, auf dem Sie beides nebeneinander platzieren. Ansonsten überdeckt nun VBA das Access-Fenster. Dargestellt wird aber immer das Modul **mdlFonts**, in dem die Prozedur **GetFontsAsCollection** ins Zentrum gebracht ist. Das also ist der Workaround, der leider nötig wird, weil wir keine andere Methode fanden, um das Verhalten des **Optionen**-Dialogs nachzustellen.

Wie ist das Formular nun aufgebaut? Beteiligt an ihm sind drei weitere Elemente, das Modul **mdlFonts** und die Tabellen **tblFonts**, **tblFontsDimensions**. Die Routinen des Moduls ermitteln über **API**-Funktionen die Schriftarten des Systems und filtern diese nach ihrem Stil, damit nur die für unsere Zwecke benötigten nicht-proportionalen Schriften (**Monospace**) übrigbleiben.

Diese werden dann in der Tabelle **tblFonts** abgelegt. Im zweiten Schritt errechnen weitere Prozeduren die Ausdehnung der jeweiligen Schrift und speichern Breite (**LineWidth**), wie Höhe (**LineHeight**), einer vorgegebenen Code-Zeile in der Tabelle **tblFontDimensions** ab.

Zusätzlich wird aus diesen beiden Parametern die von der Zeile eingenommene Fläche (**Area**) berechnet, damit die Tabelle später direkt nach dem Raumbedarf der Schrift sortiert werden kann. Die dafür herangezogene willkürliche Beispiel-Code-Zeile lautet:



**Bild 3:** Alternativ kann im Editorfenster auch der komplette Zeichensatz der Schrift angezeigt werden

```
Private Declare Function SendMessage Lib "user32" _
    Alias "SendMessageA" (ByVal hwnd As Long, ByVal wParam _
    As Long, ByVal lParam As Long, lParam As Long) As Long
```

Die beiden Tabellen sind 1:n miteinander verknüpft, weil zu einer Schriftart jeweils mehrere Größen berechnet werden. Die zweite dient dann als Unterdatenblatt für die erste – siehe Bild 4. **FontSize** ist die für die Schrift eingestellte Größe in Punkt. Die anderen Angaben sind Pixel-Größen.

**Schriftgrößen**

Aus den Tabellen geht, wie Sie sehen können, hervor, dass die Punkt-Größen keineswegs unmittelbar mit den resultierenden Pixel-Größen korrelieren. Eine 14-Punkt-Schrift etwa ist meist mehr, als doppelt so hoch, wie eine 7-Punkt-Schrift. Gleiches gilt für die horizontale Ausdehnung. Hier besteht kein linearer Zusammenhang und es ist das Geheimnis eines jeden Fonts, wie genau er gerendert wird.

Doch wozu sind all diese Nachforschungen gut? Einmal wird deutlich, dass verschiedene Schriftarten in der gleichen Punkt-Größe reell zu unterschiedlichen Zeilenhöhen und -breiten führen. Um also die Lesbarkeit eines Fonts mit der eines anderen vergleichen zu können, müssen unter Umständen abweichende Font-Größen verwendet werden. Aus der Abbildung können Sie ablesen, dass etwa **Liberation Mono** in der Größe 8 Punkt mit 12

2064 Liberation Mono					
ID	FontSize	LineHeight	LineWidth	Area	
15569	7	10	785	6926	
15571	9	14	1099	13575	
15570	8	12	1099	11636	
15572	10	15	1256	16623	
15573	11	17	1413	21195	
15575	13	19	1570	26320	
15574	12	18	1570	24935	
15576	14	22	1727	33524	
(Neu)					
2065 Lucida Console					
ID	FontSize	LineHeight	LineWidth	Area	
15577	7	9	785	6233	
15579	9	12	1099	11636	
15578	8	11	1099	10666	
15580	10	13	1256	14407	
15581	11	15	1413	18701	
15583	13	17	1570	23550	
15582	12	16	1570	22164	
15584	14	19	1727	28952	
(Neu)					
2026 Lucida Sans Typewriter					
ID	FontSize	LineHeight	LineWidth	Area	
15265	7	10	785	6926	
15267	9	14	1099	13575	
15266	8	12	1099	11636	
15268	10	16	1256	17731	
15269	11	18	1413	22441	
15271	13	20	1570	27705	
15270	12	19	1570	26320	
15272	14	23	1727	35047	
(Neu)					

**Bild 4:** In der Tabelle **tblFonts** finden sich alle installierten **Monospace**-Schriften mit ihren Abmessungen

Pixeln die gleiche Höhe besitzt, wie **Lucida Console** in 9 Punkt. (Das Unterdatenblatt ist hier übrigens nach **LineWidth** absteigend sortiert.) Folglich sollte ein visueller Vergleich zwischen beiden Schriftarten einmal mit 8 Punkt, einmal mit 9 Punkt erfolgen.

Weiter lässt sich aus **LineWidth** unmittelbar ablesen, wie viele Zeichen sich im Modulfenster darstellen lassen. Sicher ist in Ihrem Interesse, dass möglichst viele in ihren Editor passen und sie auch lange Code-Zeilen ohne horizontales Scrollen überblicken können. Aus der Abbildung allerdings ist die Analyse zunächst nicht ersichtlich, da hier alle drei Schriftarten dieselbe horizontale Ausdehnung aufweisen.

Deshalb verwenden Sie die Abfragen **qry\_LineWidth** und **qry\_LineHeight** in der Demo-Datenbank, die alle Schriftarten und alle Größen aufsteigend nach Zeilenhöhe und -breite ausgeben. Filtern Sie das Ergebnis etwa nach allen Schriften, die in einer Zeilenhöhe von 12 Pixel resultieren, so finden Sie Schriften in der Spanne von 7 bis 11 Punkt vor! Filtern Sie nach der festen Breite von 1099 Pixel, so stellen sich gar Fonts in der Größe zwischen 7 und 13 Punkt ein. Es gibt also extreme Unterschiede in der Laufweite der einzelnen Fonts.

Noch aussagekräftiger wäre eigentlich die Eigenschaft **Area**, weil diese Auskunft darüber gibt, wie viele Zeichen insgesamt in das Rechteck eines Code-Fensters passen. Die Abfrage **qry\_Area** gibt das aus, wobei hier zur leichteren Filterung die Werte für Area gerundet gruppiert sind. Filtern Sie diese Abfrage in der Datenblattansicht etwa nach 10 Punkt, so bekommen Sie für die Schrift **ProggyTinyTT** eine Zeilenfläche von 6200 Pixeln heraus, für **Source Code**

**Pro ExtraLight** eine von 21200 Pixeln. Das bedeutet, dass sich im Code-Fenster mit der ersten Schrift über dreimal mehr Zeichen darstellen lassen, wie mit der zweiten! Aus der Spalte **LineHeight** der Abfrage geht allerdings hervor, dass die erste Schriftart das vornehmlich durch eine sehr geringe Höhe von 9 Pixeln erreicht, während die zweite eine von 17 aufweist (siehe Bild 5). Beide lassen sich demzufolge nicht direkt miteinander vergleichen.

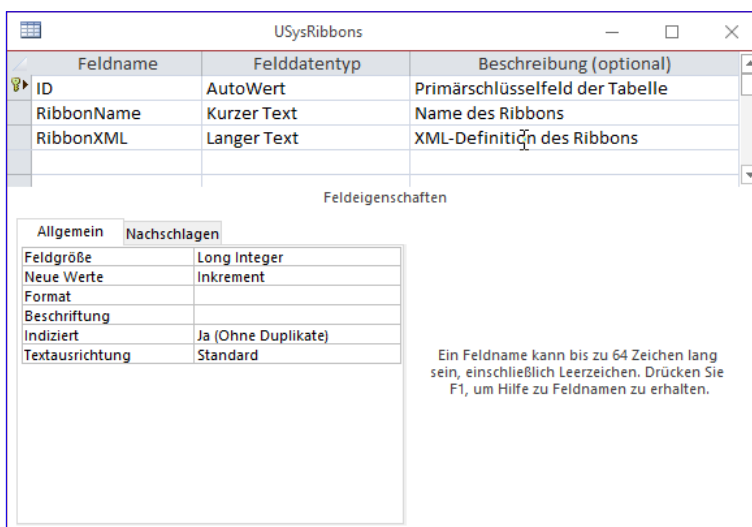
Natürlich können Sie die Abfrage weiter filtern und etwa nur Zeilenhöhen von 12 Pixeln zulassen, sowie Font-Größen von 9 bis 11. Damit wären die Schriften tatsächlich vergleichbar. Mit **Terminal** bekommen Sie dann eine

## Ribbon-Elemente aktivieren und deaktivieren

Im Beitrag »Software freischalten per Schlüssel« zeigen wir, wie Sie mit einem Freischaltsschlüssel eine Information übergeben können, welche Funktionen einer Datenbank für den Benutzer mit dem Schlüssel freigegeben sind und wie Sie etwa in einem Formular die Schaltflächen aktivieren oder deaktivieren, die diese Funktionen aufrufen. Im vorliegenden Beitrag wollen wir uns anschauen, wie Sie die Schaltflächen eines Ribbons abhängig von den mit dem Freischaltsschlüssel übergebenen Daten aktivieren oder deaktivieren können.

In dem genannten Beitrag **Software freischalten per Schlüssel** ([www.access-im-unternehmen.de/1126](http://www.access-im-unternehmen.de/1126)) gibt es eine Funktion, welche einen Wert für die mit einem Freischaltsschlüssel freigeschalteten Features des Programms mitliefert. Diese bietet einen Wert von **0** bis **7**, wobei diese Zahl in eine Binärzahl umzuwandeln ist und dann über die einzelnen Positionen angibt, welche der Features freigeschaltet sind.

Der Wert **7** würde also beispielsweise bedeuten, dass alle Features funktionieren sollen – **7** entspricht binär **111**. Der Wert **4** würde heißen, dass nur die Funktion aktiviert ist, welche der ersten Position von links entspricht, der Wert **1**, dass nur die Funktion auf der dritten Position laufen soll.



Feldname	Felddatentyp	Beschreibung (optional)
ID	AutoWert	Primärschlüsselfeld der Tabelle
RibbonName	Kurzer Text	Name des Ribbons
RibbonXML	Langer Text	XML-Definition des Ribbons

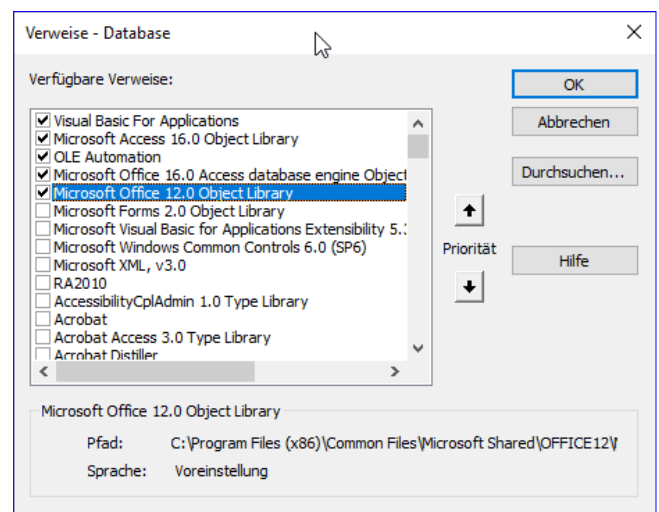
**Feldigenschaften**

Allgemein | Nachschlagen

Feldgröße	Long Integer
Neue Werte	Inkrement
Format	
Beschriftung	
Indiziert	Ja (Ohne Duplikate)
Textausrichtung	Standard

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

**Bild 2:** Definition der Tabelle **USysRibbons**



**Bild 1:** Verweis auf die Office-Bibliothek

Bevor wir uns um diesen Wert und seine Interpretation für die Steuerelemente des Ribbons kümmern, wollen wir aber erst einmal das Ribbon hinzufügen.

### Verweis auf die Office-Bibliothek

Für die Programmierung des Ribbons benötigen wir zunächst einen Verweis auf die Office-Bibliothek, den wir über den **Verweise**-Dialog hinzufügen. Diesen öffnen Sie im VBA-Editor (**Alt + F11**) über den Menüeintrag **Extras\Verweise**. Im Dialog setzen Sie einen Haken für den Eintrag **Microsoft Office x.0 Object Library**, wobei x für die bei Ihnen verfügbare Version steht (siehe Bild 1).

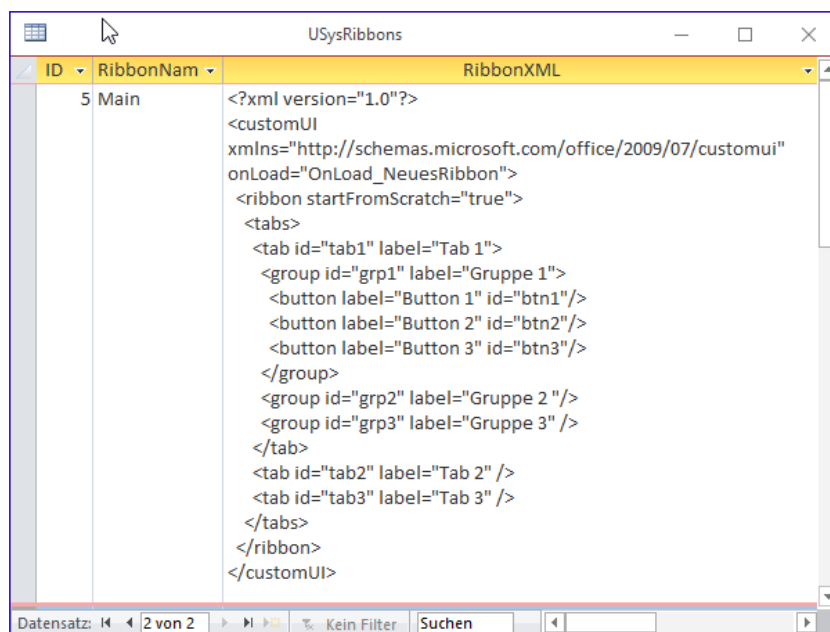
```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_NeuesRibbon">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tab1" label="Tab 1">
        <group id="grp1" label="Gruppe 1">
          <button label="Button 1" id="btn1"/>
          <button label="Button 2" id="btn2"/>
          <button label="Button 3" id="btn3"/>
        </group>
        <group id="grp2" label="Gruppe 2" getVisible="getVisible">
          <button label="Button 1" id="btn21"/>
        </group>
        <group id="grp3" label="Gruppe 3" getVisible="getVisible">
          <button label="Button 1" id="btn31"/>
        </group>
      </tab>
      <tab id="tab2" label="Tab 2" />
      <tab id="tab3" label="Tab 3" />
    </tabs>
  </ribbon>
</customUI>
```

**Listing 1:** Definition des Ribbons mit drei Schaltflächen, drei Tabs und drei Gruppen

### Tabelle zum Speichern der Ribbon-Definition

Danach fügen Sie eine Tabelle namens **USysRibbons**

zur Datenbank hinzu, welche die Ribbon-Definitionen speichern soll. Diese sieht in der Entwurfsansicht wie in Bild 2 aus.



**Bild 3:** Die Tabelle **USysRibbons** mit der Ribbon-Definition

Wegen Ihres Namens, der **USys...** beginnt, wird die Tabelle normalerweise nicht im Navigationsbereich von Access angezeigt. Sie können diese jedoch einblenden, wenn Sie in den Optionen des Navigationsbereichs, den Sie durch einen Klick mit der rechten Maustaste auf den Titel des Navigationsbereichs und Auswahl des Eintrags **Navigationsoptionen...** aus dem Kontextmenü den Eintrag **Systemobjekte anzeigen** aktivieren.

### Ribbon definieren

Anschließend können wir das Ribbon definieren. Dieses sieht im ersten Entwurf

wie in Listing 1 aus. Den dortigen Code tragen Sie in das Feld **RibbonXML** der Tabelle **USysRibbons** ein. Das Feld **Ribbonname** erhält den Wert **Main**. Die Tabelle sieht dann im Entwurf wie in Bild 3 aus.

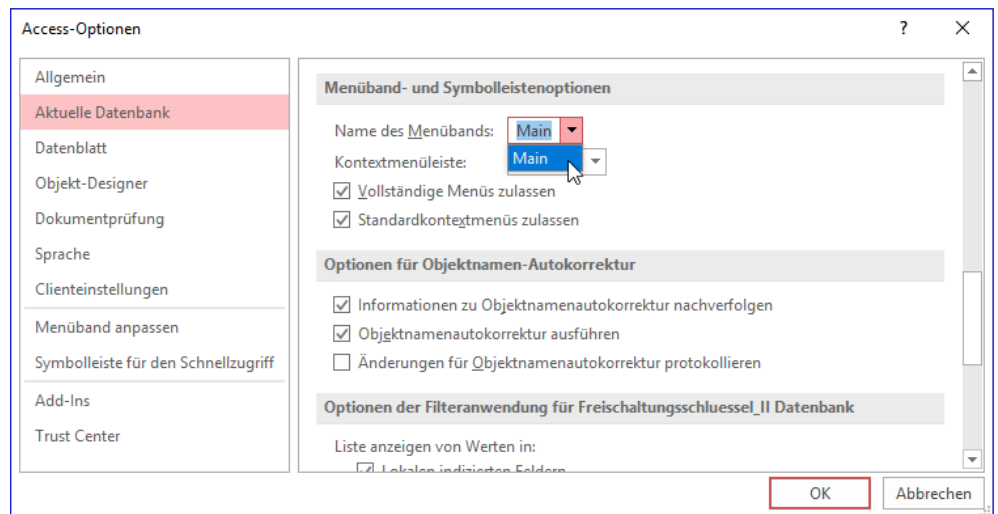
Dadurch, dass wir die Eigenschaft **start-FromScratch** des Elements **ribbon** auf den Wert **true** eingestellt haben, werden alle eingebauten Elemente des Ribbons ausgeblendet und nur noch die benutzerdefinierten Elemente angezeigt.

Damit das Ribbon nun auch noch in der Anwendung erscheint, sind noch weitere Schritte nötig. Als Erstes müssen Sie die Anwendung schließen und neu starten. Erst dann ist die neue Ribbon-Definition auch in der Auswahlliste der Ribbons für diese Anwendung enthalten. Die Auswahlliste finden Sie, wenn Sie mit einem Klick auf den **Datei**-Reiter des Ribbons den Backstage-Bereich öffnen und dann auf den Eintrag **Optionen** klicken. Im nun erscheinenden Dialog **Access-Optionen** wählen Sie links den Eintrag **Aktuelle Datenbank** aus. Rechts finden Sie dann unter **Menüband- und Symbolleistenoptionen** die Eigenschaft **Name des Menübands**, wo Sie unser soeben erstelltes Ribbon namens **Main** auswählen (siehe Bild 4).

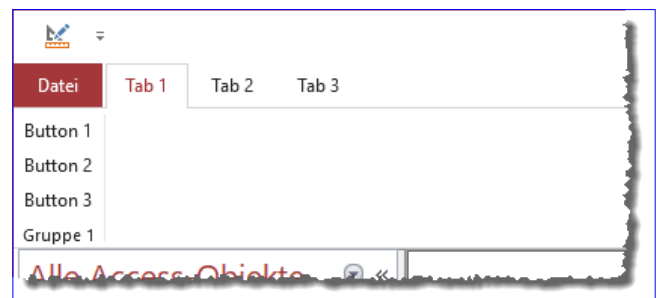
Wenn Sie danach die Anwendung nochmals schließen und wieder öffnen, erscheint unser Ribbon und sieht wie in Bild 5 aus.

### Callback-Attribute anlegen

Nun sind allerdings noch keine Elemente deaktiviert geschweige denn hätten wir überhaupt irgendwelche Maßnahmen ergriffen, um dies zu bewerkstelligen. Wie aber wollen wir dafür sorgen, dass die Schaltflächen in



**Bild 4:** Auswahl des Ribbons in den Einstellungen der Datenbank



**Bild 5:** Benutzerdefiniertes Ribbon

Abhängigkeit von den freigeschalteten Funktionen aktiviert oder deaktiviert werden? Dazu benötigen wir zunächst einige Callback-Attribute für die einzelnen Elemente sowie die VBA-Prozeduren, welche die passenden Werte zum Aktivieren oder Deaktivieren zurückliefern.

Genau genommen können wir nur die Button-Elemente aktivieren oder deaktivieren, da nur diese über eine **enabled**-Eigenschaft verfügen und eine entsprechende **getEnabled**-Eigenschaft.

Wir wollen aber auch anhand der **tab**- und **group**-Elemente zeigen, wie sich diese in Abhängigkeit vom Funktionsumfang beeinflussen lassen. Es fehlt diesen Elementen zwar an einer **enabled**-Eigenschaft, dafür stellen Sie aber ein **visible**-Attribut und ein entsprechendes **getVisible**-Callback-Attribut zur Verfügung.

## Dateieigenschaften ermitteln

In Access-Datenbanken werden ja nicht immer nur Personen, Objekte und Vorgänge verwaltet. Nicht selten lesen Sie auch Dateien und deren Eigenschaften in Tabellen ein. Denken Sie etwa an die Informationen, die Sie Audio- oder Videodateien für eine Medienverwaltung abringen möchten. Mit dem Rüstzeug von VBA allein kommen Sie hier nicht weit, wohl aber mit den Shell-Objekten von Windows.

### Dateieigenschaften im Explorer

Der Explorer von Windows zeigt einige Eigenschaften von Dateien in voreingestellten Spalten an, wenn Sie die Detailansicht auswählen. Welche das sind, hängt vom Ordnerinhalt ab, den Windows vermutet. Für allgemeine Dateien sind das neben dem Namen in der Regel noch der **Dateityp**, die **Größe**, sowie **Änderungs-** und **Erstelldatum**. Befinden sich jedoch etwa eine größere Anzahl von **JPEG**-Dateien im Verzeichnis, so wechselt der Explorer das Spalten-Set in mehr oder weniger für Bilder nützliche Informationen.

Mit Rechtsklick auf die Spaltenköpfe erscheint ein Popup-Menü, über das Sie andere gebräuchliche Spalten an- oder abwählen können (siehe Bild 1).

Benötigen Sie andere Informationen, so erscheint über **Weitere...** der Dialog in Bild 2, in dem Sie aus einer umfangreichen Liste die gewünschten Einträge markieren, wobei viele nur bestimmten Dateitypen vorbehalten sind, weshalb die dann eingblendeten Spalten im Zweifel leer bleiben. Das eingestellte Spalten-Set speichert die **Windows-Shell** pro Ordner persistent, sobald Sie ihn verlassen.

Klicken Sie auf eine Datei rechts und wählen im Kontextmenü den Eintrag **Eigenschaften**, so bekommen Sie unter dem Reiter **Details** allerlei Informationen präsentiert, die sich je nach Dateityp in Inhalt und Umfang unterscheiden (siehe Bild 3).

Der Name einer Eigenschaft ist dabei jeweils identisch zu jenem, der auch im anderen Dialog oder in den Spaltenköpfen angezeigt wird. Das Gleiche gilt für die

Werte. Einige Eigenschaften sind hier sogar editierbar, wenn sich das Wertefeld beim Markieren in eine Textbox verwandelt.

Wir haben es hier überall mit dem sogenannten **Property System** von

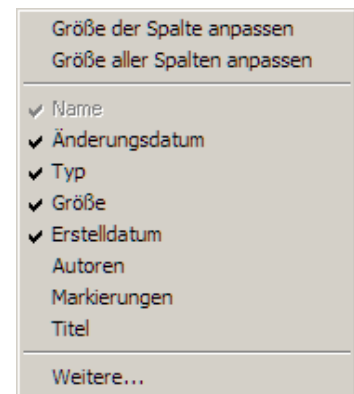


Bild 1: Popup des Explorer-Spaltenkopfs

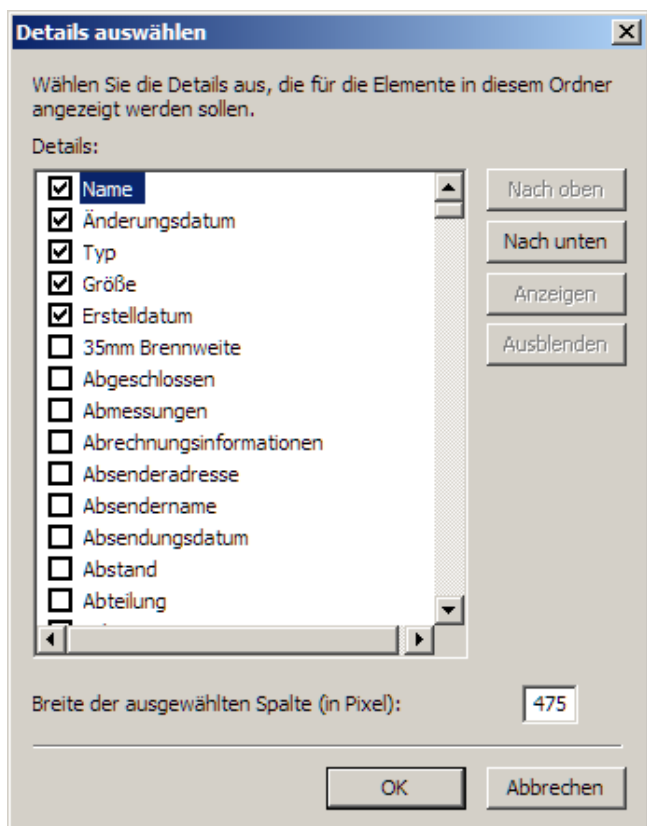


Bild 2: Liste der Dateieigenschaften im Details-Dialog des Explorer für Ordner

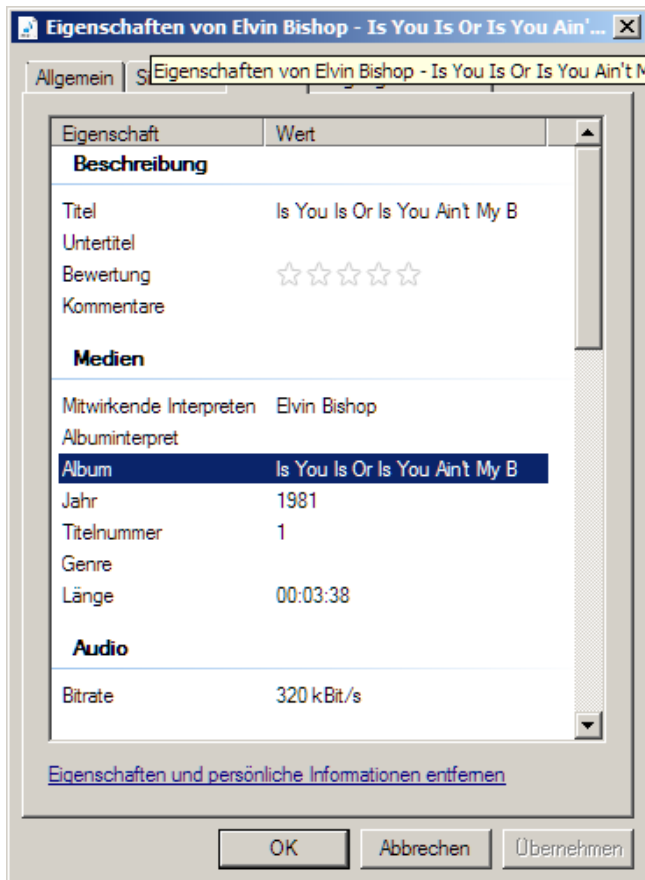


Bild 3: Informationen zu einer MP3-Datei im Eigenschaftendialog des Explorers

Windows zu tun, das in die **Shell** integriert ist. Und auf dieses können Sie mithilfe einiger Objekte auch in Ihrem VBA-Projekt zugreifen! Der Aufwand dafür hält sich in Grenzen. Es werden im Prinzip nämlich keine jener **API**-Funktionen benötigt, die man sonst häufig in entsprechenden Routinen zum Auslesen von Dateieigenschaften im Netz findet.

### Das Windows Property System

Es wurde in der jetzigen Form mit **Windows Vista** eingeführt und dient der Abstraktion von Dateieigenschaften im System. Über die Ansprache einer einzigen Komponente (**propsys.dll**) können alle verfügbaren Informationen abgerufen werden. Diese Komponente bemüht ihrerseits andere Komponenten, die die eigentliche Arbeit verrichten, weil nur sie den Aufbau bestimmter Dateitypen kennen. Und außerdem kann das **Property System**

noch durch installierte Programme erweitert werden. **Office** installiert etwa zusätzliche solcher **Property Handler**, damit Sie direkt auf Dokumenteigenschaften zugreifen können, ohne eines der Programme starten zu müssen. Dafür gibt es eine festgelegte Schnittstelle. Welche Dateitypen mit welchen Komponenten korrelieren, wird dabei in der **Registry** festgehalten.

### Datenbank für Dateieigenschaften

Die Demo zu diesem Beitrag (**FileProps.accdb**) enthält alles Nötige für den Umgang mit diesen Dateieigenschaften. Sie können Teile aus ihr problemlos in eigene Projekte übertragen.

Grundvoraussetzung ist zunächst ein **Verweis** auf die Bibliothek **Microsoft Shell Controls And Automation** im VBA-Projekt, die sich dann im Objektkatalog als **Shell32** zeigt. Die Bibliothek bringt Windows immer mit und kann deshalb ganz ohne Gewissensbisse eingesetzt werden.

Die Prozedur **SHGetPropertyString** im Modul **mdl-PropsShell** macht sich dann einige Klassenobjekte der Bibliothek zunutze. Mit gerade einmal 20 Zeilen Code erledigt sie das Auslesen beliebiger Datei- und Ordneigenschaften, die von der Shell unterstützt werden – und das sind eine ganze Menge! Listing 1 zeigt die komplette Routine. Sie übergeben der Funktion im ersten Parameter den vollen Pfad der Datei und im zweiten die gewünschte Eigenschaft. Zurück erhalten Sie dann grundsätzlich den Wert der Eigenschaft als String. Dies wäre ein Beispiel:

```
? SHGetPropertyString("d:\mp3s\fullalbum.mp3", "System.
Size")
-> "62356466"
```

Hier wird die Dateigröße der MP3-Datei ermittelt. Setzen Sie als zweiten Parameter **System.DateCreated** ein, so erhalten Sie das Erstelldatum:

```
-> "02.11.2017 16:23:48"
```



Oder mit **System.FileOwner** den Besitzer der Datei:

-> "MeinComputername\Herbert"

Was es mit diesen **Property**-Parametern auf sich hat und wie man auf sie kommt, erläutern wir später noch. Hier folgt aber zunächst die Funktionsweise des Codes.

Das zentrale Objekt, von dem sich alles Weitere ableitet, steht in der Objektvariablen **oSH** vom Typ **Shell**, die im Modulkopf global deklariert ist. Dieses Objekt kann per **New** erzeugt werden. Eine Alternative, die ganz ohne Bibliotheksverweis auskommt, wäre

```
Set oSH = CreateObject("Shell.Application")
```

Bevor das Objekt benutzt wird, müssen aber aus dem vollen Pfad in **sFile** erst der Ordner- und der Dateinamenanteil extrahiert werden, was die folgenden zwei Zeilen erledigen. **sFld** enthält danach den Ordner, **sFil** den Dateinamen. Um beim vorigen Beispiel zu bleiben:

```
sFld -> "d:\mp3s"  
sFil -> "fullalbum.mp3"
```

Die **NameSpace**-Funktion des **Shell**-Objekts gibt nun ein neues Objekt vom Typ **Folder** in **oFld** zurück, wenn man den Pfad übergibt, der hier zusätzlich mit **CStr** behandelt wird. Manchmal funktioniert die Übergabe eines VBA-Strings hier nämlich nicht korrekt und **CStr** stellt sicher, dass die Funktion einen gültigen COM-String erhält.

Der Clou an der Routine ist, dass Sie ihr nicht nur Dateien, sondern auch Ordner zur Inspektion verabreichen können. In diesem Fall ist die Variable **sFil** dann leer

```
Dim oSH As Shell32.Shell  
  
Function SHGetPropertyString(sFile As String, sProp As String) As String  
    Dim sFld As String, sFil As String  
    Dim oFld As Shell32.Folder3  
    Dim oItm As Shell32.ShellFolderItem  
    Dim vRet As Variant  
  
    If oSH Is Nothing Then Set oSH = New Shell32.Shell  
  
    sFld = Left(sFile, InStrRev(sFile, "\") - 1)  
    sFil = Mid(sFile, InStrRev(sFile, "\") + 1)  
    Set oFld = oSH.Namespace(CStr(sFld))  
    If Len(sFil) = 0 Then  
        Set oItm = oFld.Self  
    Else  
        Set oItm = oFld.Items.Item(CStr(sFil))  
    End If  
    vRet = oItm.ExtendedProperty(sProp)  
    If IsArray(vRet) Then  
        SHGetPropertyString = Join(vRet, ";")  
    Else  
        SHGetPropertyString = vRet  
    End If  
End Function
```

**Listing 1:** Die Funktion gibt eine beliebige Dateieigenschaft zurück

und die per **Len()** ermittelte Länge **0**. Davon hängt in der Bedingung ab, wie weiter verfahren wird. Denn als nächstes wird die **Items**-Auflistung des Folder-Objekts bemüht. Das ist im Prinzip das, was der Explorer auf der rechten Seite ausgibt. Ein **Item** kann eine Datei im Verzeichnis sein, oder ein Unterordner. Da der Explorer auch virtuelle Objekte darstellen kann, wie etwa die Systemsteuerung, könnte ein **Item** auch ein virtuelles Objekt widerspiegeln. Selbst dessen Eigenschaften könnten mit der Routine ausgelesen werden, doch das steht hier im Beitrag gerade nicht zur Debatte.

Das **Item** erhält die Objektvariable **oItm** vom Typ **ShellFolderItem**. Für eine Datei übergeben Sie als Index der Auflistung **Items** den Namen in **sFil**. Soll der Ordner selbst inspiziert werden, so kann er selbst in ein **Item** überführt werden, wenn die Methode **Self** auf den

**Folder** angewandt wird. Und nun sind wir soweit: Vom **Item** kann die Funktion **ExtendedProperty** abgerufen werden, der als Parameter der Systemname der Eigenschaft (**sProp**) übergeben wird. Das Ergebnis steht nun im Variant **vRet**.

Allerdings gibt diese Methode nicht immer nur einen String zurück, sondern manchmal auch ein **String-Array**. Ein Word-Dokument kann etwa mehrere Autoren aufweisen, die über den **Property**-Namen **System.Author** ermittelt würden. Die einzelnen Autoren stünden dann als Elemente eines Arrays im Ergebnis. Die VBA-Funktion **IsArray** erkennt, ob das Ergebnis ein Array ist. Dann fügt die **Join**-Funktion die Elemente des Arrays zu einem einzelnen String zusammen.

### System Property Names

Was Sie nun noch brauchen, um sämtliche Eigenschaften einer Datei auslesen zu können, sind die Systemnamen der Eigenschaften. Leider gibt es dafür keine generelle Liste. Denn erstens hat jede Windows-Version ihre eigenen abweichenden Systemeigenschaften, und zweitens lässt sich, wie erwähnt, das **Property System** ja durch neu installierte Programme erweitern, was zu neuen in der Registry eingetragenen **Property Names** führt.

Leider ist man mit der **Shell32**-Bibliothek hier am Ende der Fahnenstange angekommen. Sie enthält keine Methoden, um an die Systemnamen zu kommen. Gottlob gibt es aber die in früheren Beiträgen bereits erwähnte **Type Library oleexp**. Vorgestellt wurde sie etwa im Beitrag **Explorer Control** (Shortlink 1096). Sie stellt, so Sie sie in die Verweise aufnehmen, alle Klassen und In-

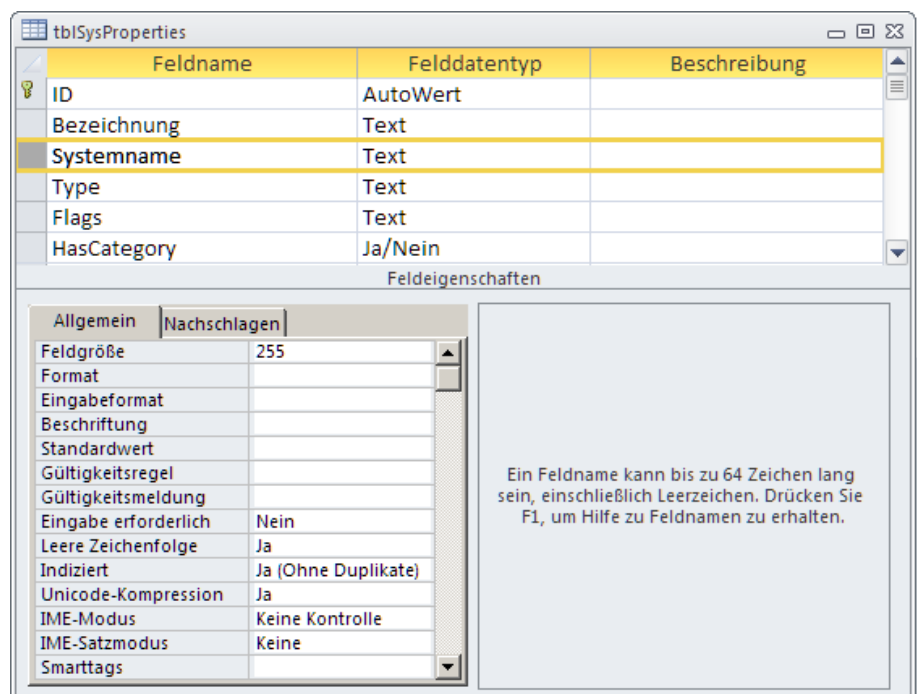
**terfaces** bereit, die für das Auslesen der Systemnamen benötigt werden.

Die Bibliothek wird übrigens von der Demo beim Start des **Intro**-Formulars **frmIntro** automatisch geladen und sollte sich dazu im Verzeichnis der Datenbank befinden:

```
On Error Resume Next
References.AddFromFile CurrentProject.Path & "\olexp.tlb"
```

Die Prozedur **StorePropertyKeys** im gesonderten Modul **mdlPropStore** ermittelt dann alle Systemnamen und speichert sie in der Tabelle **tblSysProperties**. Um es klarzustellen: Dieses Modul brauchen Sie in Ihrer produktiven Datenbank später nicht mehr und eben so wenig den Verweis auf die Bibliothek **olexp**!

Es dient hier ausschließlich dazu, um die Tabelle zu füllen, die dann die gefragten Systemnamen enthält. Nur diese brauchen Sie schließlich in Ihre Datenbank zu übernehmen.



**Bild 4:** In der Entwurfsansicht der Tabelle **tblSysProperties** ist das Feld **Systemname** eindeutig indiziert

Der Aufbau der Tabelle ergibt sich aus Bild 4. Das Feld Systemname erhält den **Property**-Namen, wie etwa **System.Author**. In **Bezeichnung** steht dann die deutsche Übersetzung, wie **Autoren**. Alle anderen Felder sind optional und nur informativ. In **Type** steht der Datentyp, wie etwa **String** oder **Integer**.

Da die Methode **ExtendedProperty** immer Strings zurückgibt, kann man nicht erkennen, um welche Art von Wert es sich handelt. **Type** gibt einen Rückschluss darauf. **Flags** wiederum ist ein Feld, das Hexadezimal-Strings ausgibt, welche weitere Informationen über die Art der Eigenschaft hervorbringen.

Es führt hier zu weit, dies zu erläutern und ich verweise auf den auskommentierten Block **Property Type Flags** im Kopf des Moduls **mdlPropStore**.

**HasCategory** schließlich gibt Auskunft darüber, ob es sich bei der Eigenschaft um ein Element einer bestimmten Kategorie handelt. Eigenschaften können nämlich kategorisiert werden. Die Hauptkategorie ist **System**. Das sind alle in Windows verbauten Eigenschaften, wie

beispielsweise **System.FileOwner**. Hier kann es aber Unterkategorien geben. Bei Bildern etwa kommt die Kategorie **System.Image** ins Spiel, die selbst keine Eigenschaft darstellt. **System.Image.HorizontalSize** aber gibt die horizontale Ausdehnung der Bilddatei zurück. Alle Systemnamen, die mehr, als einen Punkt enthalten, sind Teil einer solchen Unterkategorie.

Eigenschaften hingegen, die von installierten Programmen ermittelt werden, haben nicht die Hauptkategorie **System**, sondern eigene. Hier ist auf dem Rechner etwa die **Corel Graphics Suite** installiert, die zusätzliche **Property Handler** implementiert.

Die Hauptkategorie ist hier **Corel** und eine Eigenschaft wäre **Corel.File.CompressionRatio**, die zurückgibt, wie stark die Bilddatei komprimiert ist. Das allerdings funktioniert nur mit den speziellen **Corel**-Dateitypen, wie **.cdr**. Wie sich schließlich der Inhalt der Tabelle in der Datenblattansicht darstellt, zeigt Bild 5.

Nur der Vollständigkeit halber ist in Listing 2 die Routine abgebildet, welche die Systemnamen enumeriert und

ID	Bezeichnung	Systemname	Type	Flags	HasCategory
29897	Unvollständig	System.IsIncomplete	Boolean	80000298	<input type="checkbox"/>
29898	Lesestatus	System.IsRead	Enum	80000298	<input type="checkbox"/>
29899	Freigegeben	System.IsShared	Boolean	8000009A	<input type="checkbox"/>
29900	Ersteller	System.ItemAuthors	String	8000009B	<input type="checkbox"/>
29901	Datum	System.ItemDate	DateTime	8000009A	<input type="checkbox"/>
29902	Ordnername	System.ItemFolderNameDisplay	String	8000009A	<input type="checkbox"/>
29903	Ordnerpfad	System.ItemFolderPathDisplay	String	8000009A	<input type="checkbox"/>
29904	Ordner	System.ItemFolderPathDisplayNarrow	String	8000009A	<input type="checkbox"/>
29905	Name	System.ItemNameDisplay	String	8000049A	<input type="checkbox"/>
29906	Teilnehmer	System.ItemParticipants	String	8000009B	<input type="checkbox"/>
29907	Pfad	System.ItemPathDisplay	String	8000009A	<input type="checkbox"/>
29908	Pfad	System.ItemPathDisplayNarrow	String	8000001A	<input type="checkbox"/>
29909	Nach Ort	System.ItemSearchLocation	String	8000009A	<input type="checkbox"/>
29910	Elementtyp	System.ItemType	String	8000049A	<input type="checkbox"/>
29911	Typ	System.ItemTypeText	String	8000049A	<input type="checkbox"/>
29912	Kontaktnamen	System.Journal.Contacts	String	80000299	<input checked="" type="checkbox"/>
29913	Eintragstyp	System.Journal.EntryType	String	80000298	<input checked="" type="checkbox"/>
29914	Markierungen	System.Keywords	String	800006B9	<input type="checkbox"/>
29915	Art	System.Kind	Enum	8000009B	<input type="checkbox"/>
29926	Sprache	System.Language	String	80000298	<input type="checkbox"/>

Bild 5: Ausschnitt der Datenblattansicht der Tabelle **tblSysProperties** mit allen für das System ermittelten **Shell-Properties**

```
Public Sub StorePropertyKeys(Optional What As oleexp.PROPDESC_ENUMFILTER, _
    Optional DeleteExisting As Boolean = True)
    Dim i As Long
    Dim IPSys As IPropertySystem
    Dim IPropDesc As IPropertyDescription
    Dim IPList As IPropertyDescriptionList
    Dim nProp As Long, lpProp As Long, lType As Long, lFlag As Long
    Dim rs As DAO.Recordset
    Dim cnt As Long
    If DeleteExisting Then CurrentDb.Execute "DELETE FROM tblSysProperties"
    Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblSysProperties", dbOpenDynaset)
    PSGetPropertySystem String2UID(IID_IPropertySystem), IPSys
    On Error Resume Next
    IPSys.EnumeratePropertyDescriptions What, String2UID(IID_IPropertyDescriptionList), IPList
    IPList.GetCount nProp
    For i = 0 To (nProp - 1)
        IPList.GetAt i, String2UID(IID_IPropertyDescription), IPropDesc
        IPropDesc.GetDisplayName lpProp
        If lpProp <> 0 Then
            rs.AddNew
            rs!Bezeichnung = SysAllocString(lpProp) & ""
            CoTaskMemFree lpProp
            IPropDesc.GetCanonicalName lpProp
            IPropDesc.GetDisplayType lType
            IPropDesc.GetTypeFlags PDF_MASK_ALL, lFlag
            rs!Systemname = SysAllocString(lpProp)
            CoTaskMemFree lpProp
            rs!Type = Choose(1 + lType, "String", "Number", "Boolean", "DateTime", "Enum")
            rs!Flags = Hex(lFlag)
            rs!HasCategory = UBound(Split(rs!Systemname.Value, ".")) > 1
            Err.Clear
            rs.Update
            If Err.Number <> 0 Then rs.CancelUpdate Else cnt = cnt + 1
        End If
    Next i
    rs.Close
End Sub

Private Function String2UID(sGUID As String) As oleexp.UUID
    oleexp.CLSIDFromString sGUID, String2UID
End Function
```

**Listing 2:** Diese Prozedur enumeriert alle auf dem System verfügbaren Systemnamen und speichert sie

in die Tabelle speichert. Es liegt deutlich außerhalb des Horizonts dieses Beitrags, ihre Funktionsweise zu erläutern, denn hier kommen recht exotische **Interfaces** und

neuen Datensatzes (**rs.AddNew**) zugewiesen. Damit ist die Tabelle mit allen registrierten Eigenschaftennamen gefüllt.

Methoden zum Einsatz. Es sollte reichen, wenn Sie wissen, **was** sie tut.

Sie löscht zunächst optional (Parameter **DeleteExisting**) alle Einträge in der Tabelle **tblSysProperties** und öffnet dann ein Recordset **rs** auf sie, um sie erneut mit Datensätzen zu füllen. Mit **PSGetPropertySystem** wird folgend ein genereller Verweis auf das **Property System** von Windows erhalten, was anhand einer speziellen **Interface-GUID (IID\_IPropertySystem)** geschieht. **EnumeratePropertyDescriptions** ist dann auch schon die Methode, welche die Liste der Eigenschaftennamen öffnet.

Die Zahl derer ermittelt **GetCount** und eine Schleife arbeitet nun alle Einträge ab. Verschiedene Methoden berechnen den lokalen Anzeigenamen (**Bezeichnung**), den Systemnamen (**GetCanonicalName**), den Datentyp (**GetDisplayType**) und die Flags (**GetTypeFlags**). Die Ergebnisse werden jeweils den Feldern eines

## Outlook-Absender einstellen

E-Mails mit Outlook zu verschicken ist von Access aus mittlerweile relativ einfach. Die Übergabe von Empfänger, Cc: oder Bcc:, dem Betreff und dem Inhalt klappt reibungslos, und man kann die Mail entweder direkt versenden lassen oder auch erst noch öffnen und dann per Mausklick versenden. Was nicht trivial ist, ist jedoch das Einstellen der Absenderadresse. Die lässt sich nicht einfach einstellen, sondern muss mit einem kleinen Trick festgelegt werden. Dieser Beitrag zeigt, wie Sie Outlook-E-Mails mit dem gewünschten Absender auf den Weg bringen.

### Voraussetzungen

Bevor wir mit der Programmierung starten, fügen Sie dem VBA-Projekt der Zieldatenbank einen Verweis auf die entsprechende Version der Outlook-Bibliothek hinzu. Dazu öffnen Sie etwa mit der Tastenkombination **Alt + F11** den VBA-Editor und wählen dann aus der Menüleiste den Eintrag **Extras|Verweise** aus. Im nun erscheinenden Dialog fügen Sie den Eintrag **Microsoft Outlook x.0 Object Library** hinzu, wobei **x** der aktuellen Office-Version entspricht.

### Mail mit Standardadresse senden

Der einfachste Code, um eine E-Mail-Adresse zu versenden, sieht wie folgt aus:

```
Public Sub MailMitStandardadresse()  
    Dim objOutlook As Outlook.Application  
    Dim objMail As Outlook.MailItem  
    Set objOutlook = New Outlook.Application  
    Set objMail = objOutlook.CreateItem(olMailItem)  
    With objMail  
        .Subject = "Testmail"  
        .Body = "Dies ist eine Testmail."  
        .To = "info@access-im-unternehmen.de"  
        .Send  
    End With  
End Sub
```

Damit wird die Mail mit der standardmäßig in Outlook eingestellten E-Mail-Adresse verschickt. Aber welche ist die Standardadresse? Diese finden wir etwa in Outlook 2016

heraus, wenn wir im Ribbon auf den Reiter **Datei** klicken und dort die Schaltfläche **Kontoeinstellungen|Kontoeinstellungen** betätigen (siehe Bild 1).

Es erscheint dann der Dialog **Kontoeinstellungen** (siehe Bild 2). Die standardmäßig verwendete Absenderadresse ist hier mit einem entsprechenden Symbol markiert. Um eine andere Adresse zu nutzen, markieren Sie einen der anderen Einträge und betätigen die Schaltfläche **Als Standard festlegen**.

### Absender von Access aus verwalten

Nun wissen wir, dass Access/VBA beim automatisierten Versenden von E-Mails immer die Standardadresse

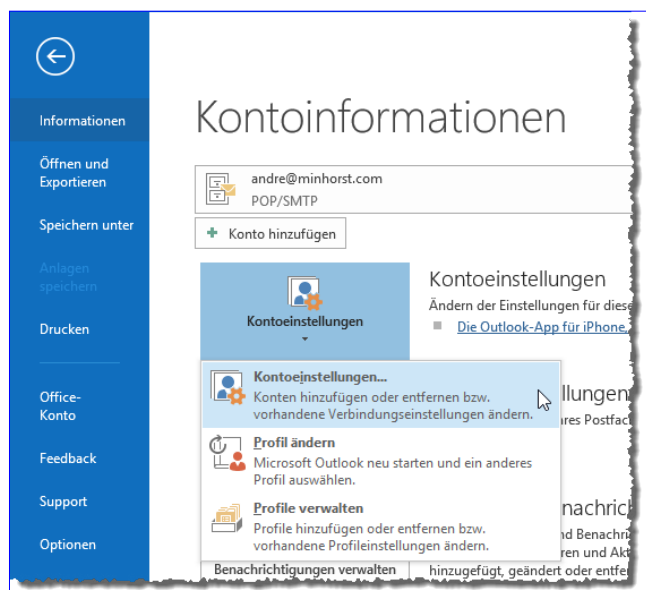


Bild 1: Kontoinformationen öffnen

## Daten in Properties verstecken

Daten schreibt man in einer Datenbank üblicherweise in die Datensätze der enthaltenen Tabellen. Manchmal möchten Sie aber vielleicht wichtige Informationen, die nicht zu den vom Benutzer verwendeten Daten gehören, versteckt unterbringen. Ein gutes Beispiel dafür ist das Datum der ersten Anwendung einer Datenbank, für die der Benutzer nur über einen bestimmten Zeitraum benutzen darf. Dieser Artikel zeigt, wo Sie solche Daten unterbringen können und wie Sie diese wieder abrufen.

### Datenbank-Properties

Die erste Möglichkeit, Daten vor herkömmlichen Benutzern zu verstecken, ist eine einfache Datenbank-Property, also eine Datenbank-Eigenschaft. Diese erreichen Sie über die Auflistung **Properties** des **Database**-Objekts. Wenn Sie alle Datenbank-Eigenschaften der aktuellen Datenbank ausgeben möchten, nutzen Sie dazu die Auflistung **Properties** des mit **CurrentDb** ermittelten **Database**-Objekts. Diese können Sie beispielsweise mit der folgenden Prozedur durchlaufen und im Direktbereich ausgeben lassen:

```
Public Sub GetAllProperties()
```

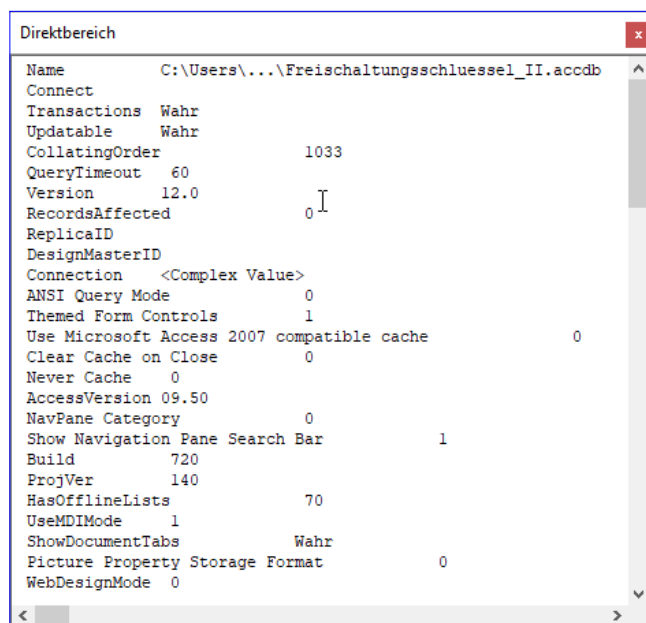


Bild 1: Ausgabe der Properties einer Datenbank

```
Dim prp As DAO.Property
Dim db As DAO.Database
Set db = CodeDb
For Each prp In db.Properties
    Debug.Print prp.Name,
    On Error Resume Next
    Debug.Print prp.Value
    If Not Err.Number = 0 Then
        Debug.Print "<Error>"
    End If
    On Error GoTo 0
Next prp
End Sub
```

Gegebenenfalls tritt bei manchen Eigenschaften ein Fehler auf, den wir hier durch die Ausgabe von **<Error>** umgehen. Das Ergebnis sieht beispielsweise wie in Bild 1 aus.

### Benutzerdefinierte Eigenschaft setzen

Wenn Sie eine eigene Eigenschaft setzen wollen, können Sie die folgende Prozedur nutzen. Diese erwartet als Parameter den Namen der zu setzenden/zu erzeugenden Eigenschaft sowie den Wert, den diese Eigenschaft annehmen soll. Wie Sie dem vorherigen Satz entnehmen können, prüft diese Prozedur indirekt, ob die Eigenschaft bereits vorhanden ist und legt diese nur an, falls dies nicht der Fall ist.

Die Prozedur deklariert ein **Database**- und ein **Property**-Element und füllt das **Database**-Objekt mit der Funktion **CurrentDb**. Danach versucht sie dem Element der **Pro**

**properties**-Auflistung des aktuellen **Database**-Objekts mit dem Namen aus **strProperty** den Wert **strValue** zuzuweisen. Ist diese Eigenschaft bereits vorhanden, gelingt dies ohne Probleme. Anderenfalls löst dies den Fehler mit der Nummer **3270** aus. Da wir für den Fall eines Fehlers einen Sprung zu der Marke **Fehler** vorgesehen haben, springt die Prozedur zu dieser Stelle und prüft, ob der ausgelöste Fehler die Fehlernummer **3270** aufweist.

Falls ja, steht fest, dass die gewünschte Eigenschaft schlicht noch nicht vorhanden ist und wir diese zunächst anlegen müssen. Das erledigen wir mit zwei Anweisungen. Die erste legt die neue Eigenschaft mit der **CreateProperty**-Methode an, wobei wir dieser den Namen, den Typ (**dbText**) und den Wert als Parameter übergeben. Anschließend hängen wir die Eigenschaft mit der **Append**-Methode an die Liste der Eigenschaften der Datenbank an (**db.Properties.Append**). Außerdem aktualisieren wir die Liste der Eigenschaften mit der **Refresh**-Methode:

```
Public Sub SetProperty(strProperty As String, _
    strValue As String)
    Dim db As DAO.Database
    Dim prp As DAO.Property
    On Error GoTo Fehler
    Set db = CurrentDb
    db.Properties(strProperty) = strValue
Ende:
    On Error Resume Next
    Exit Function
Fehler:
    If Err.Number = 3270 Then
        Set prp = db.CreateProperty(strProperty, _
            dbText, strValue)
        db.Properties.Append prp
        db.Properties.Refresh
    End If
    Resume Ende
End Sub
```

Die Prozedur können wir auch etwas umformulieren:

```
Public Function SetProperty(strProperty As String, _
    strValue As String)
    Dim db As DAO.Database
    Dim prp As DAO.Property
    Set db = CurrentDb
    On Error Resume Next
    Set prp = db.CreateProperty(strProperty, _
        dbText, strValue)
    db.Properties.Append prp
    db.Properties.Refresh
    If Not Err.Number = 0 Then
        db.Properties(strProperty) = strValue
    End If
End Function
```

In diesem Fall deaktivieren wir zunächst die Fehlerbehandlung und versuchen, die Eigenschaft neu zu erstellen und direkt zu füllen. Gelingt dies nicht, wird ein Fehler ausgelöst. Deshalb prüfen wir gleich im Anschluss, ob ein Fehler ausgelöst wurde. Falls ja, können wir davon ausgehen, dass die Eigenschaft bereits vorhanden ist und fügen den Wert durch einfache Zuweisung an **db.Properties(strProperty)** hinzu.

### Eigenschaft auslesen

Das Auslesen einer speziellen Eigenschaft erfordert von der dafür vorgesehenen Funktion wieder die Angabe eines Parameters, nämlich des Namens der auszugebenden Eigenschaft. Die folgende Funktion ermittelt bei deaktivierter Fehlerbehandlung den Wert der mit dem Parameter **strProperty** angegebenen Eigenschaft. Ist diese nicht vorhanden, liefert die Funktion eine leere Zeichenkette zurück:

```
Public Function GetProperty(strProperty As String) As String
    Dim prp As DAO.Property
    Dim db As DAO.Database
    Set db = CurrentDb
    On Error Resume Next
    Set prp = db.Properties(strProperty)
    On Error GoTo 0
```

## Software freischalten per Schlüssel

Wenn Sie eine Software weitergeben, wollen Sie natürlich sicherstellen, dass diese nur von Kunden genutzt wird, die auch für die Software bezahlt haben. Wie können Sie das sicherstellen? Eine Möglichkeit ist es, in Abhängigkeit von einem Benutzernamen einen Freischaltungsschlüssel zu erzeugen, den der Benutzer in die Anwendung eingeben muss, damit diese funktioniert. Auf diese Weise sorgen Sie nicht nur dafür, dass der Benutzer diesen Schlüssel erwerben muss, wenn er die Anwendung nutzen will. Damit ist gleichzeitig sichergestellt, dass Sie – wenn ein Benutzer die Anwendung weitergibt oder diese sogar in einem Portal zum Download bereitstellt – herausfinden, welcher Kunde die Software weitergegeben hat. Dieser Beitrag liefert die technischen Grundlagen für den Schutz einer Software mit Freischaltungsschlüssel.

### Voraussetzung

Voraussetzung für den Schutz der Software mit einem Freischaltungsschlüssel ist natürlich, dass Sie diese nur als **.mde-** oder **.accde-** Datei weitergeben. Anderenfalls kann der Benutzer ja einfach in den Code schauen und die Schutzmaßnahme aushebeln.

### Startformular

Die Abfrage der Daten für die Freischaltung, also der E-Mail-Adresse und des Freischaltungs-

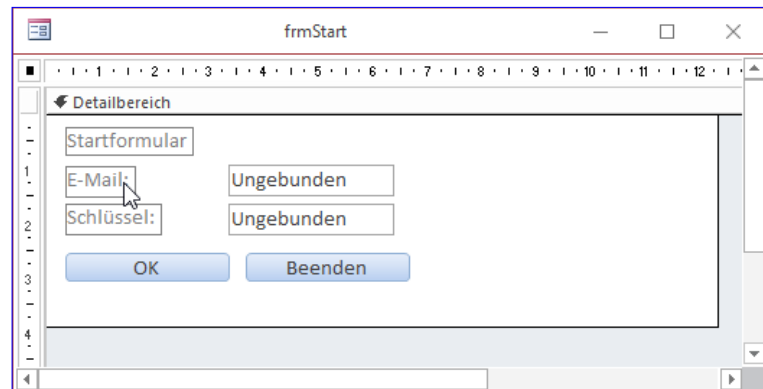


Bild 1: Formular zum Freischalten der Anwendung

Feldname	Felddatentyp	Beschreibung (optional)
ID	AutoWert	Primärschlüsselfeld der Tabelle
E-Mail	Kurzer Text	E-Mail-Adresse für die Freischaltung
Schlüssel	Kurzer Text	Schlüssel für die Freischaltung

Feldeigenschaften	
Allgemein	Nachschlagen
Feldgröße	Long Integer
Neue Werte	Inkrement
Format	
Beschriftung	
Indiziert	Ja (Ohne Duplikate)
Textausrichtung	Standard

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

Bild 2: Tabelle zum Speichern der Freischaltungsdaten

schlüssels, soll in einem Startformular erfolgen. Dieses heißt **frmStart** und sieht wie in Bild 1 aus. Für das Formular stellen wir die Eigenschaften **Bildlaufleisten**, **Navigationsschaltflächen**, **Datensatzmarkierer** und **Trennlinien** auf **Nein** und **Automatisch zentrieren** auf **Ja** ein. Außerdem soll das Formular als **Popup** geöffnet werden, also so, dass der Benutzer keine weiteren Aktionen ausführen kann, wenn dieses Formular nicht geschlossen wurde. Dazu stellen wir die Eigenschaft **Popup** auf **Ja** ein.

Um die vom Benutzer in die beiden Textfelder **txtEMail** und **txtSchluessel** eingegebenen Daten zu speichern, arbeiten wir mit einer Tabelle



namens **tblOptionen**. Diese ist wie in Bild 2 aufgebaut und enthält die drei Felder **ID**, **E-Mail** und **Schlüssel**. Im Auflieferungszustand soll die Tabelle einen leeren Datensatz enthalten.

Damit der Benutzer die Daten über das Formular **frmStart** eingeben kann, binden wir das Formular ausnahmsweise nicht über die Eigenschaft **Datenherkunft** an die Tabelle **tblOptionen**. Wir wollen die eingegebenen Daten ja nur dann speichern, wenn der Benutzer ein passendes Paar von E-Mail und Schlüssel eingegeben hat. In diesem Fall soll das Formular **frmStart** gar nicht mehr erscheinen, sondern direkt das erste Formular der Anwendung angezeigt werden.

### Startformular einrichten

Damit das Formular **frmStart** direkt beim Öffnen der Anwendung angezeigt wird, stellen wir in den Access-Optionen, die Sie über den Eintrag **Optionen** unter dem Ribbon-Tab **Datei** öffnen, im Bereich **Aktuelle Datenbank** unter **Anwendungsoptionen** die Eigenschaft **Formular anzeigen** auf das Formular **frmStart** ein (siehe Bild 3).

Wenn wir die Anwendung nun neu starten, um das Formular **frmStart** direkt beim Start anzuzeigen, gelingt das auf den ersten Blick wie gewünscht. Allerdings kann man trotz des Modus als Popup noch direkt auf die Elemente des Navigationsbereichs oder des Ribbons zugreifen. Um dies zu verhindern, müssen wir auch noch die Eigenschaft **Modal** auf **True** einstellen. Dies gelingt allerdings nicht über das Eigenschaftsfenster, sodass wir die entsprechende Zuweisung in einer Prozedur unterbringen, die durch

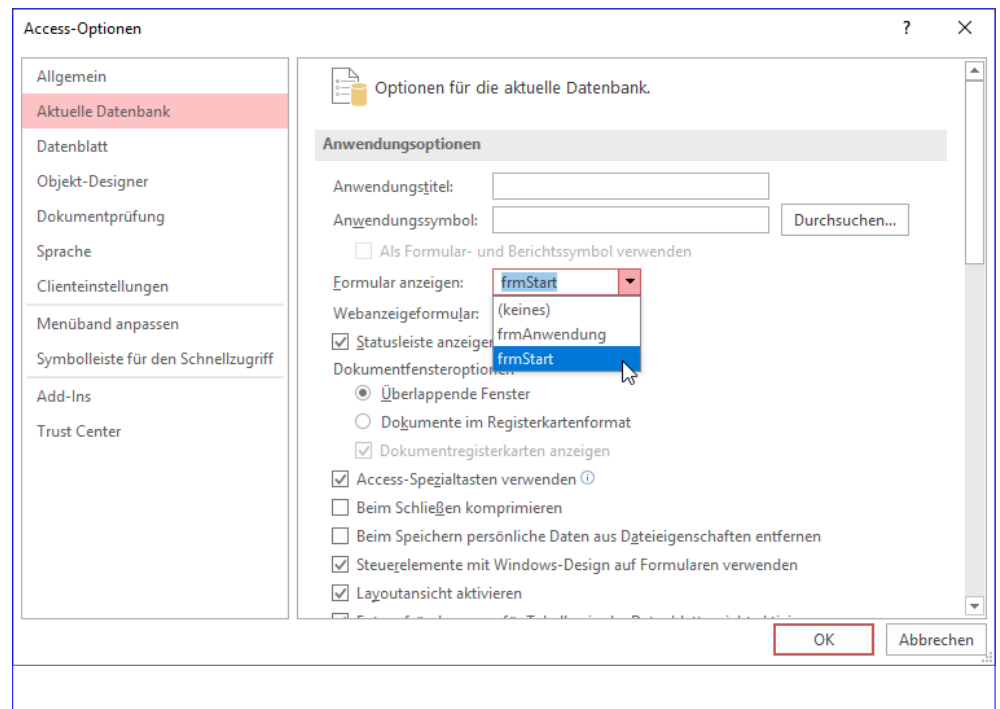


Bild 3: Einstellen des Formulars **frmStart** als Startformular

das Ereignis **Beim Laden** des Formulars ausgelöst wird. Diese Prozedur sieht wie folgt aus:

```
Private Sub Form_Load()  
    Me.Modal = True  
End Sub
```

Nun zeigt das Formular allerdings immer noch die Titelleiste sowie die Schaltflächen zum Schließen, Mini-

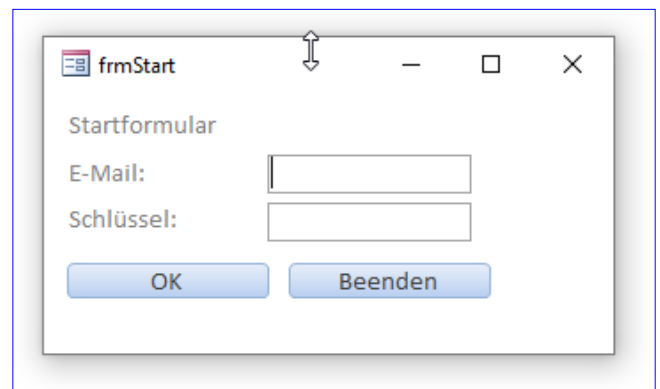


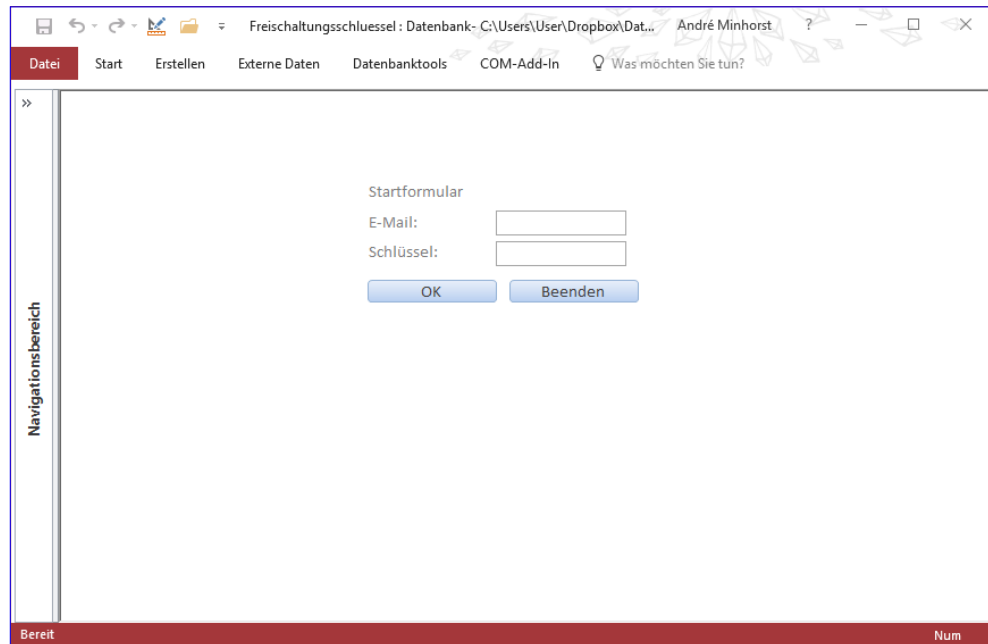
Bild 4: Formular als Popup öffnen

mieren und Maximieren an (siehe Bild 4). Dies ändern wir noch, indem wir die Eigenschaft **Rahmenart** auf **Keine** einstellen. Dadurch erscheint das Formular etwa unter Access 2016, wo der Hintergrund weiß ist, wie in Bild 5. Das sind jedoch Spielereien – wichtig ist, dass der Benutzer nicht um den Aufruf dieses Formulars herumkommt.

Übrigens: Wenn Sie das Formular auf diese Art geöffnet haben, können Sie es nicht über die herkömmlichen Elemente der Benutzeroberfläche wieder schließen – auch nicht mit der Tastenkombination **Strg + F4** oder durch einen Wechsel in die Entwurfsansicht mit der Tastenkombination **Strg + Komma (,)**.

Allerdings führt der Versuch, die Anwendung mit **Alt + F4** zu schließen, zunächst zum Schließen des Formulars und dann bei erneuter Betätigung zum Schließen der Anwendung. Diesen Weg müssten wir also noch blockieren. Dazu hinterlegen wir eine Prozedur für die Ereigniseigenschaft **Bei Taste ab** des Formulars selbst, die wir wie folgt füllen:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
        Case 115
            Select Case Shift
                Case acAltMask
                    KeyCode = 0
            End Select
        End Select
    End Select
End Sub
```



**Bild 5:** Anzeige des Startformulars

Damit diese Prozedur auch dann feuert, wenn eines der Steuerelemente des Formulars den Fokus hat, stellen wir noch die Eigenschaft **Tastenvorschau** auf **Ja** ein.

Wie können wir das Formular aber nun beim Entwickeln noch schließen, ohne direkt die komplette Anwendung schließen zu müssen? Der einzige Weg, der übrig bleibt, ist der über den Direktbereich des VBA-Fensters. Diesen rufen Sie mit der Tastenkombination **Strg + G** auf und geben dort die folgende Anweisung ein:

```
DoCmd.Close acForm, "frmStart"
```

### Eingabe des Freigabeschlüssels

Damit kommen wir zum Freigabeschlüssel selbst. Diesen soll der Benutzer neben der E-Mail-Adresse in das entsprechende Textfeld eingeben. Dann soll er den Schlüssel durch Betätigen der **OK**-Schaltfläche bestätigen. Ist die Kombination aus E-Mail und Freigabeschlüssel korrekt, sollen die beiden Informationen in der Tabelle **tblOptionen** gespeichert werden. Beim nächsten Aufruf der Anwendung sollen diese beiden Werte dann beim Aufruf des Formulars **frmStart** geprüft werden. Passen diese zusammen, setzt die Anwendung den Startvorgang fort und

zeigt das gewünschte Element der Benutzeroberfläche an. Anderenfalls wird der Benutzer gebeten, die korrekten Daten zur Freischaltung anzugeben.

Nachfolgend schauen wir uns mehrere Varianten an, wie wir die E-Mail-Adresse und den Schlüssel erstellen und prüfen können. Die erste Variante ist die zeitlich unbegrenzte, die zweite ist eine, mit der Sie eine Software befristet bis zu einem bestimmten Datum freischalten lassen können.

### Einfache Variante: E-Mail und Schlüssel müssen passen

Um die Schlüssel zu generieren, verwenden wir eine Klasse namens **clsCrypt** des Autors Tim Braun (siehe Modul im VBA-Projekt). Diese bietet vor allen die für uns interessante Funktion namens **GetSHA1** an, die eine als Parameter übergebene Zeichenkette in einen Schlüssel umwandelt.

Diese Funktion können wir nach der Initialisierung eines Objekts auf Basis der Klasse **clsCrypt** wie in der folgenden Prozedur testen:

```
Public Sub Test_GetSHA1()
    Dim objCrypt As clsCrypt
    Set objCrypt = New clsCrypt
    Debug.Print objCrypt.GetSHA1("André Minhorst")
End Sub
```

Das Ergebnis lautet dann:

```
B431123435AE6C58E661FBCA736FAEF739BE6F2F
```

Der Benutzer wird Probleme haben, daraus zu ermitteln, auf welche Weise der Schlüssel produziert wird. Es sollte natürlich nicht allzu einfach sein – wenn Sie also einfach nur die E-Mail-Adresse verschlüsseln und als Schlüssel zum Freischalten der Anwendung nutzen, besteht die Gefahr, dass der Benutzer sich seine eigenen Schlüssel erstellt.

Zum Erstellen eines einfachen Schlüssels verwenden wir nun die E-Mail-Adresse und hängen eine einfache Zeichenfolge wie etwa **xxx** an (diese darf natürlich ebenfalls etwas komplizierter ausfallen). Der zu kodierende Ausdruck heißt dann im Falle meiner eigenen E-Mail-Adresse etwa so: **andre@minhorst.comxxx**

Der verschlüsselte Ausdruck heißt dann:

```
D382469636E463CE93079D0F9714212EFC4D54A6
```

Wir wollen uns eine kleine Funktion bauen, um die Schlüssel einfacher herstellen zu können. Diese sieht so aus:

```
Public Function Verschluesseln(strAusdruck As String) As String
    Dim objCrypt As clsCrypt
    Dim strVerschluesselt As String
    Set objCrypt = New clsCrypt
    strVerschluesselt = objCrypt.GetSHA1(strAusdruck)
    Verschluesseln = strVerschluesselt
End Function
```

Im Direktfenster setzen Sie die Funktion wie folgt ein:

```
? Verschluesseln("andre@minhorst.comxxx")
D382469636E463CE93079D0F9714212EFC4D54A6
```

### Eingabe des Schlüssels

Nun, wo wir eine Methode zum Erzeugen der Schlüssel programmiert haben, wollen wir uns um das Formular zur Eingabe der E-Mail-Adresse und des Schlüssels kümmern. Dazu fügen wir der Schaltfläche **cmdOK** den Code aus Listing 1 hinzu.

Die Ereignisprozedur **cmdOK\_Click** trägt die Werte der beiden Textfelder **txtEMail** und **txtSchluessel** in die beiden Variablen **strEMail** und **strSchluessel** ein. Dann prüft sie in einer **If...Then**-Bedingung, ob der verschlüsselte Wert der E-Mails-Adresse plus der angehängten Zeichenkette **xxx** mit der vom Benutzer eingegebenen Schlüssel-

Zeichenfolge übereinstimmt. Ist dies der Fall, ruft die Prozedur die Funktion **SchlüsselSchreiben** auf und übergibt die E-Mail und den Schlüssel an diese Funktion. Es erscheint eine Meldung, welche die erfolgreiche Freischaltung bestätigt und das Formular wird geschlossen.

Hat der Benutzer nicht die richtige Kombination aus E-Mail-Adresse und Schlüssel eingegeben, erscheint ebenfalls eine Meldung und das Start-Formular bleibt geöffnet. Er kann es nun erneut probieren oder aber mit einem Klick auf die Schaltfläche **cmdBeenden** die Anwendung schließen.

### Schreiben von E-Mail und Schlüssel

Die von der oben beschriebenen Prozedur aufgerufene Funktion **SchlüsselSchreiben** erwartet die E-Mail-Adresse sowie den Schlüssel als Parameter (siehe Listing 2).

```
Private Sub cmdOK_Click()
    Dim strEMail As String
    Dim strSchluessel As String
    strEMail = Me!txtEMail
    strSchluessel = Me!txtSchluessel
    If VerschluesseIn(strEMail & "xxx") = strSchluessel Then
        SchlüsselSchreiben strEMail, strSchluessel
        MsgBox "Die Anwendung wurde erfolgreich freigeschaltet."
        DoCmd.Close acForm, Me.Name
    Else
        MsgBox "E-Mail-Adresse und/oder Schlüssel sind nicht korrekt."
    End If
End Sub
```

**Listing 1:** Prüfung des Schlüssels

```
Private Sub SchlüsselSchreiben(strEMail As String, strSchluessel As String)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblOptionen", dbOpenDynaset)
    If rst.EOF Then
        rst.AddNew
        rst!EMail = strEMail
        rst!Schluessel = strSchluessel
        rst.Update
    Else
        rst.Edit
        rst!EMail = strEMail
        rst!Schluessel = strSchluessel
        rst.Update
    End If
End Sub
```

**Listing 2:** Schreiben des Schlüssels

Die Funktion prüft, ob sich bereits ein Datensatz in der Tabelle **tblOptionen** befindet. Falls nicht, fügt sie einen neuen Datensatz hinzu und trägt die beiden Werte aus den Parametern **strEMail** und **strSchluessel** in die beiden Felder der Tabelle ein. Ist bereits ein Datensatz vorhanden, werden die Inhalte der beiden Felder durch die neuen Werte ersetzt.

Die Tabelle sieht danach etwa wie in Bild 6 aus.

ID	EMail	Schluessel
1	andre@minhorst.com	D382469636E463CE93079D0F9714212EFC4D54A6
*	(Neu)	

**Bild 6:** Tabelle mit den Freischaltungsdaten

### Inhalt der Tabelle beim Öffnen prüfen

Nun haben wir allerdings nur den Fall abgedeckt, dass der Benutzer die Anwendung zum ersten Mal öffnet. Wenn er nun eine gültige Kombination aus E-Mail-Adresse und Schlüssel angegeben hat, soll er ja nicht bei jedem erneuten Öffnen der Anwendung erneut die Daten eingeben müssen. Also wollen wir die Prozedur **cmdOK\_Click** so erweitern, dass diese zunächst prüft, ob bereits gültige Daten in der Tabelle **tblOptionen** vorliegen.

```
Private Function SchlüsselPruefen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strEMail As String
    Dim strSchlüssel As String
    Dim bolSchlüsselPruefen As Boolean
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblOptionen", dbOpenDynaset)
    If Not rst.EOF Then
        strEMail = rst!EMail
        strSchlüssel = rst!Schlüssel
        If VerschluesseIn(strEMail & "xxx") = strSchlüssel Then
            bolSchlüsselPruefen = True
        Else
            bolSchlüsselPruefen = False
        End If
    Else
        bolSchlüsselPruefen = False
    End If
    SchlüsselPruefen = bolSchlüsselPruefen
End Function
```

**Listing 3:** Prüfen, ob der in der Tabelle **tblOptionen** gespeicherte Schlüssel korrekt ist

Dazu fügen wir der Ereignisprozedur, die durch das Ereignis **Beim Öffnen** ausgelöst wird, die folgenden Anweisungen hinzu:

```
Private Sub Form_Open(Cancel As Integer)
    If SchlüsselPruefen = True Then
        Cancel = True
    End If
End Sub
```

Wenn die Funktion **SchlüsselPruefen** den Wert **True** zurückliefert, stellt die Prozedur den Wert des Parameters **Cancel** auf **True** ein. Dies führt dazu, dass das Formular **frmStart** direkt wieder geschlossen wird.

Es fehlt nun noch die Funktion **SchlüsselPruefen**. Diese Funktion finden Sie in Listing 3. Die Funktion erwartet keine Parameter, da sie die zu prüfenden Werte direkt aus der Tabelle **tblOptionen** einliest. Auch hier kann es wieder sein, dass noch gar kein Datensatz in der Tabelle gespeichert ist. Dies prüft die Prozedur gleich in der ersten **If...**

**Then**-Bedingung mit dem Ausdruck **rst.EOF**. Liefert dies **True**, enthält die Tabelle noch keinen Datensatz. In diesem Fall soll die Funktion den Wert **False** zurückliefern.

Ist hingegen bereits ein Datensatz vorhanden, liest die Prozedur die beiden Werte der Felder **EMail** und **Schlüssel** in die Variablen **strEMail** und **strSchlüssel** ein.

In einer weiteren **If...Then**-Bedingung prüft die Funktion dann, ob der mit der Funktion **VerschluesseIn** verschlüsselte Ausdruck aus der E-Mail-Adresse und der Zeichenkette **xxx** dem Schlüssel aus **strSchlüssel** entspricht. Ist dies der Fall, erhält die Variable **bolSchlüsselPruefen** den Wert **True**, anderenfalls den Wert **False**.

Wenn der Benutzer nun einmalig die richtige Kombination aus E-Mail-Adresse und Schlüssel eingegeben hat, wird das Formular **frmStart** erst gar nicht geöffnet – der Vorgang endet bereits in der Prozedur, die durch das Ereignis **Beim Öffnen** ausgelöst wird.

```
Public Function VerschluesseInMitFunktion(strMail As String, bolFunktion1 As Boolean, bolFunktion2 As Boolean, _  
    bolFunktion3 As Boolean) As String  
    Dim objCrypt As clsCrypt  
    Dim strVerschluesseIt As String  
    Dim strAusdruck As String  
    Set objCrypt = New clsCrypt  
    strAusdruck = strMail & "xxx" & bolFunktion1 * -4 + bolFunktion2 * -2 + bolFunktion3 * -1  
    Debug.Print strAusdruck  
    strVerschluesseIt = objCrypt.GetSHA1(strAusdruck)  
    VerschluesseIn = strVerschluesseIt  
End Function
```

**Listing 4:** Neue Version der Funktion **VerschluesseIn** namens **VerschluesseInMitFunktion**, die weitere Eigenschaften berücksichtigt

### Weitere Bedingung mit dem Schlüssel übergeben

Dies war die einfachste Variante für das Freischalten der Software durch die Angabe einer E-Mail-Adresse und einen Schlüssel. Es kann aber sein, dass Sie weitergehende Anforderungen haben, die über die Angabe von E-Mail-Adresse und Registrierungsschlüssel übermittelt werden.

Zum Beispiel könnte es sein, dass die Anwendung verschiedene Funktionen bereitstellt, von denen je nach erworbener Lizenz nur eine oder mehrere Funktionen freigeschaltet sein sollen. Wir wollen dies im Folgenden so nachbilden, dass im Anschluss an das Formular **frmStart** das Formular **frmAnwendung** geöffnet wird, von dem abhängig vom vorliegenden Schlüssel eine oder mehrere Schaltflächen aktiviert oder auch nicht aktiviert sind.

Das Formular **frmStart** soll also grundsätzlich direkt wieder verschwinden, wenn der Benutzer einen der gültigen Schlüssel erhalten hat.

Die einzelnen Funktionen wollen wir durch verschiedene Zusätze codieren, die wir in Form einer Zahl an die E-Mail-Adresse und die Zeichenfolge **xxx** anhängen. Wir wollen drei Funktionen vereinbaren, die wir in einer binären Zahl als nutzbar oder nicht nutzbar kodieren. Wenn alle drei Funktionen verfügbar sind, soll die binäre Zahl also **111** lauten und als **7** an die zu verschlüsselnde Zeichenfolge angehängt werden. Wenn nur die letzte Funktion verfügbar

sein soll, verwenden wir den Code **001**, also **1**, wenn nur die erste Funktion verfügbar ist, den Code **100**, also **4**.

Achtung: Diese Version der Lösung finden Sie unter dem Dateinamen **Freischaltungsschlüssel\_II.accdb**.

Die Funktion **VerschluesseInMitFunktion**, die wir ausschließlich zum Erstellen der Schlüssel verwenden, erwartet neben der Angabe der E-Mail-Adresse noch drei **Boolean**-Werte für die Parameter **bolFunktion1**, **bolFunktion2** und **bolFunktion3**, die angeben, ob die Funktionen 1, 2 oder 3 freigeschaltet werden sollen (siehe Listing 4). Die drei Boolean-Variablen werden mit den Werten **-4**, **-2** oder **-1** multipliziert und addiert. Da die Boolean-Parameter im Falle von **True** den Wert **-1** aufweisen, erhalten wir auf diese Weise Werte von **0** bis **7** für diese Berechnung. Der Rest der Funktion funktioniert wie die bisher bekannte Variante, fügt allerdings auch die Zeichenkette **xxx** noch automatisch hinzu.

Dadurch ergibt sich auch eine neue Version der Ereignisprozedur, die durch die Schaltfläche **cmdOK** ausgelöst wird. Diese sieht wie in Listing 5 aus und enthält eine zusätzliche Variable namens **i**. Diese dient als Variable in einer **For...Next**-Schleife, die alle Zahlen von 0 bis 7 durchläuft – also alle Werte, die für die verschiedenen Kombinationen der freigeschalteten Funktionen der Anwendung vorkommen können. Innerhalb der Schleife haben wir eine **If...Then**-Bedingung, die prüft, ob der ver-

## DHL-Versand vorbereiten

Ende Februar 2018 hat DHL die Unterstützung der Intraship-Schnittstelle beendet. Das heißt, dass wir unsere DHL-Etiketten nun auf eine andere Weise erstellen müssen. Für Datenbank-Experten kein Problem: Wir benötigen nur eine Beschreibung der gewünschten Schnittstelle und liefern dann die Daten in der gewünschten Form.

### Abkündigung von Intraship

Wer DHL-Kunde ist und bisher seine Paket-Etiketten mit Intraship erstellt hat, ist von DHL per Mail hinreichend über die anstehende Einstellung dieses Services informiert worden. Wer das lange genug ignoriert hat, hat Anfang März 2018 zwar noch den Link **Versenden (Intraship)** auf der Startseite des Geschäftskundenportals gesehen (siehe Bild 1), aber hinter dem Link landete man dann auf einer Seite, auf der die Anmeldung nicht mehr funktionierte.



Bild 1: Das Versenden per Intraship funktioniert nicht mehr.

Stattdessen soll man nun die Benutzeroberfläche nutzen, die sich direkt über den Link Versenden öffnen lässt und der die möglichen Dienste in übersichtlicher Form darstellt (siehe Bild 2).

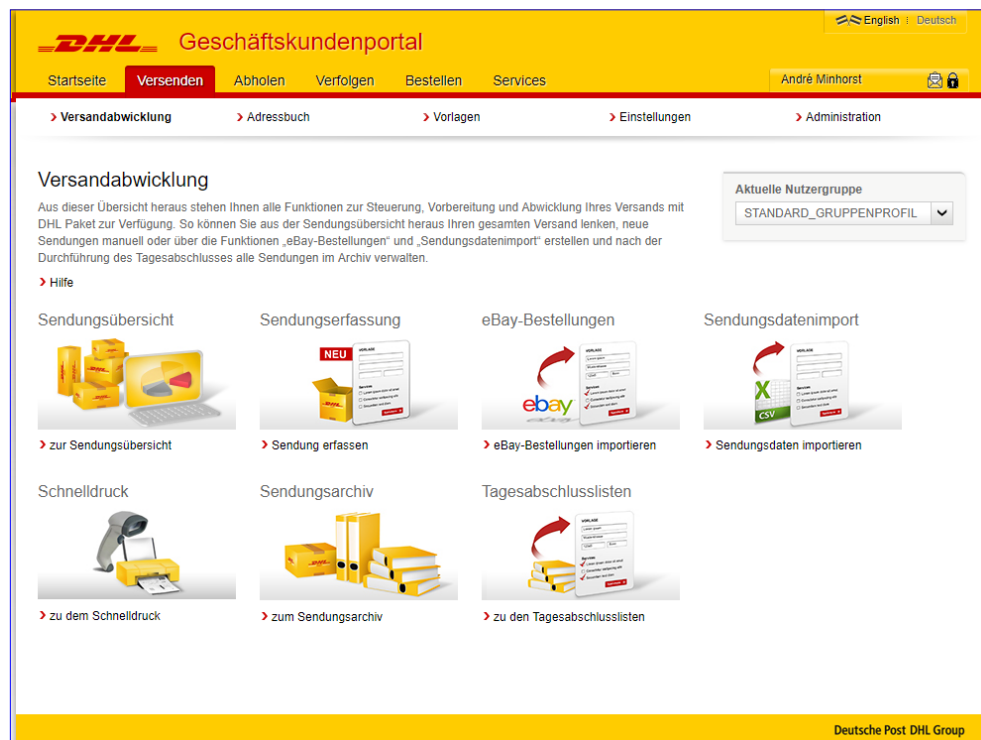


Bild 2: Neue Benutzeroberfläche für den Versand von DHL-Paketen

Was aber ändert sich nun für uns, wenn wir bis dahin unsere Paket-Etiketten mit Intraship erstellt haben? Zunächst einmal können wir das alte Export-Format, das wir im Beitrag **Versandetiketten mit DHL-Intraship** ([www.access-im-unternehmen.de/991](http://www.access-im-unternehmen.de/991)) vorgestellt haben, verabschieden. Das Format war nicht gerade übersichtlich, daher ist der Wechsel

zu einem neuen Format vielleicht gar nicht schlecht. Dafür programmieren wir den Export nun komplett neu. Wie die Export-Datei aussehen muss, finden wir im Anschluss heraus.

Dazu klicken wir zunächst auf den Link **Sendungsdaten importieren**, was den Dialog aus Bild 3 öffnet. Hier finden wir bereits die Möglichkeit, eine Vorlage auszuwählen sowie eine Schaltfläche zum Auswählen der zu importierenden CSV-Datei. Wir benötigen allerdings Informationen darüber, wie die CSV-Datei aufgebaut sein soll.

Klicken Sie hier im Text auf den Link **> Vorlagen**, öffnet sich der Dialog aus Bild 4.

Wenn wir uns nun beispielsweise die Vorlage **DHL Vorlage Sendungsdatenimport** ansehen, finden wir eine Definition der Vorlage vor, die im oberen Teil zunächst allgemeine Formatierungsvorgaben liefert (siehe Bild 5).

Dort legen Sie beispielsweise fest, welches Trennzeichen zwischen den Inhalten der einzelnen Spalten angegeben werden soll, welches Zeichen als Dezimalzeichen zum Einsatz kommt und wie Texte eingefasst werden – hier etwa durch Anführungszeichen. Außerdem legen Sie den Zeichensatz (hier ISO 8859-1) und das Datumsformat (TT.MM.JJ) fest.

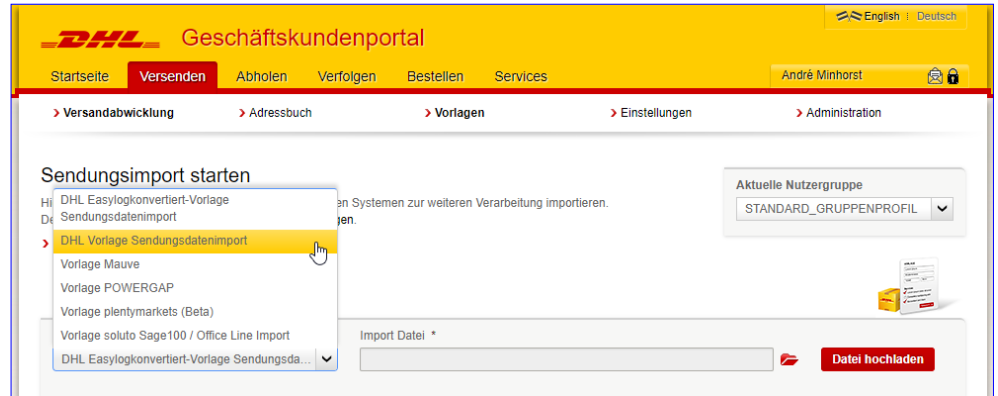


Bild 3: Maske zum Hochladen der CSV-Datei

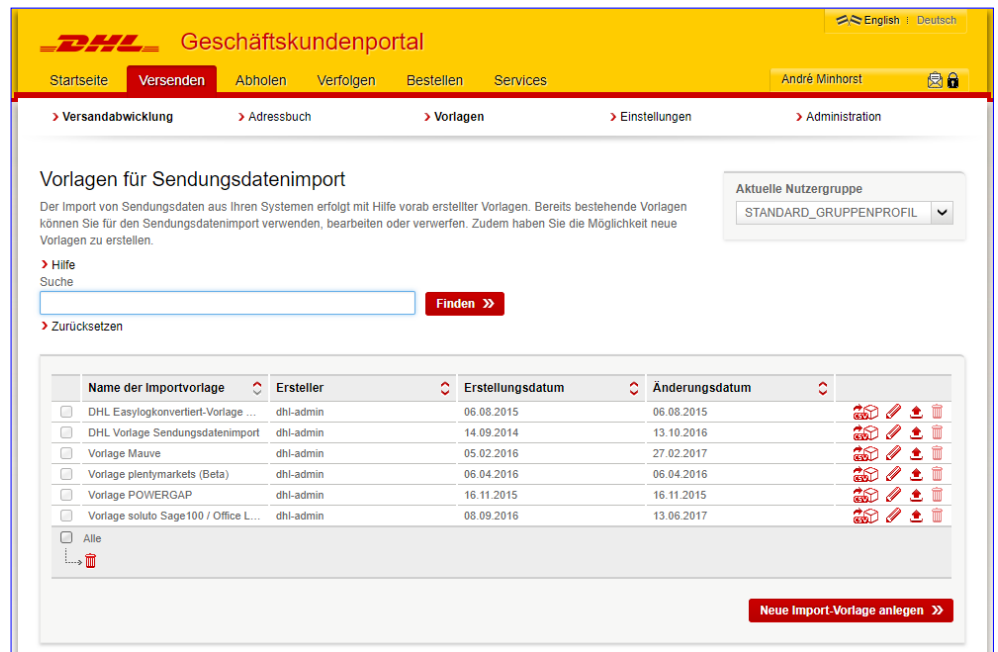


Bild 4: Auswahl einer CSV-Vorlage

Darunter befinden sich alle Felder, die DHL überhaupt für den Versand einer Lieferung verwenden kann

Hier wurde für alle zu verwendenden Eigenschaften die Spaltenüberschrift der zu importierenden CSV-Datei angegeben.

Nach kurzem Überfliegen der angegebenen Spaltenüberschriften stellen wir fest, dass wir für unsere Zwecke viel weniger Spalten zur Verfügung stellen wollen. Dazu müssen wir eine neue Vorlage anlegen.



## Neue Vorlage anlegen

Eine neue Vorlage legen wir ausgehend von der Seite Vorlagen für Sendungsdatenimport an, indem wir dort auf den Link **Neue Import-Vorlage anlegen** >> klicken.

Hier finden wir die gleiche Ansicht wie in der zuvor betrachteten Vorlage vor – mit dem Unterschied, dass die rechte Spalte noch nicht mit den Spaltennamen der von uns gelieferten CSV-Datei gefüllt sind.

Das kann auch nicht der Fall sein, da das System unsere CSV-Datei ja noch gar nicht kennt.

Um das zu ändern, müssten wir auch zunächst einmal eine CSV-Datei erstellen, denn diese können wir dann unter dem Punkt **2. Hochladen der Musterdatei zum Auslesen der Inhalte** auswählen und mit einem Klick auf den Link **Datei analysieren** >> untersuchen lassen (siehe Bild 6).

## Bestehende Lösung erweitern

Im Beitrag **Bestellverwaltung mit Versand (www.access-im-unternehmen.de/993)** haben wir bereits eine kleine Lösung program-

miert, die uns per Mausclick einen CSV-Export für die bis vor kurzem aktive Schnittstelle Intraship erzeugt

Bild 5: Aussehen einer der vorgefertigten Vorlagen

Bild 6: Hochladen der Musterdatei

hat. Diese wollen wir nun aufgreifen und so anpassen, dass wir einen Export erhalten, den wir mit der neuen Schnittstelle nutzen können. In dieser Lösung haben wir mit einigen Klassen gearbeitet, von denen jede zum Zusammenstellen einer Zeile eines Satzes für eine DHL-Lieferung diente.

Unter Intraship bestand die CSV-Datei nämlich nicht nur aus einer Zeile je Datensatz, sondern die Daten wurden auf mehrere Zeilen aufgeteilt. Das ist nun natürlich hinfällig – die neue Schnittstelle nimmt nur noch eine Zeile je Datensatz an. Das heißt, dass wir die in der bestehenden Lösung verwendeten Klassen auch mehr oder weniger aufgeben können.

Jede dieser Klassen stellte einige Eigenschaften zur Verfügung, mit denen die gewünschten Werte etwa aus einem Datensatz der Datenbank gefüllt werden konnten. Außerdem lieferten die Klassen für die Eigenschaft Satz die komplette Zeile mit den Daten dieser Klasse.

### Klasse zum Zusammenstellen von Datensätzen für die CSV-Datei

Eine solche Klasse wollen wir nun für die neuen Daten auch erstellen.

Wir arbeiten uns dabei durch die einzelnen Zeilen der Liste aus Bild 7 durch und berücksichtigen dabei zunächst nur die offensichtlich notwendigen Felder.

**3. Zuweisen der Inhalte zu den erforderlichen Angaben**

Legende:  Spaltenauswahl  Spaltenname  feste Konstante

Systemelemente ?	Manuelle Zuordnung ?	
<b>Sendung</b>		
Sendungsreferenz	<input type="text"/>	
Sendungsref. (Retoure)	<input type="text"/>	
Sendungsdatum	<input type="text"/>	
Sendungserfassungsvorlage	<input type="text"/>	
Creation-Software	<input type="text"/>	
<b>Adresse</b>		
<b>Absenderdetails</b>		
Absenderreferenz	<input type="text"/>	
Name 1 (Absender)	<input type="text"/>	
Name 2 (Absender)	<input type="text"/>	
Name 3 (Absender)	<input type="text"/>	
<b>Straße (Absender)</b>	<input type="text"/>	
Hausnummer (Absender)	<input type="text"/>	
Straße und Hausnummer (Absender)	<input type="text"/>	
Adresszusatz 1 (Absender)	<input type="text"/>	
Adresszusatz 2 (Absender)	<input type="text"/>	
Zustellinformation (Absender)	<input type="text"/>	
PLZ (Absender)	<input type="text"/>	
Ort (Absender)	<input type="text"/>	
Postleitzahl und Ort (Absender)	<input type="text"/>	
Provinz (Absender)	<input type="text"/>	
Land (Absender)	<input type="text"/>	
Ansprechpartner (Absender)	<input type="text"/>	
E-Mail-Adresse (Absender)	<input type="text"/>	

**Bild 7:** Freie Felder zum Zuweisen der Spalten der Vorlage

Hier stellt sich nun die Frage, wie wir am schnellsten zum Ziel kommen. Die bisherige Lösung bestand aus einer Hauptklasse, welche die unterschiedlichen Kategorien von Informationen wie Sendung, Benachrichtigungen, Absender, Empfänger und so weiter in einzelnen Klassen behandelt hat. Auf diese Weise war das Zusammenstellen leicht zu programmieren.

Wollen wir auf diesen Klassen aufsetzen und diese so ändern, dass diese die Daten nun so zurückliefern, wie wir sie für den Import benötigen? Oder legen wir eine komplett neue Klasse an? Wir entscheiden uns, die einmal gemachte Arbeit als Grundlage für den neuen Export zu nutzen und passen die bestehenden Klassen an.



vorerst auskommentiert haben. Gleiches erledigen wir mit der Zeile, welche den Wert der Variablen **m\_Ordnungsnummer** hinzufügt. Diese haben wir in der Methode **Satz** der Klasse **clsDPEEMain** zwar schon nicht mehr an die einzelnen Klassen übergeben, weshalb diese im obigen Beispiel für die Ausgabe nicht mehr auftauchte, aber wir wollen auch noch die leere Stelle in der Ausgabe entfernen:

```
Public Property Get Satz() As String
    Dim strSatz As String
    'strSatz = strSatz & m_Ordnungsnummer & "|"
    'strSatz = strSatz & "DPEE-ITEM|"
    strSatz = strSatz & m_GewichtDesPackstueckesInKg & "|"
    strSatz = strSatz & m_LaengeDesPackstueckesInCm & "|"
    strSatz = strSatz & m_BreiteDesPackstueckesInCm & "|"
    strSatz = strSatz & m_HoeheDesPackstueckesInCm & "|"
    strSatz = strSatz & m_PackstueckBeschreibung & "|"
    strSatz = strSatz & m_PackartKollitraeger & "|"
    strSatz = strSatz & m_Packstueckreferenz & "|"
    strSatz = strSatz & m_Referenznummer & "|"
    ' strSatz = Left(strSatz, Len(strSatz) - 1)
    Satz = strSatz
End Property
```

Schließlich entfernen wir auch noch die Zeile, die das abschließende Pipe-Zeichen entfernt – immerhin wollen wir später alle fünf Sätze in eine Zeile schreiben, wo wir auch das Trennzeichen zwischen dem letzten Element des vorhergehenden und dem ersten Element des folgenden Satzes ein Trennzeichen benötigen.

Auf die gleiche Art und Weise bearbeiten wir die Methode **Satz** der übrigen Klassen. Damit sieht die Ausgabe schon etwas schlanker aus:

```
EPN|20180306|1|EUR|01|62754
98028|André Minhorst Verlag|AndréMinhorst|Borkhofer Str
.|17|47137|Duisburg|DE|andre@minhorst.com|0203/1212121
2|Herr André Minhorst|André Minhorst
```

```
Verlag||Herr André Minhorst|Borkhofer Str.|17|47137|Duis
burg|DE|andre@minhorst.com|PK|andre@minhorst.com|
```

### Spaltenüberschrift erstellen

Nun wollen wir die Spaltenüberschriften hinzufügen. Das ist eine reine Fleißarbeit. Wir schnappen uns dazu zum Beispiel die folgenden Anweisungen der **Satz**-Methode der Klasse **clsDPEEItem**:

```
strSatz = strSatz & m_GewichtDesPackstueckesInKg & "|"
strSatz = strSatz & m_LaengeDesPackstueckesInCm & "|"
strSatz = strSatz & m_BreiteDesPackstueckesInCm & "|"
strSatz = strSatz & m_HoeheDesPackstueckesInCm & "|"
strSatz = strSatz & m_PackstueckBeschreibung & "|"
strSatz = strSatz & m_PackartKollitraeger & "|"
strSatz = strSatz & m_Packstueckreferenz & "|"
strSatz = strSatz & m_Referenznummer & "|"
```

Dann entfernen wir die Zeilenumbrüche, den vorderen Teil bis zum Unterstrich (**strSatz = strSatz & m\_**) lassen hinten auch nur das Pipe-Zeichen über. Das Ergebnis sieht dann für die erste Klasse wie folgt aus:

```
GewichtDesPackstueckesInKg|LaengeDesPackstueckesIn-
Cm|BreiteDesPackstueckesInCm|HoeheDesPackstueckesIn-
Cm|PackstueckBeschreibung|PackartKollitraeger|Packstueck-
referenz|Referenznummer|
```

Statt dieses Zeichenkette zu speichern und bei Bedarf bereitzustellen, können wir auch den Klassen jeweils eine Eigenschaft namens **Kopfzeile** hinzufügen, welche am Beispiel der Klasse **clsDPEEItem** wie folgt aussieht:

```
Public Property Get Kopfzeile() As String
    Dim strKopf As String
    strKopf = strKopf & "GewichtDesPackstueckesInKg|"
    strKopf = strKopf & "LaengeDesPackstueckesInCm|"
    strKopf = strKopf & "BreiteDesPackstueckesInCm|"
    strKopf = strKopf & "HoeheDesPackstueckesInCm|"
    strKopf = strKopf & "PackstueckBeschreibung|"
```

```
strKopf = strKopf & "PackartKollitraeger|"
strKopf = strKopf & "Packstueckreferenz|"
strKopf = strKopf & "Referenznummer|"
Satz = strKopf
End Property
```

Diese ist dann genauso aufgebaut wie die Eigenschaft Satz. Wenn einmal eine Zeile hinzukommt oder wegfällt, können wir auch die Spaltenköpfe viel leichter anpassen. Für die beiden Klassen **clsDPEESender** und **clsDPEEReceiver**, die teilweise gleich bezeichnete Eigenschaften enthalten, stellen wir den Spaltenüberschriften noch jeweils die Zeichenkette **Sender\_** beziehungsweise **Receiver\_** voran:

```
Public Property Get Kopfzeile() As String
    Dim strKopf As String
    strKopf = strKopf & "Sender_Kundennummer|"
    strKopf = strKopf & "Sender_Firmenname1|"
    strKopf = strKopf & "Sender_Firmenname2|"
    strKopf = strKopf & "Sender_Kontaktperson|"
    strKopf = strKopf & "Sender_Strasse|"
    strKopf = strKopf & "Sender_Hausnummer|"
    strKopf = strKopf & "Sender_ZusaetzlicheAdressinformation|"
    strKopf = strKopf & "Sender_PLZ|"
    strKopf = strKopf & "Sender_Stadt|"
    strKopf = strKopf & "Sender_Laendercode|"
    strKopf = strKopf & "Sender_Bemerkung|"
    strKopf = strKopf & "Sender_Emailadresse|"
    ...
    Kopf = strKopf
End Property
```

In der Klasse **clsDPEEMain** fügen wir ebenfalls eine neue Methode hinzu, welche die Kopfzeilen-Anteile der einzelnen Klassen zusammenführt:

```
Public Property Get Kopfzeile() As String
    Dim strKopf As String
    strKopf = strKopf & m_DPEEShipment.Kopfzeile
    strKopf = strKopf & m_DPEESender.Kopfzeile
```

```
strKopf = strKopf & m_DPEEReceiver.Kopfzeile
strKopf = strKopf & m_DPEEItem.Kopfzeile
strKopf = strKopf & m_DPEENotification.Kopfzeile
Kopfzeile = strKopf
End Property
```

Wichtig ist hier, dass die Kopfzeilen der einzelnen Abschnitte in der gleichen Reihenfolge zusammengefügt werden wie die Inhalte der Abschnitte selbst.

### Weitere Änderungen

Die neue Vorlage erlaubt wohl sowohl nur ein einzelnes Packstück und auch nur eine Notification. Daher müssen wir in der Klasse **clsDPEEMain** ein paar Anpassungen vornehmen. Als Erstes fliegen die nachfolgend auskommentierten Zeilen aus der Methode **Initialize** heraus. Dafür kommen die beiden Anweisungen zur Methode hinzu, welche jeweils ein Objekt auf Basis der Klasse **clsDPEEItem** und eines der Klasse **clsDPEENotification** initialisieren:

```
Private Sub Class_Initialize()
    Set m_DPEEShipment = New clsDPEEShipment
    Set m_DPEESender = New clsDPEESender
    Set m_DPEEReceiver = New clsDPEEReceiver
    ' Set m_DPEEItems = New Collection
    ' Set m_DPEENotifications = New Collection
    Set m_DPEEItem = New clsDPEEItem
    Set m_DPEENotification = New clsDPEENotification
End Sub
```

Damit wir von außen auf die entsprechenden Elemente zugreifen können, fügen wir die beiden öffentlichen Eigenschaften hinzu:

```
Public Property Get DPEEItem() As clsDPEEItem
    Set DPEEItem = m_DPEEItem
End Property
```

```
Public Property Get DPEENotification() As clsDPEENotification
    Set DPEENotification = m_DPEENotification
```

direkt auf die **Drucken**-Schaltfläche klicken, um ein PDF des Etiketts zu erzeugen und zu drucken. Achtung: Sie müssen die Anzeige von Popup für diese Seite aktivieren, da die Etiketten sonst nicht angezeigt werden!

Nach dem Anklicken der **Drucken**-Schaltfläche erscheint das Etikett in einem neuen Fenster. Sie können es nun ausdrucken oder speichern (siehe Bild 21).

### Einsetzen der Lösung in einer eigenen Anwendung

Der schwierige Teil für den Leser ist es nun, die hier vorgestellte abgewandelte Lösung des Intraship-Imports in eine eigene Anwendung einzubauen. Als Erstes fügen Sie die sechs Klassen **clsDPEEMain**, **clsDPEEShipment**, **clsDPEEReceiver**, **clsDPEESender**, **clsDPEENotification** und **clsDPEEShipment** zur Zielanwendung hinzu.

Außerdem kopieren Sie die Tabelle **tblOptionen** in die Zielanwendung. Diese haben wir zuvor in **tblOptionenDHL** umbenannt, da gegebenenfalls bereits eine Tabelle namens **tblOptionen** vorhanden ist.

Schließlich benötigen wir noch das Modul **mdID-PEE**, welches die Prozeduren **SatzErstellen** und **KopfzeileErstellen** enthält. Diese nutzen wir als Vorlage, um eine neue passende Prozedur für die Zielanwendung zu programmieren. Außerdem enthält dieses Modul noch ein paar Hilfsroutinen und die Enumerationen.j

Die Funktion **SatzErstellen** passen Sie dann so an, dass diese die in Ihrer Anwendung enthaltenen Daten aus den Tabellen entnimmt und in die Eigenschaften der Klassen schreibt. Dabei können Sie durchaus auch mehrere Sätze gleichzeitig in eine CSV-Datei schreiben – Sie müssen dazu nur eine entsprechende Schleife anlegen und diese

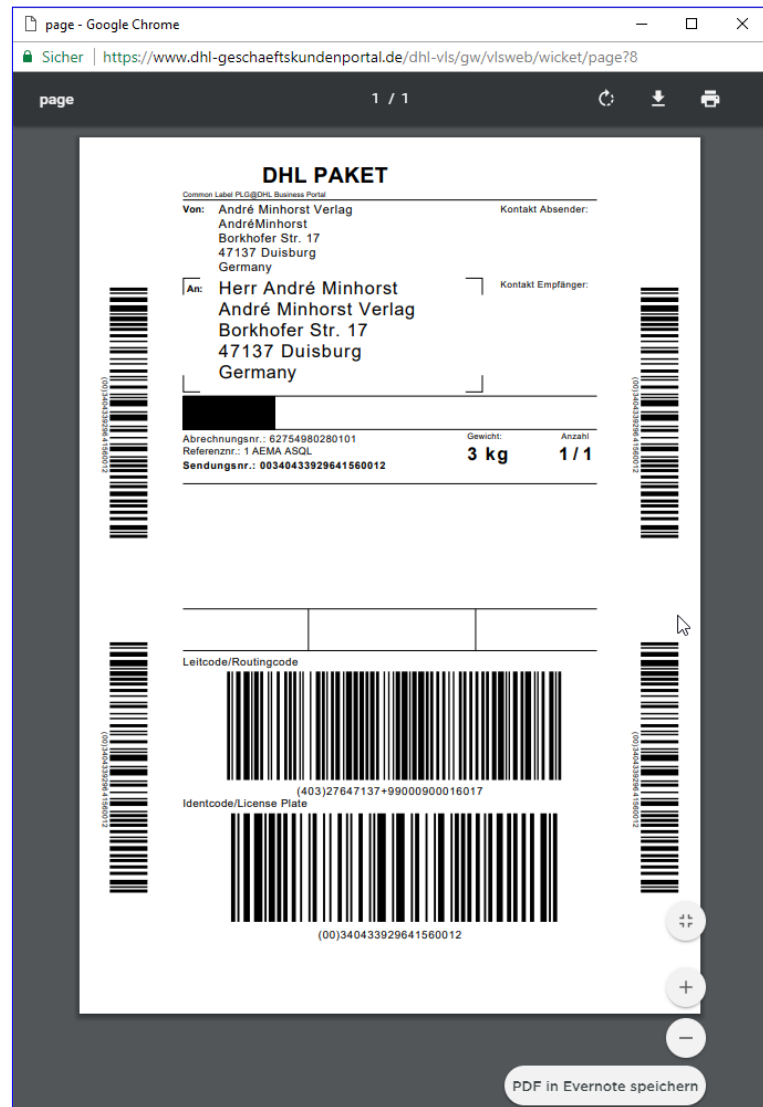


Bild 21: Fertiges Etikett

durchlaufen. Dabei würden Sie dann jeweils die Funktion **Satz** aufrufen und diesen an die CSV-Datei anhängen.

### Zusammenfassung und Ausblick

Dieser Beitrag stellt eine Sammlung von Klassen vor, mit denen Sie Versanddaten aus einer Access-Anwendung in eine CSV-Datei umwandeln können. Die Lösung wurde von einer bestehenden Lösung abgeleitet, die mit der Intraship-Schnittstelle von DHL zusammengearbeitet hat. Diese Schnittstelle wurde nun jedoch deaktiviert, sodass eine neue Version unserer Lösung programmiert werden musste.