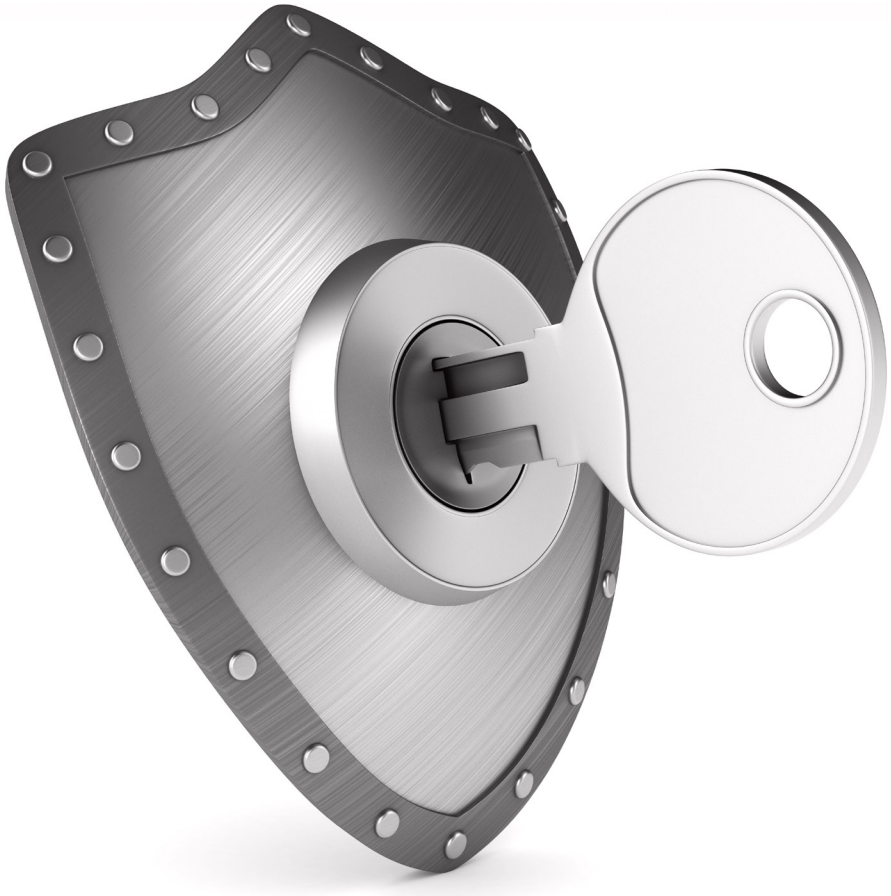
HAUFE.
Ausgabe 03/2020

ACCESS
IM UNTERNEHMEN

SQL SERVER-SICHERHEIT

Einstieg in sichere Datenbanken, Teil 1: Zugriffsschutz in Access (ab S. 35)



In diesem Heft:

Mat-Nr. H1583-5119

SQL Server-Sicherheit

Microsoft hat die Sicherheits-Funktionen von Access-Datenbanken schon vor langer Zeit beschnitten. Vielleicht mit dem Hintergedanken, mehr Benutzer zum SQL Server zu locken – vielleicht aber auch, weil klar war, dass das Access-Sicherheitssystem nur eine trügerische Sicherheit bot. Wir schauen uns an, welche Sicherheitsfunktionen Access bietet – und arbeiten uns dann zur optimal gesicherten Datenbank vor.

Die Sicherheit steht bei der Entwicklung von Access-Anwendungen meist nicht direkt im Vordergrund, sind doch viele Anwendungen improvisiert und nur für die vorübergehende Nutzung gedacht. Die eine oder andere Anwendung wird aber dann doch so lange genutzt, bis man sich Gedanken darüber machen muss, ob die darin verwalteten Daten überhaupt sicher untergebracht sind. Das ist in vielen Fällen nicht der Fall, denn Access bietet keine umfassenden Sicherheitsfunktionen. Die können Sie aber ergänzen, wenn Sie die Tabellen der Datenbank beispielsweise in eine SQL Server-Datenbank auslagern und von einer Access-Anwendung aus darauf zugreifen. Dann stehen die vollständigen Sicherheitsfunktionen zur Verfügung, die auch für Anwendungen mit sensibleren Daten zum Einsatz kommen. In der Beitragsreihe **Access und SQL Server-Security, Teil 1: Zugriffsschutz in Access** schaut sich Migrationsexperte Bernd Jungbluth ab Seite 35 an, welche Möglichkeiten sich bezogen auf die Datensicherheit mit Access-Bordmitteln ergeben.

Wer öfter mal mit Währungsgeschäften zu tun hat, ist an den aktuellen und auch an den historischen Wechselkursen zwischen Währungen interessiert. Im Beitrag **Wechselkurse per Webservice** zeigen wir ab Seite 48, wie Sie einen Webservice von Access aus abfragen und die Wechselkurse für verschiedene Kombinationen aus Währungen und Zeiten in die Tabellen einer Datenbank schreiben.

Umfangreichere Anwendungen gehen beizeiten während der Entwicklung kaputt. Lässt sich eine solche nicht mehr reparieren, legt man eine neue, leere Datenbank an und importiert alle Objekte der alten Datenbank hinein. Das geht mit wenigen Mausklicks. Aufwendiger wird es, wenn die Anwendung viele Verweise nutzt, die manuell neu angelegt werden müssen. Unsere Lösung aus dem Beitrag **Verweise aus anderer Datenbank importieren** zeigt ab Seite 62, wie Sie diese Aufgabe mit einem Add-In und wenigen Mausklicks erledigen.

Bestellverwaltungen sind eines der Standardbeispiele für Access-Anwendungen. Allerdings gilt es dort mitunter Bedingungen zu berücksichtigen, die nicht so einfach zu programmieren sind: Zum Beispiel sollen Produkte, die aus dem Sortiment genommen wurden, nicht mehr in den Auswahllisten zum Hinzufügen von Bestellpositionen auftauchen. Wenn Sie eine ältere Bestellung öffnen, die das ausgelaufene Produkt noch enthält, soll dieses dort jedoch noch erscheinen. Wie wir dies für ein und dasselbe Formular programmieren, zeigen wir unter dem Titel **Bestellungen mit gelöschten Produkten** ab Seite 24.

Schließlich decken wir in **Anwendungstitel zur Laufzeit einstellen** (ab Seite 11) und **Merkwürdiges Kombinationsfeld-Verhalten** (ab Seite 20) auf, warum manchmal Access-Optionen unerwartetes Verhalten verursachen. Unter **Icons in Access-Formularen und Berichten** erfahren Sie ab Seite 15, wie Sie Access-Objekte mit individuellen Icons ausstatten und unter **Bilder in MSysResources verwalten** (ab Seite 2) zeigen wir, wie Sie Bilddateien in der Datenbank verwalten.

Viel Spaß beim Ausprobieren!

Ihr André Minhorst

Bilder in MSysResources verwalten

Die Tabelle MSysResources ist relativ unbekannt. Kein Wunder, denn als Systemtabelle bleibt sie dem Benutzer in der Standardeinstellung verborgen, obwohl sie bereits seit Access 2010 in allen Datenbanken mit der Dateiendung .accdb sofort angelegt wird, wenn der Benutzer Objekte in der Datenbank anlegt. Dennoch ist sie sehr wichtig, denn sie speichert zum Beispiel die Bilddateien, die Sie Formularen in Bildsteuerelementen oder Schaltflächen zugewiesen haben. Genau um diese wollen wir uns in diesem Beitrag auch kümmern und eine Möglichkeit vorstellen, sich schnell einen Überblick über die enthaltenen Bilder zu verschaffen und schnell Bilder zu dieser Tabelle hinzuzufügen. Diese stehen dann zum Einfügen in Bildsteuerelemente und Schaltflächen bereit.

Systemtabellen einblenden

Wenn Sie Access öffnen, sehen Sie im Navigationsbereich erst einmal nur die Tabellen, die Sie selbst angelegt haben. Systemtabellen wie die Tabelle **MSysObjects** können Sie allerdings leicht sichtbar machen. Dazu klicken Sie mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs und wählen den Eintrag **Navigationsoptionen...** aus.

Im nun erscheinenden Dialog **Navigationsoptionen** finden Sie im Bereich **Anzeigeoptionen** die beiden Optionen **Ausgeblendete Objekte anzeigen** und **Systemobjekte** anzeigen (siehe Bild 1).

Diese aktivieren Sie beide, denn einige Systemtabellen sind gleichzeitig auch noch als ausgeblendete Objekte gekennzeichnet. Direkt nach dem Schließen erscheinen die Systemtabellen dann in hellgrauer Schrift im Navigationsbereich. Allerdings finden wir hier noch nicht die Tabelle **MSysResources** vor (siehe Bild 2).

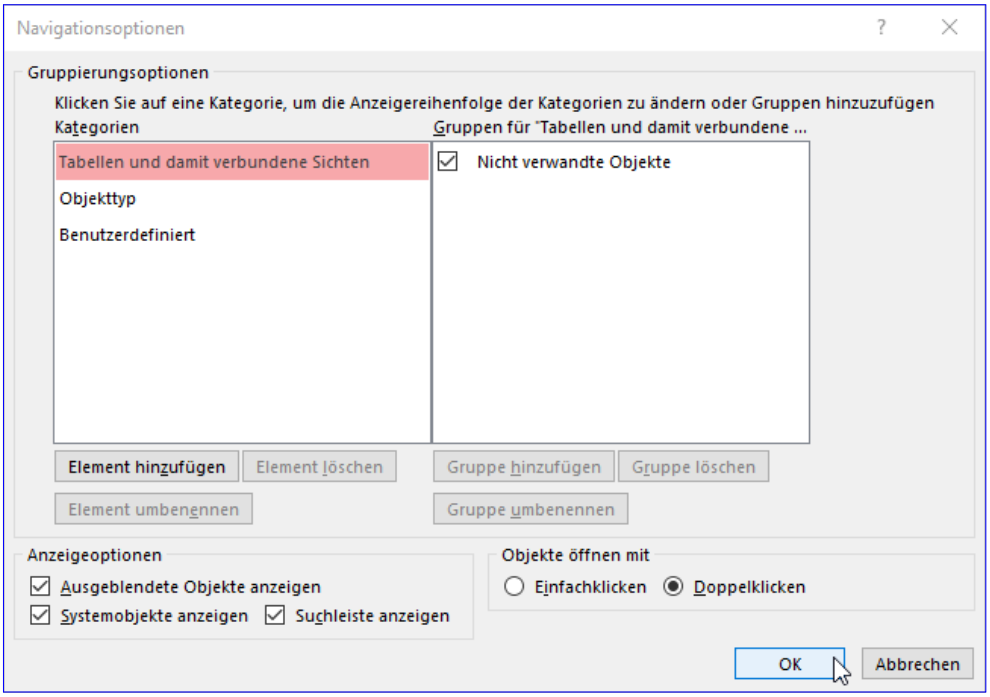


Bild 1: Einblenden von Systemtabellen

MSysResources erzeugen

Erst wenn Sie der Datenbank eine Tabelle hinzufügen – dazu reicht das Anlegen eines neuen Formulars im Entwurf –, und dann die **F5**-Schaltfläche zum Aktualisieren des Navigationsbereichs betätigen, zeigt dieser eine

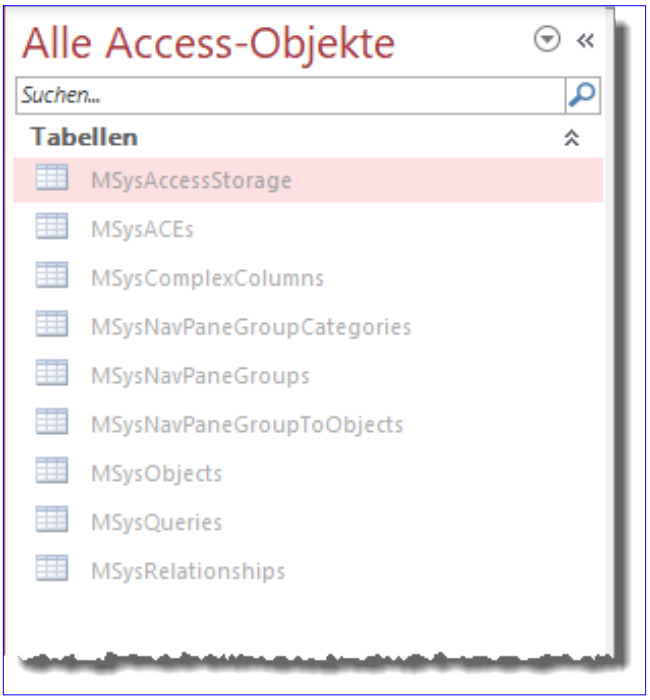


Bild 2: Systemtabellen einer neuen, leeren Access-Datenbank

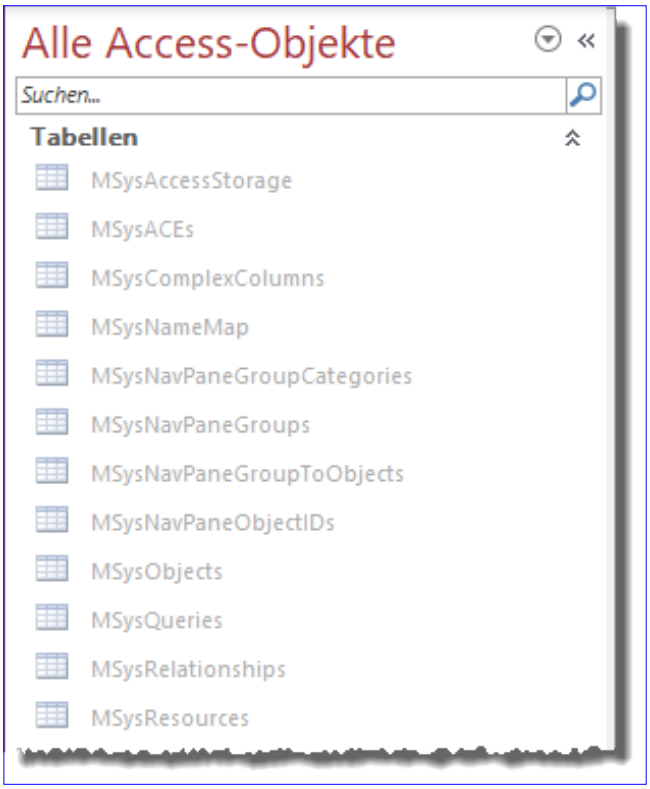


Bild 3: Systemtabellen nach dem Anlegen eines Formulars

weitere Tabelle an, nämlich die bereits erwähnte Tabelle **MSysObjects**. Wenn Sie gleich ein Formular anlegen, erscheinen sogar gleich zwei neue Tabellen in der Liste – **MSysResources** und **MSysNameMap** (siehe Bild 3).

MSysResources im Detail

Uns interessiert aber nur die Tabelle **MSysResources**. Diese sieht im Entwurf wie in Bild 4 aus. Das Anlage-Feld **Data** nimmt die Bild-Dateien und andere Inhalte auf. **Extension** gibt die Dateinamenerweiterung an. **ID** ist der Primärschlüsselindex. **Name** enthält den Namen

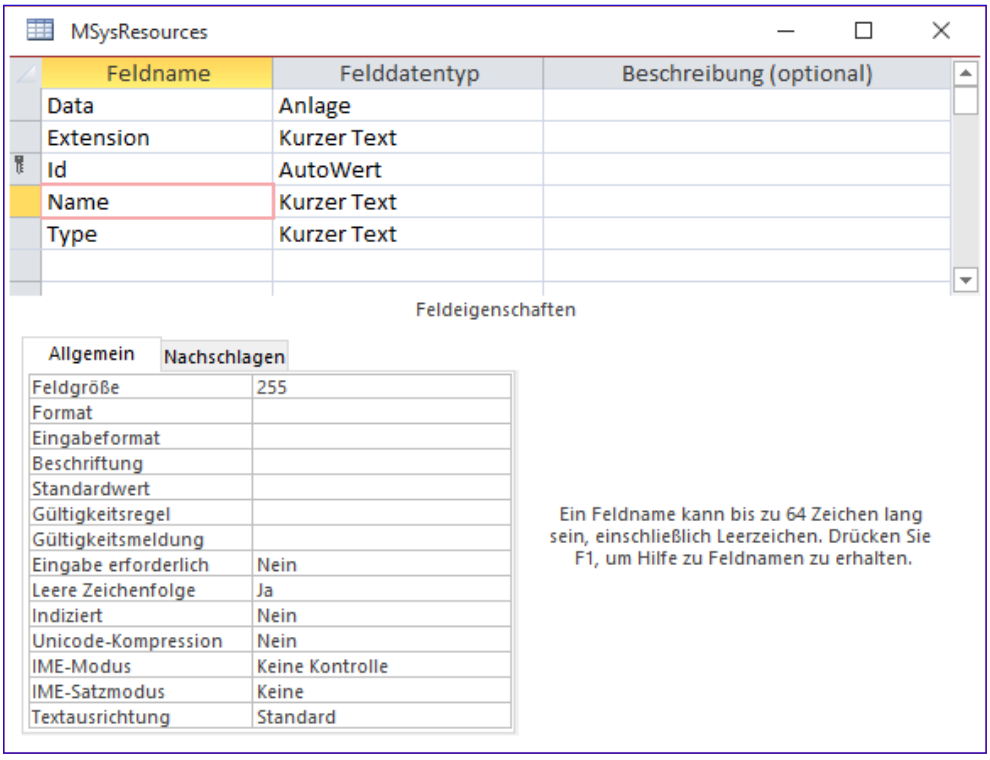


Bild 4: Entwurf der Tabelle **MSysResources**

der Datei ohne Dateiendung und **Type** den Typ, im Falle von Bilddateien immer **img**. Interessanterweise ist keines der Textfelder mit einem eindeutigen Index versehen, und es gibt auch keinen zusammengesetzten Index über die beiden Felder **Name** und **Extension**. Das bedeutet, dass Sie darauf achten müssen, nicht mehrere Bilddateien mit gleichem Namen zu speichern. Sonst kann es sein, dass Sie zwar die zuletzt angelegte Bilddatei in einem Steuerelement anzeigen wollen, Access Ihnen aber eine zuvor angelegte Bilddatei präsentiert.

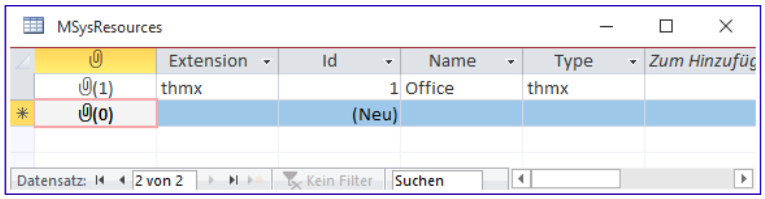


Bild 5: Die Tabelle **MSysResources** in der Datenblattansicht

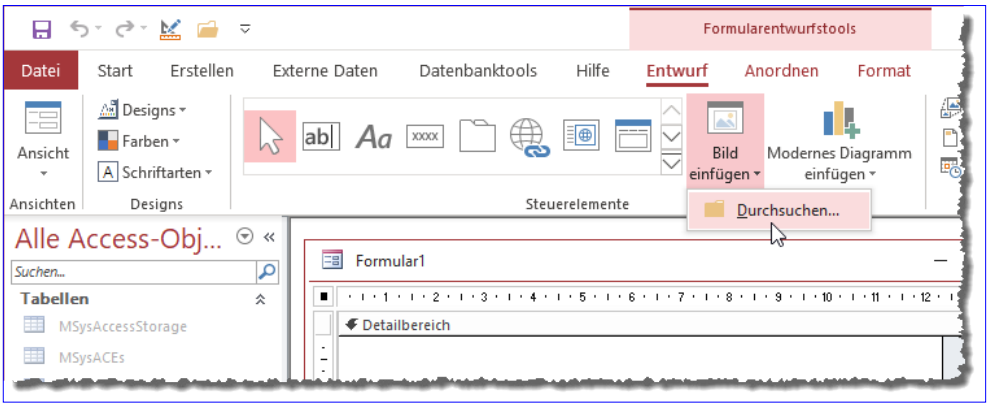


Bild 6: Hinzufügen eines Bildes

Wechseln wir in die Datenblattansicht dieser Tabelle, finden wir dort bereits einen Datensatz vor (siehe Bild 5).

Dabei handelt es sich um eine Theme-Datei, in der Informationen zur Formatierung von Formularen und Steuerelementen befinden.

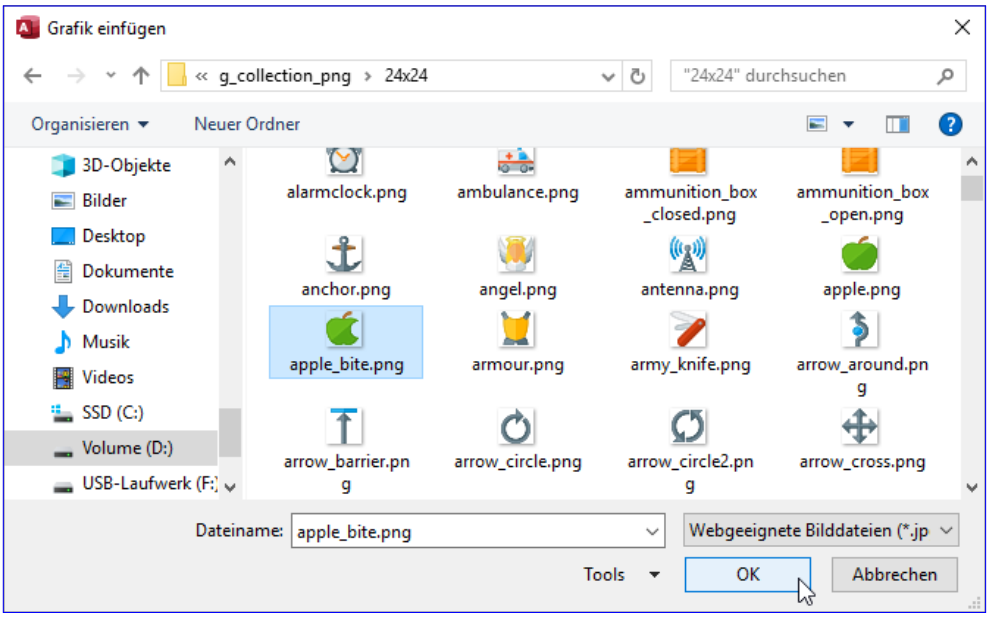


Bild 7: Auswählen der gewünschten Bilddatei

Bild hinzufügen per Ribbon

Es gibt verschiedene Möglichkeiten, Bilder in dieser Tabelle zu speichern. Der Weg über die Benutzeroberfläche gelingt wie folgt:

- Erstellen Sie ein neues, leeres Formular in der Entwurfsansicht.
- Wählen Sie im Ribbon den Befehl **Entwurf|Steuerelemente|Bild einfügen|Durchsuchen...** aus (siehe Bild 6).
- Selektieren Sie im nun erscheinenden Dialog **Grafik einfügen** das gewünschte Bild aus – beispielsweise ein Icon für eine Schaltfläche (siehe Bild 7).

- Das Bild erscheint nun in der Bildergalerie, wenn Sie diese erneut öffnen (siehe Bild 8).

- Damit brechen wir das Erstellen des Formulars ab – wir brauchen das Bild nicht zum Formular hinzuzufügen, damit es in der Datenbank gespeichert ist.

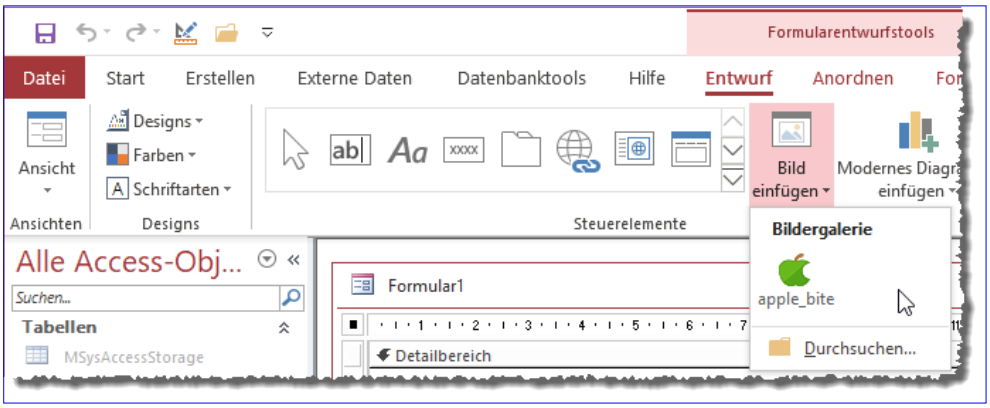


Bild 8: Das gewählte Bild erscheint nun in der Bildergalerie.

Wenn Sie nun die Tabelle **MSysResources** erneut öffnen beziehungsweise den Inhalt der noch offenen Tabelle mit **F5** aktualisieren, finden Sie einen neuen Eintrag vor (siehe Bild 9).

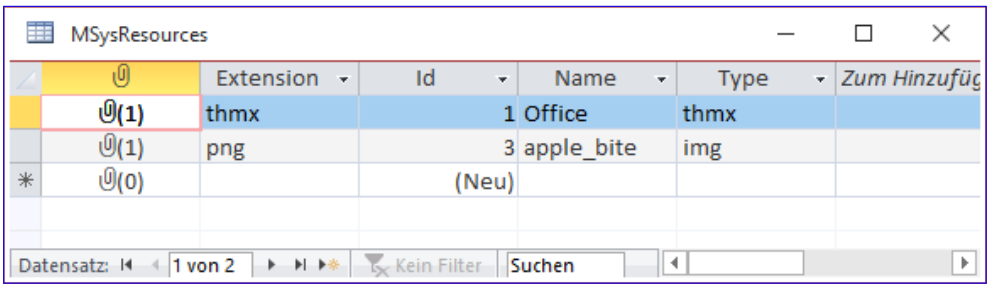


Bild 9: Das neue Bild in der Tabelle **MSysResources**

Bilder per MSysResources hinzufügen

Schauen wir uns den zweiten Weg an, Bilder zur Tabelle **MSysResources** hinzuzufügen. Dazu legen Sie direkt in der Tabelle einen neuen Datensatz an, indem Sie das Feld **Extension** mit dem Wert **png**, **Name** mit **baseball** und **Type** mit **img** füllen. Dann klicken Sie doppelt auf das Feld mit dem Büroklammer-Symbol und finden den Dialog aus Bild 10 vor.

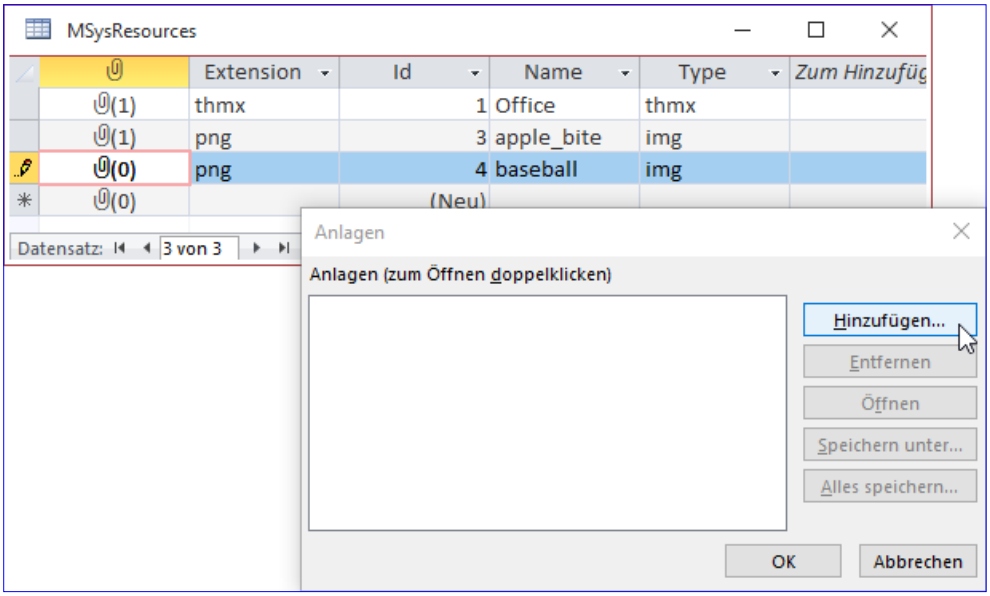


Bild 10: Hinzufügen einer Bilddatei über den **Anlagen**-Dialog

Hier klicken Sie auf die Schaltfläche **Hinzufügen...** und wählen im nun erscheinenden Dialog die dazu passende Bilddatei aus. Der Name der Bilddatei taucht dann in der Liste der Anlagen im Dialog **Anlegen** auf (siehe Bild 11). Nach dem Schließen des Dialogs erscheint im Feld

Anwendungstitel zur Laufzeit einstellen

Der Anwendungstitel ist der Text in der Titelzeile einer Windows-Anwendung. Dieser wird normalerweise von der Anwendung selbst ausgefüllt. Unter Access ergibt sich ein Sonderfall, weil es ja ein Container für eigene Datenbankanwendungen mit eigener Benutzeroberfläche ist. Deshalb möchte man auch einen entsprechenden Titel in der Titelleiste anzeigen. Die verschiedenen Möglichkeiten dazu stellen wir in diesem Beitrag vor.

In der aktuellsten Version der mit Office 365 gelieferten Access-Version zeigt Access in der Titelzeile einige hilfreiche Informationen an:

- den Dateinamen ohne Dateiendung,
- den Pfad zur Datei,
- das Dateiformat (aktuell **Access 2007 - 2016**) sowie
- den Text **Access** (siehe Bild 1).

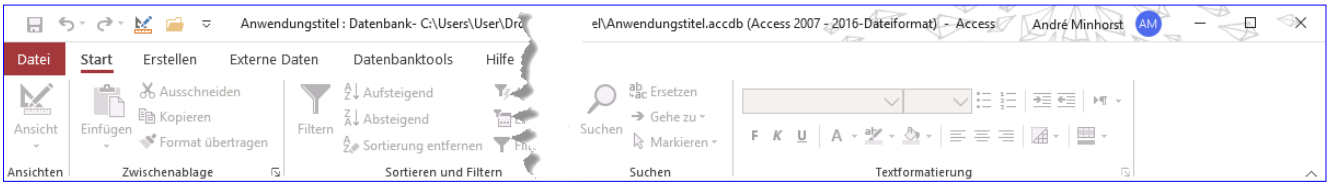


Bild 1: Anwendungstitel einer Access-Datenbank

Statischen Anwendungstitel festlegen

Wenn Sie einen statischen Anwendungstitel festlegen wollen, erledigen Sie das einfach über die Access-Optionen. Dazu klicken Sie im Ribbon auf **Datei|Optionen**. Es erscheint der Dialog **Access-Optionen**, wo Sie links zum Bereich **Aktuelle Datenbank** wechseln. Hier finden Sie unter Anwendungsoptionen die Eigenschaft **Anwendungstitel** vor, die wir hier auf den Wert **Statischer Anwendungstitel** ändern wollen (siehe Bild 2).

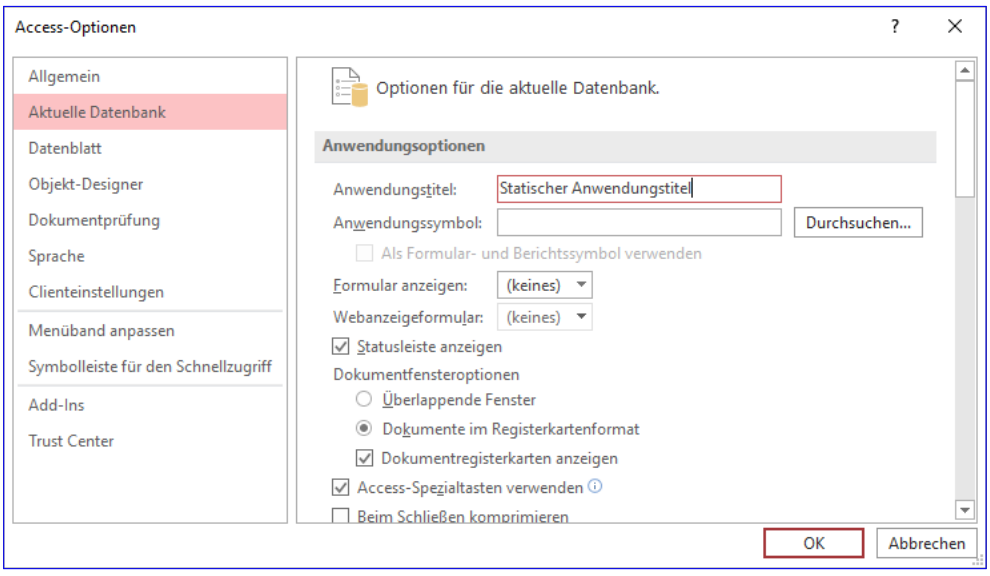


Bild 2: Anwendungstitel in den Access-Optionen festlegen

Schließen Sie den Dialog, wird der neue Titel direkt in der Titelleiste angezeigt (siehe Bild 3).

Dynamischer Anwendungstitel

Es gibt aber auch Anforderungen, die eine Einstellung des Anwendungstitels zur Laufzeit erfordern. Wenn Sie beispielsweise die Version der Anwendung in der Titelzeile

ausgeben wollen, die in einer Tabelle gespeichert ist, dann kommen Sie mit dem manuellen Ändern der Option **Anwendungstitel** nicht weit. Das gilt auch für andere Daten wie etwa den aktuellen Benutzernamen et cetera.

Diese Daten können Sie mit zwei verschiedenen Methoden in die Titelleiste schreiben:

- durch Ändern der Access-Option, die wir oben schon manuell geändert haben, per VBA oder
- per API-Funktion.

Beide Varianten schauen wir uns im Anschluss an.

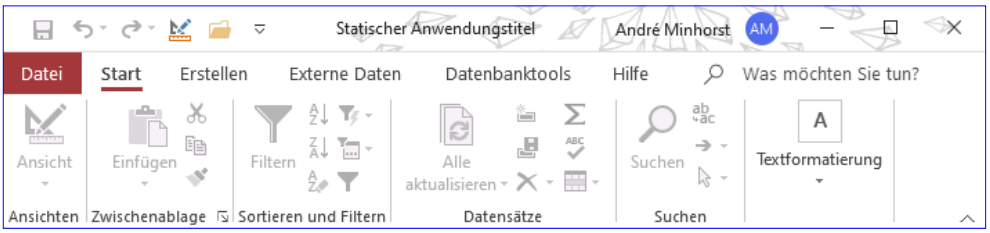


Bild 3: Neu festgelegter Anwendungstitel

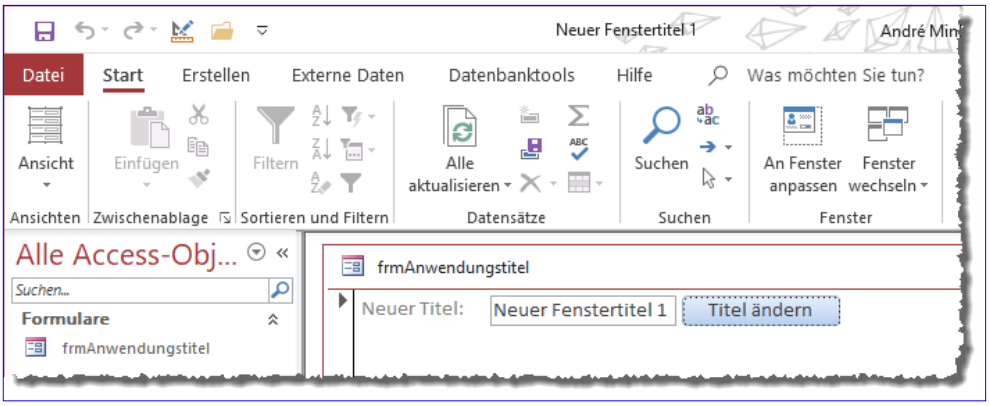


Bild 4: Anwendungstitel, per VBA festgelegt

Ändern der Access-Option Anwendungstitel per VBA

Wir erstellen ein kleines Beispielformular, in das Sie über ein Textfeld namens **txtTitel** den gewünschten Fenstertitel eingeben können (siehe Bild 4). Die Schaltfläche neben dem Textfeld löst die folgende Prozedur aus:

```
Private Sub cmdTitelAendern_Click()
    AnwendungstitelAendern Nz(Me!txtTitel, _
        "Kein Titel festgelegt")
End Sub
```

Die dadurch aufgerufene Prozedur hat den folgenden Code:

```
Public Sub AnwendungstitelAendern(strTitel As String)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Properties("AppTitle") = strTitel
    Application.RefreshTitleBar
End Sub
```

Die Option **Anwendungstitel** wird in einer Property namens **AppTitle** gespeichert, die wir über die **Properties**-Auflösung der aktuellen Datenbank einstellen können. Dazu benötigen wir einen Verweis auf das **Database**-Objekt der aktuellen Datenbank, den wir in der Variablen **db** speichern. Dem Eintrag **Properties("AppTitel")** können wir dann den neuen Titel zuweisen. Danach tut sich noch nichts, denn wir müssen die Anzeige noch aktualisieren. Dazu rufen wir die Methode **RefreshTitleBar** des **Application**-Objekts auf.

Option noch nicht vorhanden?

Wenn Sie die Option vor dem Aufruf dieser Prozedur noch nie über die Access-Optionen eingestellt haben, ist die Property **AppTitle** noch nicht vorhanden und die Prozedur

Icons in Access-Formularen und Berichten

Die Formulare in Access-Anwendungen lassen sich mittlerweile recht einfach mit Icons ausstatten. Nicht mit Bordmitteln gelingt jedoch das Anzeigen eines benutzerdefinierten Icons in der Titelleiste von Formularen und Berichten. Dabei wäre es doch ein optischer Leckerbissen, wenn sich die Formulare schon in der linken oberen Ecke voneinander unterscheiden würden – und wenn Sie die Objekte der Access-Datenbank als Registerkarten anzeigen, sind sie auch noch praktisch. In diesem Beitrag schauen wir uns zwei Möglichkeiten an, wie Sie das gewünschte und zum Formular passende Icon einbauen.

Individuelles Icon für Formulare

Ziel dieses Beitrags ist es, Formulare mit individuellen Icons auszustatten wie es in der Abbildung aus Bild 1 der Fall ist.

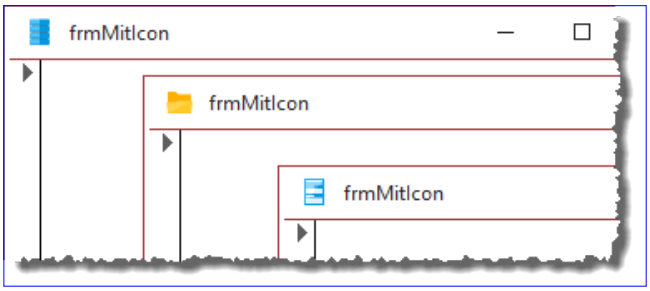


Bild 1: Formulare mit Icons

Dazu benötigen wir erst einmal einige API-Funktionen und Konstanten sowie eine Funktion, welche diese Elemente nutzt. Im Detail handelt es sich um die beiden API-Funktionen **LoadImage** und **SendMessage** (siehe Listing 1). Die **LoadImage**-Funktion lädt die **.ico**-Datei, die mit dem Parameter **lpsz** übergeben wird, in den Speicher und liefert ein Handle der Adresse zurück, an der sich das Bild dann befindet. Die übrigen Parameter nehmen beispielsweise den Bildtyp (**IMAGE_ICON**), die

```
Public Declare Function LoadImage Lib "user32" Alias "LoadImageA" (ByVal hInst As Long, ByVal lpsz As String, _
    ByVal un1 As Long, ByVal n1 As Long, ByVal n2 As Long, ByVal un2 As Long) As Long

Public Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As Long, _
    ByVal wParam As Long, lParam As Any) As Long

Public Const IMAGE_ICON = 1
Public Const LR_LOADFROMFILE = &H10
Public Const WM_SETICON = &H80
Public Const ICON_SMALL = 0

Public Function SetFormIcon(ByVal lngHWnd As Long, ByVal strIconFile As String) As Boolean
    Dim lngIcon As Long
    lngIcon = LoadImage(0, strIconFile, IMAGE_ICON, 16, 16, LR_LOADFROMFILE)
    If lngIcon <> 0 Then
        Call SendMessage(lngHWnd, WM_SETICON, ICON_SMALL, ByVal lngIcon)
    End If
End Function
```

Listing 1: Code zum Anzeigen von Icons aus dem Dateisystem

Abmessungen in Pixeln und die Art des Ladevorgangs entgegen (**LR_LOADFROMFILE**). Das Ergebnis ist ein Wert mit dem Datentyp **Long**.

Die Funktion **SendMessage** ist flexibel einsetzbar. In diesem Fall soll es das Handle auf das Icon zum Fenster mit einem bestimmten Handle schicken.

Das erledigen wir mit der Funktion **SetFormIcon**. Diese erwartet das Handle des Formulars, das mit dem neuen Icon ausgestattet werden soll, sowie den Pfad zur **.ico**-Datei als Parameter.

Dann ermittelt sie zuerst mit einem Aufruf der Funktion **LoadImage** das Handle auf die in den Speicher geladene Bilddatei und ruft dann, falls **hgIcon** nicht gleich **0** ist, die Funktion **SendMessage** auf. Dieser übergibt sie das Handle des Zielformulars, als Nachricht **WM_SETICON**, als Parameter **ICON_SMALL** und schließlich das Handle auf die Bilddatei.

Formular mit Icon ausstatten

Um das Formular über die Funktion **SetFormIcon** mit einem Formular auszustatten, bietet sich das Ereignis **Beim Laden** an. Wir fügen eine entsprechende Ereignisprozedur hinzu und legen dort die folgende Anweisung an:

```
Private Sub Form_Load()
    SetFormIcon Me.hwnd, CurrentProject.Path & "\form.ico"
End Sub
```

Damit wird die angegebene **.ico**-Datei in den Speicher geladen und dem Formular zugewiesen. Mit **CurrentProject.Path** geben wir an, dass sich die Datei im Verzeichnis der aktuellen Datenbank befindet.

Variante ohne .ico-Dateien im Dateisystem

Die Optik von Formularen gewinnt deutlich mit individuellen Icons. Der kleine Nachteil ist jedoch, dass sich die **.ico**-Dateien dafür im Verzeichnis der Datenbank befinden müssen (zumindest in unserem Beispiel – Sie können natürlich auch andere Verzeichnisse verwenden).

Auf dieses Verzeichnis hat der Benutzer in der Regel schreibenden Zugriff – und wenn das der Fall ist, kann er auch die **.ico**-Dateien versehentlich löschen.

Wir probieren eine Zwischenvariante aus, bei der wir die **.ico**-Dateien in der Tabelle **MSysResources** speichern, die auch für das Speichern von Bildern für Schaltflächen et cetera in Formularen verwendet wird.

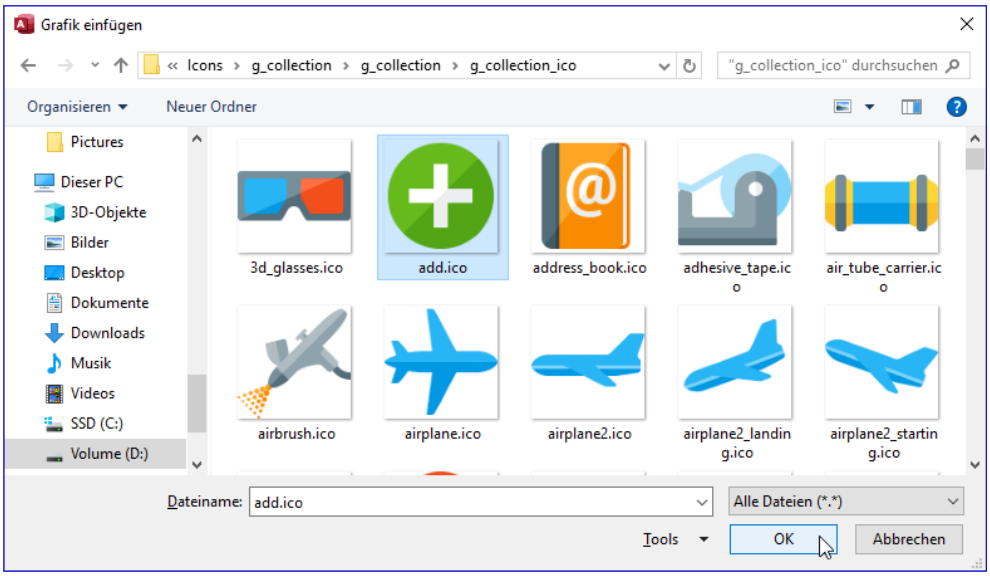


Bild 2: Icons zur Tabelle **MSysResources** hinzufügen

Wir schreiben die Bilder dann zur Laufzeit ins Verzeichnis der Datenbank, damit diese sicher vorhanden sind, wenn das Formular diese anfragt.

Für diese Methode benötigen wir eine weitere Funktion – aber zunächst wollen wir die gewünsch-

ten Bilddateien in die Tabelle **MSysResources** einfügen. Das geht am einfachsten über einen eingebauten Weg: Dazu öffnen Sie ein Formular in der Entwurfsansicht und zeigen dann mit dem Ribbon-Befehl **Entwurf|Steuerelemente|Bild einfügen|Durchsuchen...** einen **Grafik einfügen**-Dialog an.

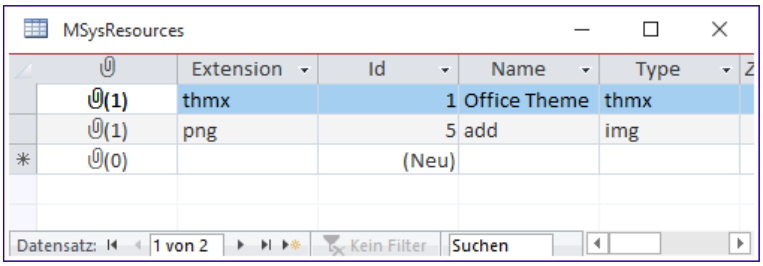


Bild 3: Die Bilddatei in der Tabelle **MSysResources**

Mit diesem wählen Sie dann die **.ico**-Datei aus, die Sie als Formular-Icon anzeigen möchten (siehe Bild 2).

Wir schauen uns das Ergebnis in der Tabelle **MSysResources** an. Diese finden Sie, wenn Sie in den Navigationsoptionen des Navigationsbereiches (Kontextmenü der Titelleiste des Navigationsbereiches, Eintrag **Navigationsoptionen...**) die Optionen **Ausgeblendete Objekte anzeigen** und **Systemobjekte anzeigen** aktivieren.

Öffnen Sie diese Tabelle, finden Sie zwar einen Eintrag vor, in dessen Feld **Name** die Bezeichnung **add** steht. Allerdings enthält das Feld **Extension** den Ausdruck **png** (siehe Bild 3).

Wenn Sie die Datei extrahieren, indem Sie doppelt auf das Anlage-Feld klicken und im Dialog **Anlegen** die Schaltfläche **Speichern unter...** betätigen (siehe Bild 4), finden Sie anschließend im Windows Explorer eine **.png**-Datei vor.

Und diese können wir leider nicht als Quelle für das Icon eines Formulars verwenden – hier werden nur **.ico**-Dateien akzeptiert. Es gibt jedoch noch einen Ausweg: Dabei fügen wir die **.ico**-Datei einfach ebenfalls über den **Anlagen**-Dialog zu einem neuen Datensatz in der Tabelle **MSysResources** hinzu.

Tragen Sie dann für das Feld **Extension** den Wert **ico** ein, für **Name** den Namen der Datei ohne Erweiterung und für das Feld **Type** den Wert **img** (siehe Bild 5).

.ico-Datei extrahieren und als Icon im Formular nutzen

Nun benötigen Sie eine Funktion, welche die gewünschte Datei aus der Tabelle **MSysResources** in das Dateisystem exportiert.

Dieser Funktion heißt **MSysResourcesExportieren** und ist in Listing 2 zu finden.

Die Prozedur erwartet die folgenden Parameter:

- **strZieldatei**: Pfad der zu erstellenden Datei

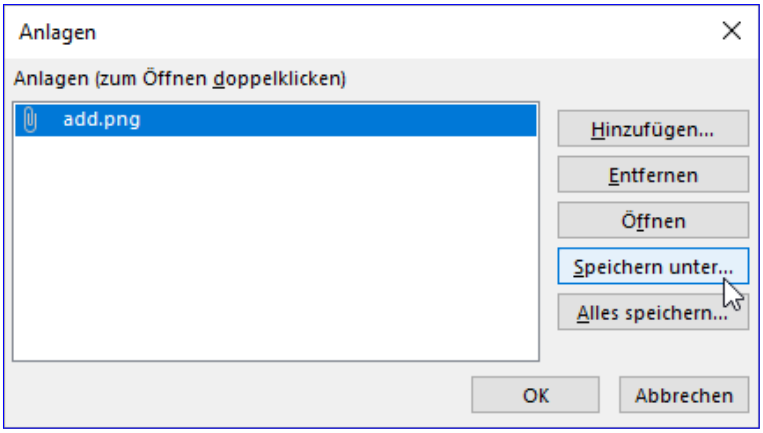


Bild 4: Extrahieren der Bilddatei aus dem Anlagefeld

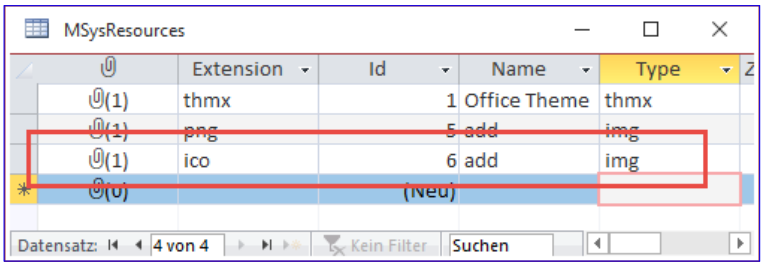


Bild 5: Die **.ico**-Datei in der Tabelle **MSysResources**

Merkwürdiges Kombinationsfeld-Verhalten

Das Problem, das sich hinter dem Titel verbirgt, ist möglicherweise nicht auf das Verhalten von Kombinationsfeldern beschränkt – es ist jedoch der Punkt, an dem es uns dank einer Leseranfrage aufgefallen ist. Diese enthielt die Frage, warum die »Automatisch ergänzen«-Funktion für Kombinationsfelder nicht mehr funktionierte. Das haben wir uns einmal genauer angeschaut und waren verblüfft über die Lösung: Eine Access-Option hat uns einen Strich durch die Rechnung gemacht.

Um das Problem nachzustellen (wenn es in Ihrer aktuellen Access-Konfiguration besteht), erstellen Sie eine Tabelle namens **tblAnreden** mit den beiden Feldern **ID** (Primärschlüsselfeld mit Autowert) und **Anrede** (Textfeld) und füllen diese mit zwei Datensätzen mit den Werten **Herr** und **Frau**.

Dann erstellen Sie ein neues, leeres Formular und fügen diesem ein Kombinationsfeld hinzu. Stellen Sie für die Eigenschaft **Datensatzherkunft** die soeben erstellte Tabelle **tblAnreden** ein. Legen Sie für die beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** die Werte **2** und **0cm** fest.

Wenn Sie nun das Formular in der Formularansicht anzeigen und den ersten Buchstaben eines der Einträge eingeben, sollte das Kombinationsfeld wie in Bild 1 reagieren und den Inhalt automatisch ergänzen.

Sollte Ihr Access-System jedoch eine bestimmte Einstellung aufweisen, dann geschieht genau das nicht – dann müssen Sie den Wert wie in Bild 2 komplett selbst eingeben oder halt per Kombinationsfeld auswählen.

Was die Lösung erschwert hat: Es konnte kein allgemeiner Bug sein, denn auf all meinen anderen Datenbanken hat das Kombinationsfeld wie erwartet funktioniert.

Die Suche nach der Lösung

Wie findet man nun die Lösung für ein solches Problem? In Kurzform: Als erstes prüft man natürlich, ob die naheliegenden Eigenschaften unterschiedlich sind – also zum

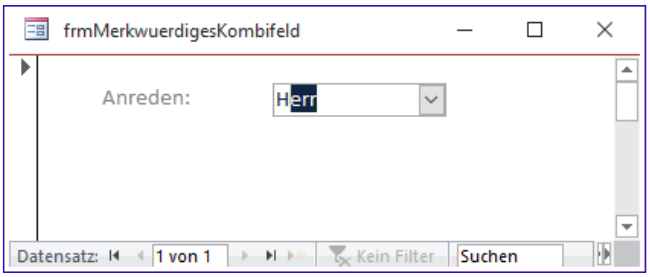


Bild 1: So soll die automatische Ergänzung funktionieren.

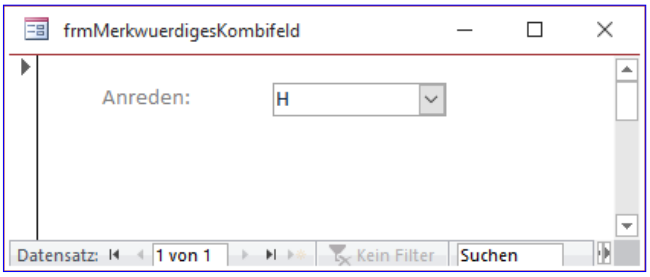


Bild 2: Unter bestimmten Umständen funktioniert es jedoch nicht.

Beispiel ob **Automatisch ergänzen** überhaupt aktiviert ist. Das war jedoch der Fall.

Da die Kombinationsfelder in allen Datenbanken funktionierten außer in der meines Kunden, habe ich erst einmal ein einfaches Formular wie das zuvor beschriebene mit dem Kombinationsfeld zur Auswahl der Anreden erst in der Anwendung des Kunden erstellt und dann in einer meiner eigenen Anwendungen.

Das Verhalten blieb gleich: In der Kundenanwendung keine automatische Ergänzung, in meinen Anwendungen schon.

Dann habe ich die Formulare der beiden Anwendungen mit der folgenden Anweisung in Textdateien exportiert:

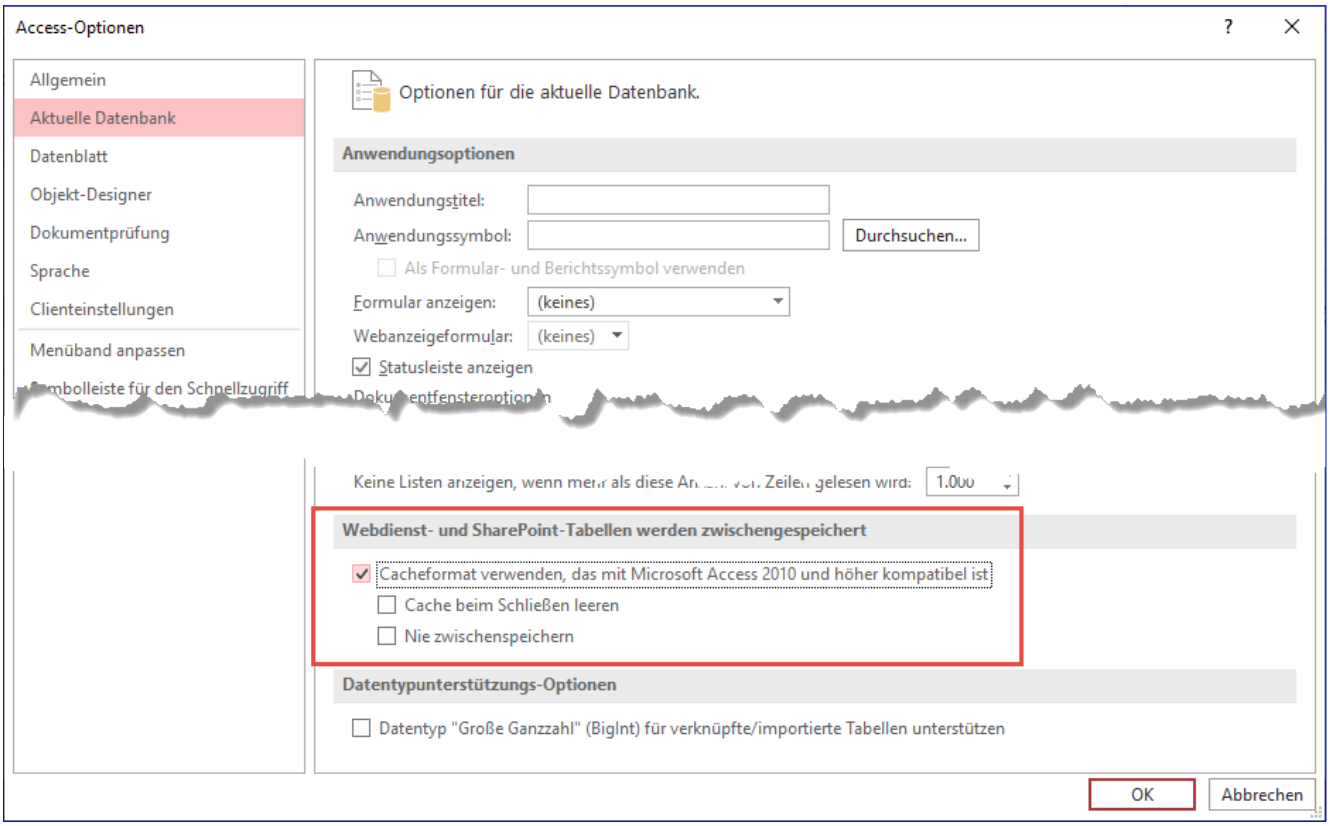


Bild 3: Diese Option führte kurzzeitig zum Erfolg.

```
Application.SaveAsText acForm, "<Formularname>", "<Pfad>"
```

Die so entstandenen Textdateien habe ich dann verglichen und keine nennenswerten Unterschiede gefunden.

Danach habe ich mir überlegt, welche Eigenschaften einer Datenbank sich noch auf das Verhalten von Steuerelementen auswirken könnten und bin auf die Access-Optionen für die aktuelle Datenbank gekommen.

Diese habe ich dann geöffnet und ein wenig herumexperimentiert. Ich dachte erst, ich hätte die Lösung, als ich die Eigenschaft **Cacheformat verwenden, das mit Microsoft Access 2010 und höher kompatibel ist** deaktiviert habe (siehe Bild 3).

Direkt danach funktionierte die automatische Ergänzung wieder! Allerdings nur solange, bis ich die Datenbank geschlossen und wieder geöffnet habe. Auch wenn danach die oben genannte Option immer noch deaktiviert war, blieb die automatische Ergänzung aus. Der Verdacht, dass das Verhalten etwas mit den Optionen der Access-Datenbank zu tun hatte, erhärtete sich.

Allerdings brachten Experimente mit den übrigen Eigenschaften aus dem Bereich **Aktuelle Datenbank** keine neuen Erkenntnisse. Also holte ich die Holzhammer-Methode heraus. Ich wollte alle Optionen leeren beziehungsweise die, hinter denen eine Property der aktuellen Datenbank stand, löschen. Viele der Optionen kann man über die **Properties**-Auflistung des **Database**-Objekts der aktuellen Datenbankdatei einstellen.

Dazu verwendete ich die folgende Prozedur. Diese deklariert zunächst eine Variable für das aktuelle **Database**-Objekt namens **db** und eine Laufvariable namens **i**:

Bestellungen mit gelöschten Produkten

In einer Bestellverwaltung verwalten Sie Kunden, Bestellungen und Bestellpositionen mit den jeweiligen Produkten. Gelegentlich werden Produkte aus dem Programm genommen. Das ist für neue Bestellungen kein Problem – die Auswahl des Produkts einer Bestellposition kann auf vorhandene Produkte beschränkt werden. Aber wie gehen wir vor, wenn wir bereits ausgelieferte Bestellungen betrachten wollen, die Bestellpositionen mit Produkten enthalten, die nicht mehr im Programm sind? Wie das gelingt, zeigen wir in diesem Beitrag.

Datenmodell der Beispielanwendung

Die Beispielanwendung zu diesem Beitrag enthält die üblichen Tabellen einer Bestellverwaltung mit den mindestens notwendigen Feldern, deren Datenmodell Sie in Bild 1 sehen.

Interessant ist hier vor allem die Tabelle **tblProdukte**. Die Produkte werden in der Tabelle **tblBestellpositionen** zu einer Bestellung aus der Tabelle **tblBestellungen** hinzugefügt. Hinzu kommen in der Tabelle **tblBestellpositionen** noch andere Felder, um die zum Zeitpunkt der Bestellung gültigen Werte für den Einzelpreis und den Mehrwertsteuersatz festzuhalten.

Beispielformular zur Verwaltung von Bestellungen

Das Zusammenstellen einer Bestellung erledigen wir im Formular **frmBestellungDetail**, das an die Tabelle **tblBestellungen** gebunden ist und alle Felder dieser Tabelle im Detailbereich anzeigt.

Außerdem enthält es ein Unterformular namens **sfmBestellungDetail**. Dieses ist an die Tabelle **tblBestellpositionen** gebunden und zeigt die Felder **ProduktID**, **Einzelpreis**, **Mehrwertsteuersatz** und **Menge** in der Datenblattansicht an. Damit es immer die Bestellpositionen zu der im Hauptformular angezeigten Bestel-

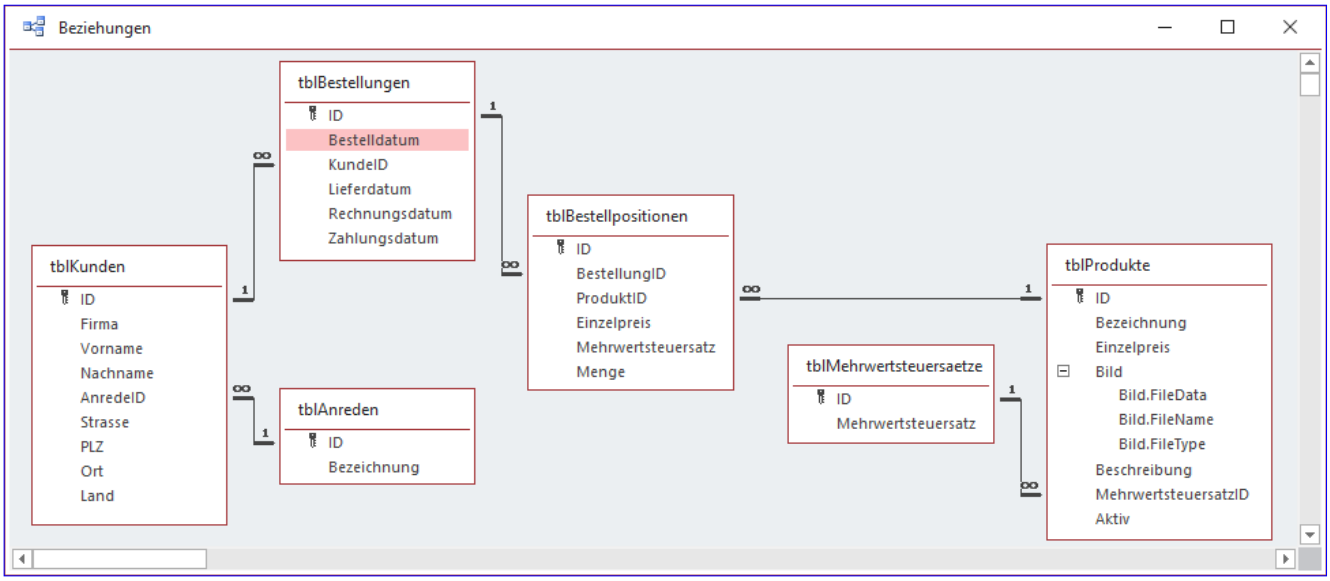


Bild 1: Beziehungsfenster des Datenmodells der Beispieldatenbank

lung liefert, erhält die Eigenschaft **Verknüpfen von** des Unterformular-Steuerelements den Wert **BestellungID** und die Eigenschaft **Verknüpfen nach** den Wert **ID**. Dies sieht im Entwurf wie in Bild 2 aus.

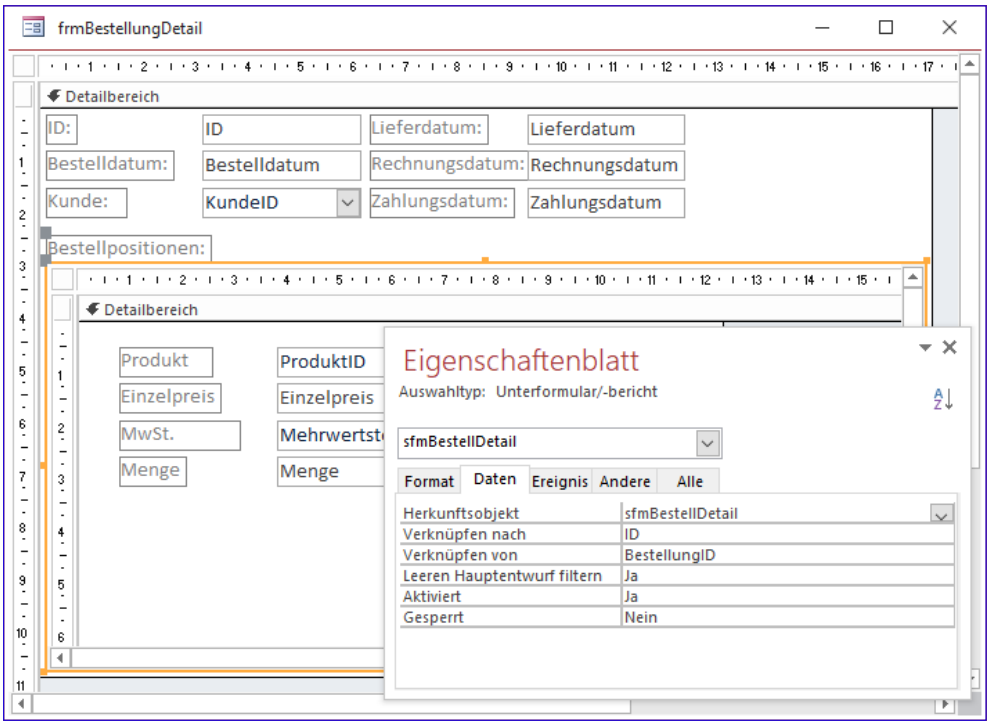


Bild 2: Entwurf des Formulars **frmBestellungDetail**

Wenn Sie in die Formularansicht wechseln, können Sie nach dem Anlegen eines Datensatzes in der Tabelle **tblBestellungen** über das Hauptformular damit beginnen, die Bestellpositionen hinzuzufügen.

Dazu wählen Sie einfach aus dem Kombinationsfeld **ProduktID** das gewünschte Produkt aus (siehe Bild 3).

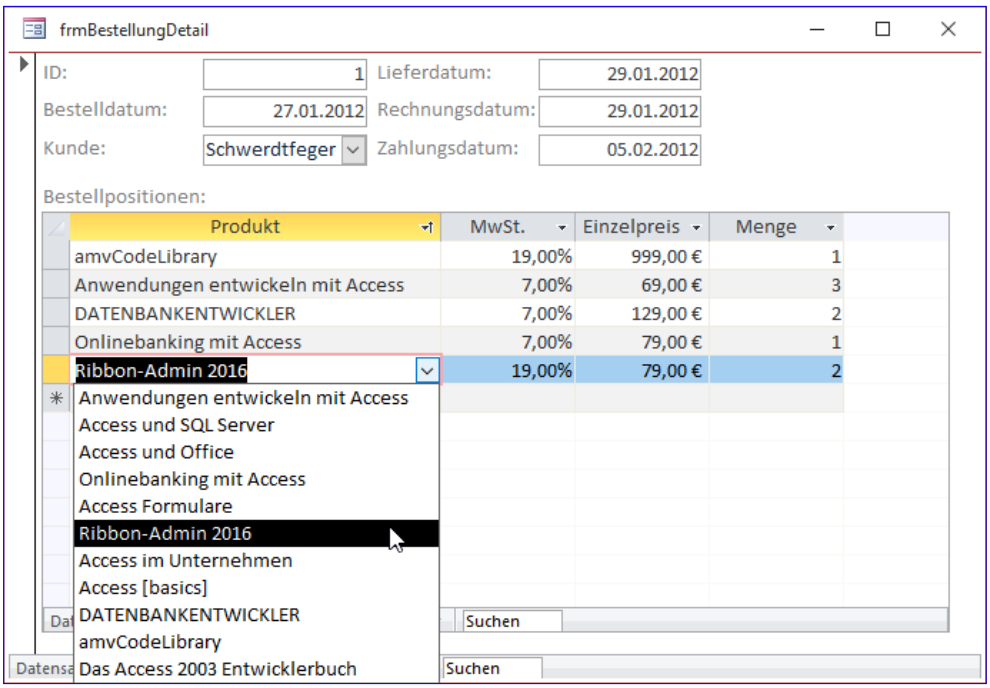


Bild 3: Formularansicht des Formulars **frmBestellungDetail**

Damit die Werte für die Felder **Mehrwertsteuersatz** und **Einzelpreis** aus der Tabelle **tblProdukte** beziehungsweise **tblMehrwertsteuersaetze** entnommen und direkt in den neuen Datensatz der Tabelle **tblBestellpositionen** eingetragen werden, legen wir für das Ereignis **Nach Aktualisierung** des Kombinationsfeldes eine Ereignisprozedur an. Diese sieht wie in Listing 1 aus. Die Prozedur ermittelt per **DLookup**-Funktion den Einzelpreis für das hinzugefügte Produkt aus der Tabelle **tblProdukte**. Danach liest sie, ebenfalls per **DLookup**, den Wert des Feldes **MehrwertsteuersatzID** der Tabelle **tblProdukte** ein. Da wir aber

```
Private Sub ProduktID_AfterUpdate()
    Dim lngMehrwertsteuersatzID As Long
    Me!Einzelpreis = DLookup("Einzelpreis", "tblProdukte", "ID = " & Me!ProduktID)
    lngMehrwertsteuersatzID = DLookup("MehrwertsteuersatzID", "tblProdukte", "ID = " & Me!ProduktID)
    Me!Mehrwertsteuersatz = DLookup("Mehrwertsteuersatz", "tblMehrwertsteuersaetze", "ID = " & lngMehrwertsteuersatzID)
End Sub
```

Listing 1: Einfügen von Mehrwertsteuersatz und Einzelpreis nach dem Einfügen einer Bestellposition

den Mehrwertsteuersatz in der Tabelle **tblBestellpositionen** nicht als Fremdschlüsselfeld auf einen Eintrag der Tabelle **tblMehrwertsteuersaetze** anlegen wollen, sondern den Prozentsatz speichern wollen, müssen wir diesen anschließend noch aus der Tabelle **tblMehrwertsteuersaetze** auslesen.

Angezeigte Produkte einschränken

Nun wollen wir Produkte, die nicht mehr verfügbar sind, nicht mehr im Kombinationsfeld des Unterformulars **sfmBestellungDetail** auflisten. Deshalb haben wir vorsorglich schon einmal ein Feld namens **Aktiv** in den Entwurf der Tabelle **tblProdukte** eingearbeitet (siehe Bild 4). Dieses stellen standardmäßig auf den Wert **Ja** ein, damit neu angelegte Produkte direkt als aktiviert markiert sind.

Für Produkte, die nicht mehr aktiv sein sollen, müssen wir dann manuell den Wert des Feldes **Aktiv** auf **Nein** einstellen (siehe Bild 5).

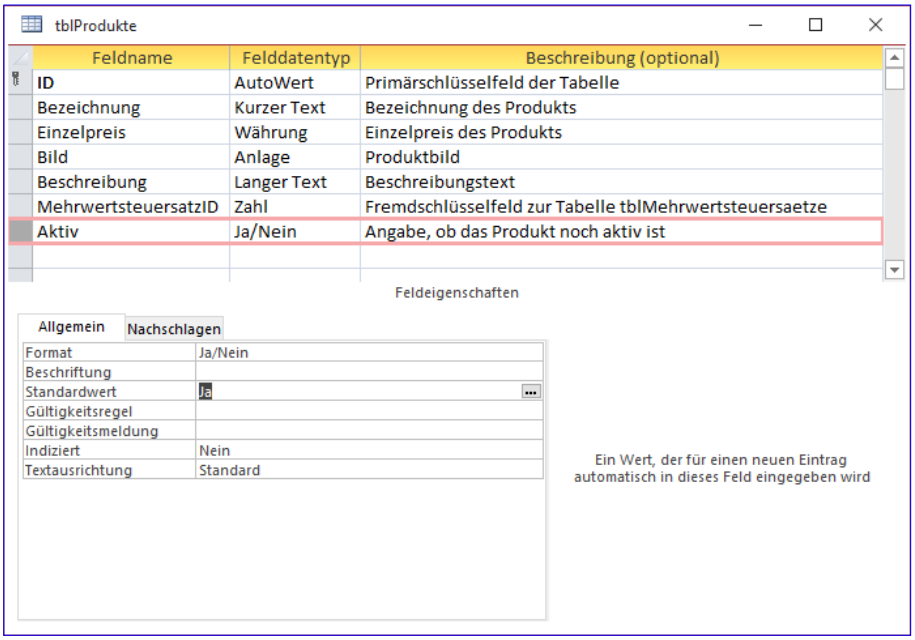


Bild 4: Entwurf der Tabelle **tblProdukte** mit dem Feld **Aktiv**

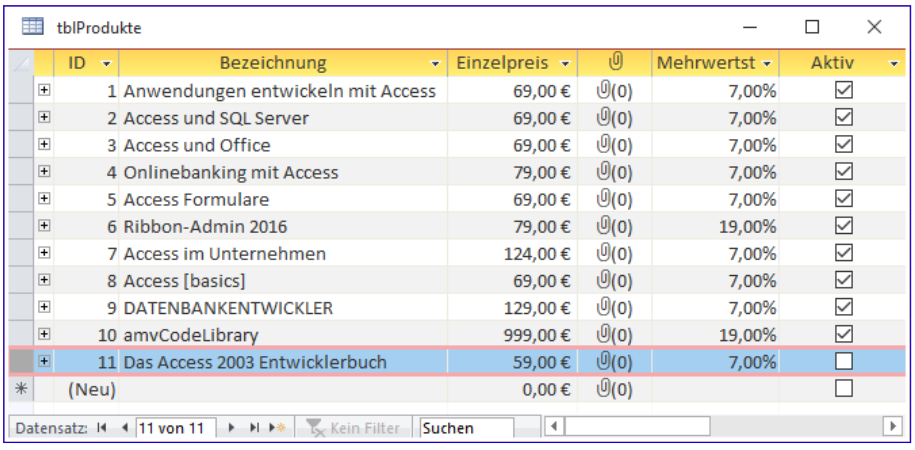


Bild 5: Einstellen der Werte des Feldes **Aktiv**

Damit können wir nun die Datensatzherkunft des Kombinationsfeldes anpassen. Der bereits vorhandenen Abfrage, die durch die Übernahme der Konfiguration des Nachschlagefeldes der Tabelle **tblBestellpositionen** für

das Kombinationsfeld angelegt wurde, fügen wir das Feld **Aktiv** aus der Tabelle **tblProdukte** hinzu und legen dafür als Kriterium den Vergleichswert **Wahr** fest (siehe Bild 6).

Wenn Sie dann im Formular **frmBestellungDetail** in die Formularansicht wechseln, taucht ein als nicht mehr aktiv markierter Eintrag nicht mehr im Kombinationsfeld auf (siehe Bild 7).

Wechseln Sie dann allerdings zu einem Datensatz, der das nicht mehr aktive Produkt bereits in einer Bestellposition enthält, wird dieses nicht mehr im Kombinationsfeld im Unterformular angezeigt (siehe Bild 8).

Wir müssen uns also überlegen, wie wir erkennen, ob gerade ein Datensatz im Formular **frmBestellungDetail** angezeigt wird, der bereits versendet ist, und in diesem Fall die Datensatzherkunft des Kombinationsfeldes zur Auswahl der Produkte anpasst.

Das können wir beispielsweise erledigen, indem wir das Lieferdatum abfragen. Ist dieses leer, sollen nicht mehr aktive Produkte noch angezeigt werden.

Anderenfalls ist die Bestellung noch nicht als geliefert markiert und es ist korrekt, wenn das Produkt nicht mehr auswählbar ist. Wir realisieren wir das?

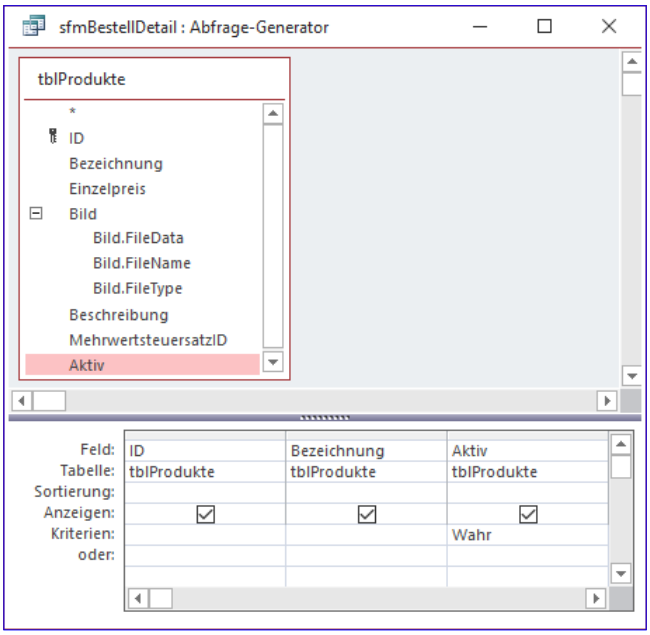


Bild 6: Neue Datensatzherkunft für das Kombinationsfeld zur Auswahl des Produkts einer Bestellposition

Ganz einfach: Wir legen eine Ereignisprozedur an, die beim Wechseln des Datensatzes im Hauptformular ausgelöst wird und die den Wert des Feldes **Lieferdatum** ermittelt. Wenn das Feld **Lieferdatum** den Wert **Null** hat, sollen nur die aktiven Produkte angezeigt werden. Wenn das Feld

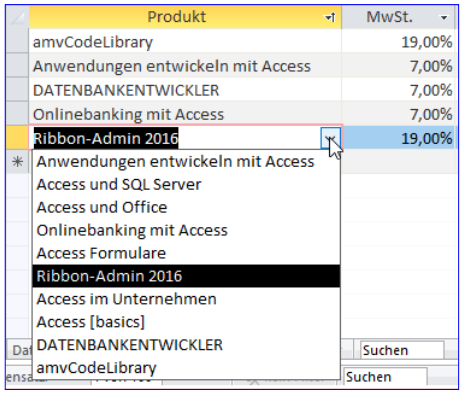


Bild 7: Das als nicht aktiv markierte Produkt taucht nicht mehr in der Liste auf.

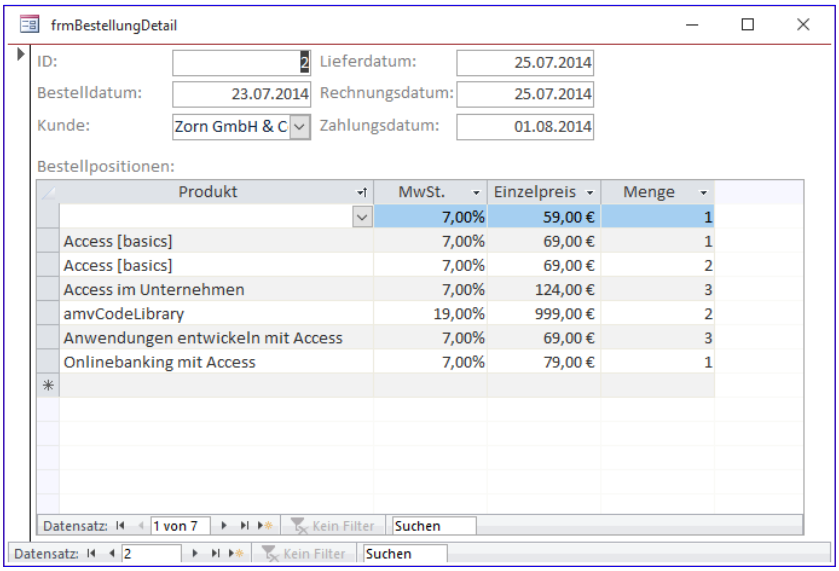


Bild 8: Ist das nicht mehr aktive Produkt bereits in einer Bestellung vorhanden, wird es allerdings nicht mehr im Kombinationsfeld angezeigt.

```
Private Sub Form_Current()
    Dim strSQL As String
    If IsNull(Me!Lieferdatum) Then
        strSQL = "SELECT ID, Bezeichnung, Aktiv FROM tblProdukte WHERE Aktiv=True"
    Else
        strSQL = "SELECT ID, Bezeichnung, Aktiv FROM tblProdukte"
    End If
    Me!sfmBestellDetail.Form.ProduktID.RowSource = strSQL
End Sub
```

Listing 2: Ein- und Ausblenden der nicht mehr aktiven Artikel aus der Datensatzherkunft des Kombinationsfeldes zur Auswahl der Produkte

Lieferdatum nicht **Null** ist, soll das Kombinationsfeld alle Produkte liefern (siehe Listing 2).

Wenn Sie es genau nehmen, müssen Sie auch für das Feld zur Eingabe des Lieferdatums eine Ereignisprozedur vorsehen, die nach der Aktualisierung des enthaltenen Wertes die Datensatzherkunft des Kombinationsfeldes aktualisiert.

Also lagern wir die Anweisungen der Prozedur **Form_Current** in eine eigene Prozedur namens **ProdukteEinAusblenden** aus, die wie in Listing 3 aussieht. Die beiden Prozeduren, die durch das Ereignis **Beim Anzeigen** des Hauptformulars und durch das Ereignis **Nach Aktualisierung** des Textfeldes **Lieferdatum** ausgelöst werden, sehen so aus:

```
Private Sub Form_Current()
    ProdukteEinAusblenden
End Sub

Private Sub Lieferdatum_AfterUpdate()
    ProdukteEinAusblenden
End Sub
```

Zwischenstand

Damit haben wir nun folgendes erreicht: Wenn der Benutzer einen Bestelldatensatz anzeigt, dessen Lieferdatum bereits gesetzt ist, enthält das Kombinationsfeld **ProduktID** noch die nicht aktiven Datensätze, sodass diese noch angezeigt werden können.

Wenn er zu einem Datensatz wechselt, der eine noch nicht gelieferte Bestellung enthält, werden nicht mehr aktive Produkte nicht mehr zur Auswahl im Kombinationsfeld angeboten.

Was aber geschieht, wenn eine Bestellung angelegt wird, die ein Produkt enthält, das vor dem Eintragen des Lieferdatums als inaktiv markiert wird? Wenn die Bestellung nicht direkt nach dem Anlegen versendet wird, dann

```
Private Sub ProdukteEinAusblenden()
    Dim strSQL As String
    If IsNull(Me!Lieferdatum) Then
        strSQL = "SELECT ID, Bezeichnung, Aktiv FROM tblProdukte WHERE Aktiv=True"
    Else
        strSQL = "SELECT ID, Bezeichnung, Aktiv FROM tblProdukte"
    End If
    Me!sfmBestellDetail.Form.ProduktID.RowSource = strSQL
End Sub
```

Listing 3: Ausgelagerte Anweisungen zum Anpassen der Datensatzherkunft des Kombinationsfeldes zum Auswählen der Produkte

SQL Server-Security, Teil 1: Zugriffsschutz in Access

Schützen Sie Ihre Daten vor unerlaubtem Zugriff? Wenn Sie es nicht tun, gibt es dafür bestimmt Gründe. Nur sind es mit hoher Wahrscheinlichkeit keine guten Gründe. Nicht selten liegt es an fehlendem Fachwissen oder einer falschen Einschätzung der Daten oder es ist schlicht und ergreifend Bequemlichkeit. Die Sache mit der Bequemlichkeit sollten Sie selbst hinterfragen, das Fachwissen vermittelt Ihnen dieser wie die noch folgenden Artikel der Serie »Access und SQL Server-Security«.

»Die Daten dieser Datenbank sind nicht wichtig.« Diese Aussage basiert meist auf einer falschen Einschätzung der Daten. Es gibt in Unternehmen keine unwichtigen Daten. Es mag zwar Daten geben, die nicht unternehmenskritisch sind, aber unwichtige Daten? Hand aufs Herz, haben Sie Zeit oder gar Budget für unwichtige Daten? In der Regel sind alle Daten eines Unternehmens relevant und somit auch vor fremden Zugriff zu schützen. Dies hat inzwischen auch der Gesetzgeber verstanden. In den letzten Jahren sind verschiedene Gesetze in Kraft getreten, die entsprechende Zugriffskontrollen einfordern, sei es der Datenschutz nach der DSGVO und dem neuen BDSG oder das IT-Sicherheitsgesetz.

Es ist schon erstaunlich, dass es Gesetze braucht, um Unternehmen darauf hinzuweisen, dass ihre Daten und somit ihr Wissen schützenswert sind. Ob nun Produktionspläne, Forschungsergebnisse, Rezepturen, Patente, erfolgreiche Marketingaktionen, Kundenanalysen, Forecasts etc., bei all diesen Informationen handelt es sich um Wissen – relevantes Wissen für den täglichen Betrieb und zur Sicherung des Wettbewerbsvorteils eines Unternehmens.

Gelangen diese Daten in falsche Hände, stellt dies in vielen Fällen ein indirektes oder gar direktes Risiko für das Unternehmen dar. Deshalb sollte jeder im Unternehmen, vom Geschäftsführer bis zum Angestellten, darauf achten, wer zu welchem Zweck Zugriff auf die Unternehmensdaten hat.

Zugriffsschutz in Access

Es sieht dürftig aus. Access bietet als Zugriffsschutz lediglich das Datenbank-Kennwort. Zwar gibt es immer noch die Möglichkeit einer MDW-Datei, doch funktioniert diese nur mit dem veralteten Datenbankformat MDB.

Das Datenbank-Kennwort verschlüsselt die Access-Datenbank. Dabei ist es empfehlenswert, ein Kennwort zu verwenden, das aus Buchstaben, Zahlen und Sonderzeichen besteht und mindestens zehn Zeichen lang ist.

Es sollte aber auch nicht länger als 19 Zeichen sein, wenn es sich bei der Datenbank um ein Backend handelt, deren Tabellen in ein Access-Frontend eingebunden werden. Zwar akzeptiert das Backend durchaus mehr als 19 Zeichen, jedoch wird ein solches Kennwort beim Einbinden der Tabellen in das Frontend mit der Meldung **Kein zulässiges Kennwort** quittiert.

Dem Datenbank-Kennwort wird nachgesagt, es sei einfach zu knacken. Das ist seit dem Wechsel von MDB nach ACCDB ab Access 2007 nicht mehr der Fall. Um das Kennwort zu ermitteln, bedarf es schon guter Crack-Tools mit der Möglichkeit von Brute-Force-Angriffen. Je länger und komplexer ein Kennwort, umso höher ist der Aufwand für einen solchen Angriff.

Das Datenbank-Kennwort bietet also durchaus einen Zugriffsschutz. Dieser ist abhängig von der Länge und der Komplexität des Kennworts.

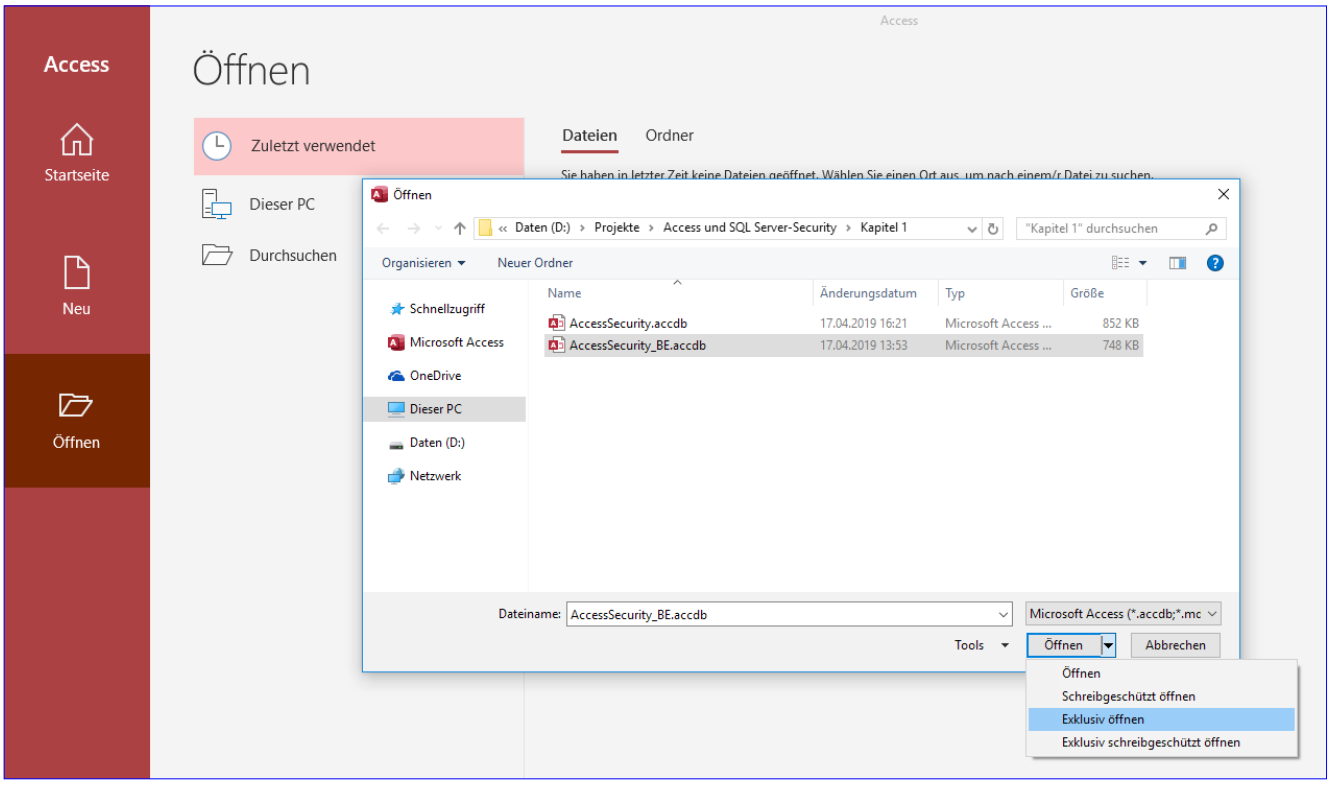


Bild 1: Exklusives Öffnen einer Datenbank

Doch genug der Theorie. Jetzt wird die Backend-Datenbank der Beispiel-Applikation **WaWi** verschlüsselt. Dazu öffnen Sie das Access-Backend **WaWi_BE.accdb** exklusiv (siehe Bild 1).

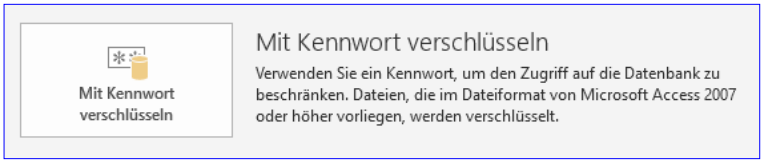


Bild 2: Verschlüsseln einer Datenbank mit einem Kennwort

Hier wechseln Sie zum Menüpunkt **Datei** und klicken auf **Informationen**. Dort sehen Sie die Schaltfläche **Mit Kennwort verschlüsseln** (siehe Bild 2).

Ein Klick auf diese Schaltfläche liefert Ihnen einen Dialog (siehe Bild 3), in dem Sie im ersten Eingabefeld Ihr Kennwort eingeben und dies mit einer erneuten Eingabe im zweiten Eingabefeld bestätigen.

Ihr Kennwort sollte den oben genannten Empfehlungen entsprechen. Um bei diesem und den folgenden Beispielen nicht immer mühsam ein komplexes Kennwort einzugeben, tragen Sie als Kennwort den Begriff **unsicher** ein. Danach erhalten Sie die Meldung aus Bild 4, die Sie darauf hinweist, dass eine Sperrung auf Datensatzebene mit der Verschlüsselung nicht kompatibel ist. Sollte eine Sperrung auf Datensatzebene für Sie unabdingbar sein, können Sie die Verschlüsselung nicht nutzen. Bestätigen Sie die Meldung, wird die Sperrung auf Datensatzebene in Zukunft ignoriert.

Bild 3: Festlegen des Kennworts

Soweit so gut – die Access-Datenbank ist nun verschlüsselt. Ab

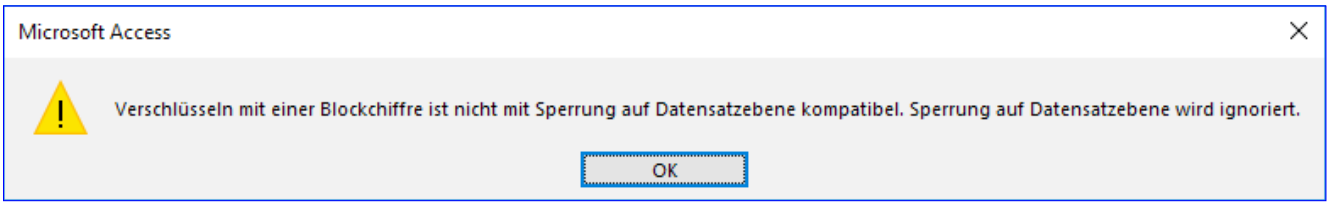


Bild 4: Hinweise zum Verschlüsseln

jetzt müssen Sie dieses Kennwort immer eingeben, wenn Sie die Datenbank öffnen. Testen Sie es einmal. Schließen Sie die Datenbank und öffnen Sie sie direkt wieder. Ohne Kennworteingabe ist der Zugriff auf die Daten nicht mehr möglich.

Dies gilt auch für das Access-Frontend, mit dem die Daten der nun verschlüsselten Datenbank verarbeitet werden. Die Daten stehen im Frontend nur nach Eingabe des Kennworts zur Verfügung.

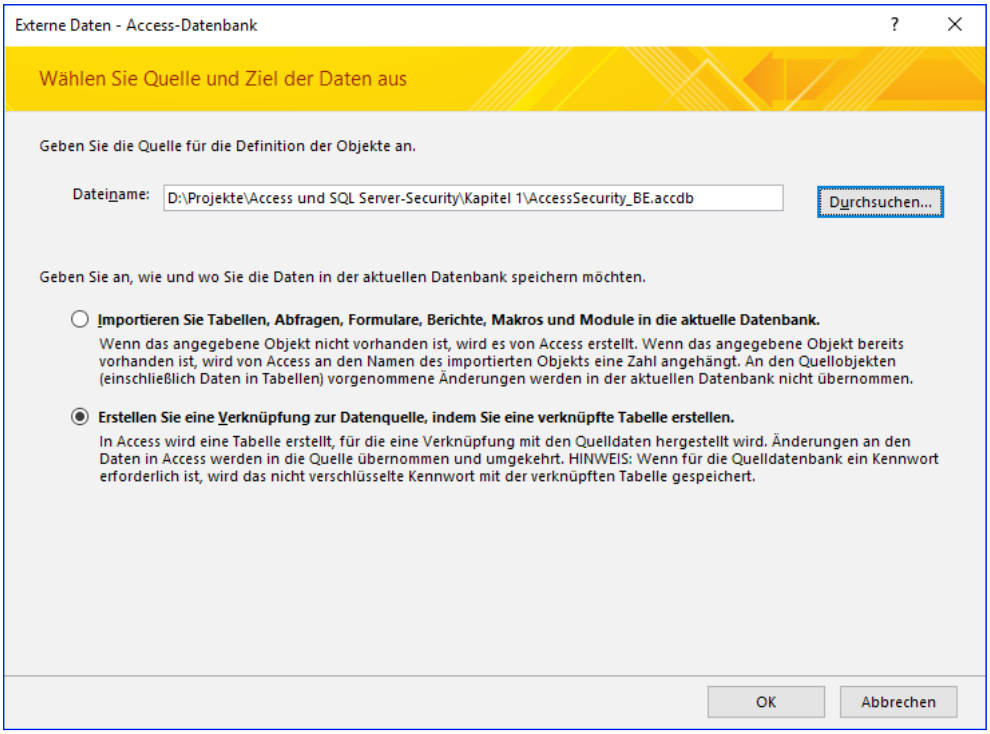


Bild 5: Erstellen einer Verknüpfung

Es gibt eine gute Nachricht. Sie müssen das Kennwort nicht jedes Mal eingeben. Der Einfachheit halber lässt sich das Kennwort beim Einbinden der Tabellen angeben und wird ab dann bei jedem Datenzugriff verwendet.

Leider ist der Tabellenverknüpfungs-Manager dabei keine Hilfe. Sie müssen die eingebundenen Tabellen aus dem Frontend entfernen und neu einbinden.

Öffnen Sie also das Frontend der Beispiel-Applikation **WaWi** und entfernen Sie die eingebundenen Tabellen. Anschließend navigieren Sie über **Externe Daten|Neue Datenquelle|Aus Datenbank|Access** zum Dialog **Externe Daten**. Dort wählen Sie über die Schaltfläche **Durchsuchen** die verschlüsselte Access-Datenbank aus und markieren die Option **Erstellen Sie eine Verknüpfung zur Datenquelle, indem Sie eine verknüpfte Tabelle erstellen** (siehe Bild 5).

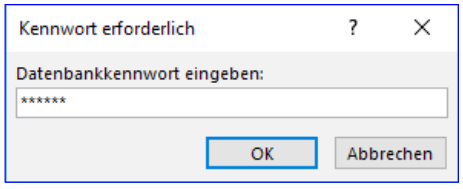


Bild 6: Abfrage des Kennworts

Nach einem Klick auf **OK** werden Sie aufgefordert, das Datenbank-Kennwort einzugeben (siehe Bild 6). Anschließend wählen Sie alle Tabellen aus und binden diese per **OK** ein.

Die Eingabe des Kennworts ist eine einmalige Angelegenheit. Sie können dies einfach testen. Schließen Sie das Access-Frontend und öffnen Sie es direkt wieder. Ein Kennwort müssen Sie dabei nicht eingeben, auch nicht beim Öffnen einer Tabelle.

Bild 7: Kennwort ermitteln per Direktbereich

Allerdings ist das Kennwort für den Datenzugriff erforderlich. Es muss also in irgendeiner Art und Weise hinterlegt sein. Doch wo ist das Kennwort gespeichert?

Um das herauszufinden, öffnen Sie mit der Tastenkombination Strg + G den Direktbereich im VBA-Editor. Hier führen Sie folgende Anweisung aus:

```
? CurrentDb.TableDefs("Ansprechpartner").Connect
```

Der Direktbereich zeigt Ihnen die **Connect**-Eigenschaft der eingebundenen Tabelle. Schon an zweiter Position ist das Kennwort zu sehen (siehe Bild 7). Es ist also kein Hexenwerk, das Datenbank-Kennwort einer verschlüsselten Datenbank zu ermitteln.

Das Auslesen des Kennworts lässt sich vermeiden, indem Sie auch das Access-Frontend verschlüsseln. Also öffnen Sie das Access-Frontend exklusiv, wechseln über das Menü **Datei** zu **Informationen** und klicken dort auf die Schaltfläche **Mit Kennwort verschlüsseln**.

Als Kennwort geben Sie natürlich nicht das gleiche wie beim Backend an. Verwenden Sie ein anderes sicheres Kennwort, das beim Frontend auch mehr als 19 Zeichen haben darf. Für dieses Beispiel kann es gerne wieder etwas Einfaches sein. Wie wäre es mit dem Kennwort **sicher**?

Es folgt der bereits bekannte Hinweis, dass die Datensatzsperre in Zukunft ignoriert wird. Mit Bestätigen dieser Meldung ist nun auch das Frontend verschlüsselt.

Dies können Sie testen, indem Sie das Frontend schließen und es direkt wieder öffnen. Als erstes wird von Ihnen die Eingabe des Kennworts verlangt.

Erst nach der Kennworteingabe lässt das Frontend den Zugriff auf das Startformular, die Access-Objekte und die eingebundenen Tabellen zu. Dieses Kennwort müssen Sie ab jetzt bei jedem Start des Frontends angeben. Aber nicht nur Sie, sondern jeder Anwender, der mit dieser Access-Applikation arbeitet.

Aus diesem Grund ist es wichtig, Frontend und Backend der Access-Applikation mit unterschiedlichen Kennwörtern zu verschlüsseln. Andernfalls wäre es für den Anwender recht einfach, das Access-Backend zu öffnen. Er muss dort nur das ihm bekannte Kennwort vom Frontend eingeben.

Ein sicheres Kennwort

Das von Ihnen gewählte Datenbank-Kennwort sollte den oben angeführten Regeln entsprechen. Allerdings muss es sich der Anwender auch merken können. Genau hier beginnt das Dilemma. Wer kann sich schon an ein komplexes Kennwort erinnern? Die Lösung ist nicht selten eine Notiz am Bildschirm oder unter der Tastatur.

Dabei ist die Vergabe eines komplexen Kennworts recht einfach. Denken Sie sich einen Satz mit einer Zahl aus und nehmen Sie davon die Anfangsbuchstaben und die Satzzeichen. Fertig ist das sichere Kennwort.

Der Anwender merkt sich einen Satz wie **Seit 2020 ist die Auftragsverarbeitung mit einem Kennwort geschützt!** und gibt als Kennwort **S2020idAmeKg!** ein. Natürlich verwenden Sie bitte nicht dieses Beispiel als Ihr Kennwort. Denken Sie sich einen Satz aus, mit

dem die Anwender etwas anfangen können.

Nimmt man es ganz genau, besitzt ein solch erstelltes Kennwort zwar einen hohen Sicherheitsgrad, es gilt dennoch als unsicher. Ein sicheres Kennwort ergibt sich aus der Zufälligkeit. Es sollte weder etwas mit der Applikation noch mit den Benutzern noch mit irgendetwas zu tun haben. Sichere Kennwörter erstellen Sie am besten mit einem Password-Generator.

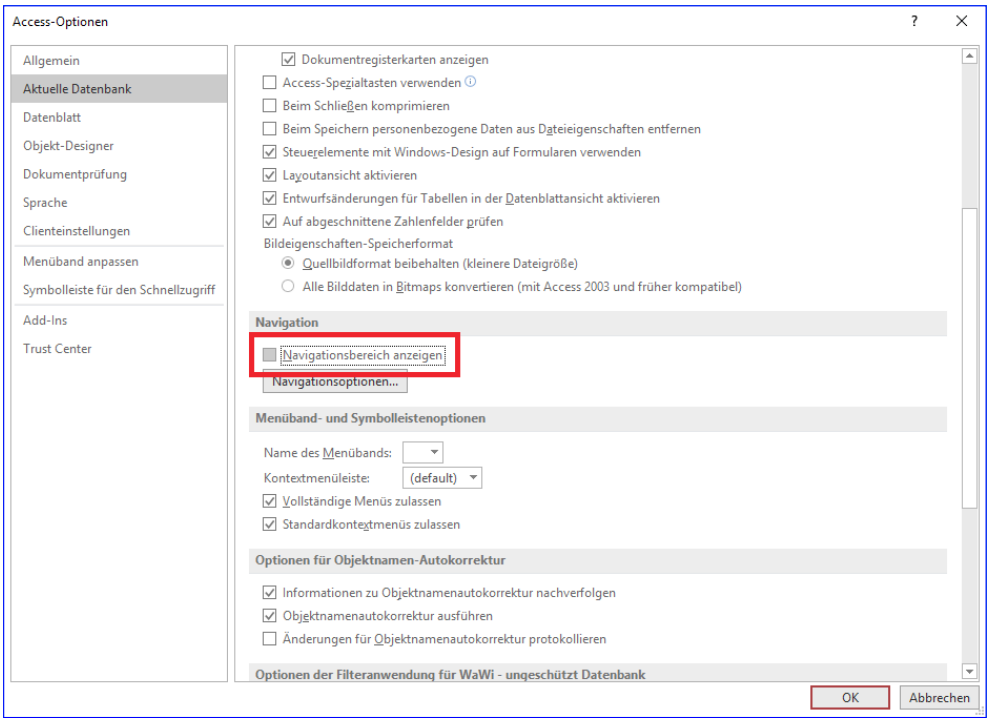


Bild 8: Navigationsbereich ausblenden

Ein generiertes Kennwort kann sich aber kein Mensch merken. Schon sind Sie wieder beim oben beschriebenen Dilemma und das Kennwort landet früher oder später als Notiz auf einem Zettel. So wird auch das sicherste Kennwort untauglich, ist es doch leicht zugänglich für jedermann.

Dann doch lieber ein Kennwort, das bereits einen hohen Sicherheitsgrad aufweist und das sich die Anwender merken können.

Auch sichere Kennwörter sollten Sie in regelmäßigen Abständen ändern. Kennt ein Unbefugter das Kennwort, darf er sich nach der Änderung erneut die Mühe machen, an das Kennwort heranzukommen.

Access-Funktionen unterbinden

Nach Eingabe des Datenbank-Kennworts stehen dem Anwender alle ihm erlaubten Access-Funktionen zur Verfügung. Er kann zum Beispiel den VBA-Editor öffnen und dort mittels der **Connect**-Eigenschaft einer Tabelle das Kennwort des Access-Backends auslesen.

Aus diesem Grund darf der Zugriff auf den VBA-Editor nicht ohne weiteres möglich sein. Um dies zu erreichen, ersetzen Sie die Standard-Ribbons mit eigenen Ribbons und blenden über die Access-Optionen den Navigationsbereich aus (siehe Bild 8). Diese beiden Aktionen verbergen allerdings nur die Möglichkeiten zum Öffnen des VBA-Editors. Mit der Tastenkombination **Alt + F11** lässt er sich weiterhin öffnen.

Kennwort für VBA

Als weitere Maßnahme schützen Sie das VBA-Projekt mit einem Kennwort. Dieses Kennwort müssen Sie dann jedes Mal eingeben, wenn Sie Ihren VBA-Quellcode lesen oder ändern möchten.

Dazu wechseln Sie zum VBA-Editor und wählen im Menü **Extras** den Eintrag **Eigenschaften von**. In dem folgenden Dialog gehen Sie zur Registerkarte **Schutz**, aktivieren die Option **Projekt für die Anzeige sperren** und geben ein neues sicheres Kennwort in die beiden Eingabefelder ein (siehe Bild 9).

GUIDs per VBA erstellen (64-Bit-kompatibel)

Access kann ab Version 2010 auch als 64-Bit-Variante installiert werden. Ab Version 2019 ist 64-Bit die Voreinstellung bei der Installation. Wir stellen in dieser Beitragsreihe die Änderungen vor, die für die Verwendung verschiedener oft genutzter VBA-Prozeduren und Komponenten unter der 64-Bit-Version nötig sind. Dabei wollen wir auch dafür sorgen, dass diese unter der 32-Bit-Version von Access älter als 2010 ebenfalls noch laufen. Den Anfang machen wir mit der 64-Bit-Version der in vielen Beispiellösungen verwendeten Funktion zum Ermitteln von GUIDs.

Wenn Sie die alte Version unter Access in der 64-Bit-Version kompilieren wollen, stoßen Sie schnell auf einige der Inkompatibilitäten, und zwar noch vor dem Kompilieren. Die beiden API-Funktionen **CoCreateGuid** und **StringFromGUID2** werden rot angezeigt (siehe Bild 1).

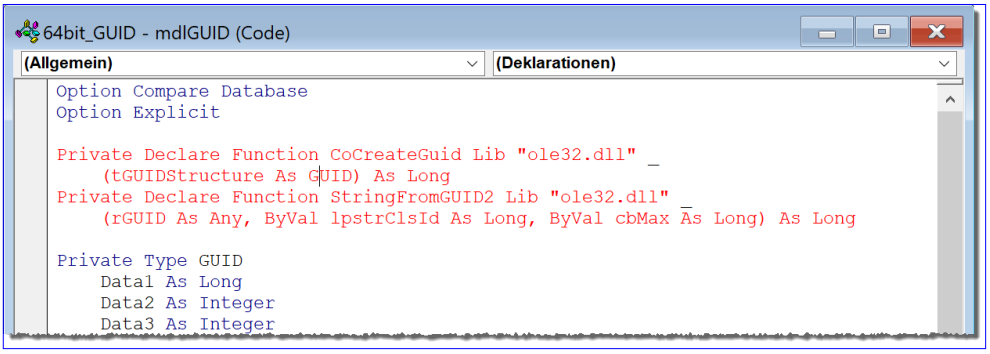


Bild 1: Fehlerhaft markierte API-Deklarationen

Damit sich das ändert, fügen Sie zwischen die beiden Schlüsselwörter **Declare** und **Function** das Schlüsselwort **PtrSafe** ein:

```
Private Declare PtrSafe Function CoCreateGuid ⤸
        Lib "ole32.dll" (tGUIDStructure As GUID) As Long
Private Declare Function StringFromGUID2 Lib "ole32.dll" ⤸
        (rGUID As Any, ByVal lpstrClsId As Long, ⤸
        ByVal cbMax As Long) As Long
```

Damit entfällt bereits die rote Markierung dieser Funktionsdeklarationen und Sie können den Befehl **Debuggen|Kompilieren von <Projektname>** aufrufen.

Dieser liefert einen Kompilierfehler für den Aufruf der Funktion **StringFromGUID2**, wo für den zweiten Parameter eine Typunverträglichkeit angezeigt wird. Dies ändern wir, indem wir den Typ des zweiten Parameters in der Deklaration der API-Funktion **StringFromGUID2** von **Long** in **LongPtr** ändern:

```
Private Declare Function StringFromGUID2 Lib "ole32.dll" ⤸
        (rGUID As Any, ByVal lpstrClsId As LongPtr, ⤸
        ByVal cbMax As Long) As Long
```

Danach lässt sich die Funktion kompilieren und wir können diese mit dem folgenden Befehl im Direktbereich des VBA-Editors aufrufen:

```
Debug.Print CreateGUID
{058FB0DE-A528-41B9-8AE0-F19DA8FD27F6}
```

Probleme gibt es nun nur noch, wenn Sie diese Funktion in Access-Versionen verwenden, die noch kein VBA 7 enthalten, sondern VBA 6. Dann müssen Sie noch eine Unterscheidung einfügen, die prüft, welche VBA-Version vorliegt und die entsprechende Deklaration der API-Funktionen einsetzt.

Dabei verwenden wir die sogenannten Compiler-Konstanten. Es gibt beispielsweise die Konstanten **VBA6** und **VBA7**. **VBA6** liefert den Wert **True**, wenn VBA 6 verwendet wird. **VBA7** liefert den Wert **True**, wenn VBA 7 zum Einsatz kommt.

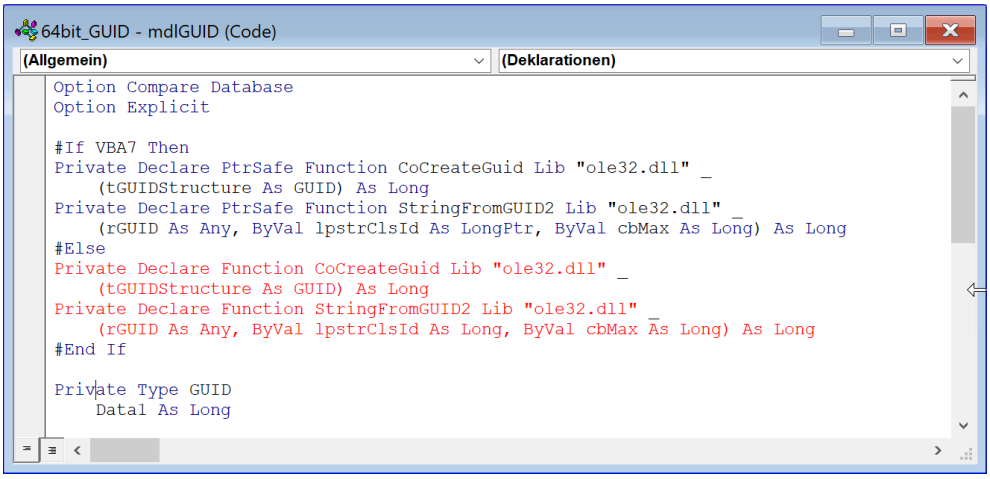


Bild 2: API-Deklaration für 32- und 64-Bit

Nur VBA 7 unterstützt die 32-Bit-Version der API-Befehle und die 64-Bit-Version der API-Befehle parallel. Daher prüfen wir in einer **If... Then**-Bedingung, ob **VBA7** den Wert **True** liefert. Falls ja, deklarieren wir die 64-Bit-Version der API-Funktionen, andernfalls die 32-Bit-Version (siehe Bild 2).

Aber Deklarationen in einer **If...Then**-Bedingung? Wir verwenden hier keine normale **If...Then**-Bedingung, sondern eine, die festlegt, welche Teile des Codes überhaupt kompiliert werden. Die einzelnen Anweisungen der **If... Then**-Bedingung werden daher mit den Schlüsselwörtern **#If ... Then**, **#Else** und **#End If** ausgeführt. Das sieht schließlich wie folgt aus:

```
#If <Bedingung> Then
    '... Anweisungen
#Else
    '...Anweisungen
#End If
```

In unserem Fall verwenden wir als Bedingung im **#If**-Teil schlicht **VBA7** (was **VBA7 = True** entspricht und tragen im **#If**-Zweig die 64-Bit-Version der Deklarationen ein. Im **#Else**-Zweig legen wir die 32-Bit-Version der Deklaration ab (siehe Listing 1).

Die **VBA6**-Version wird dann immer noch rot markiert, aber wenn Sie das Projekt kompilieren, werden keine Fehler angezeigt, solange der jeweilige Zweig der **#If... Then**-Bedingung keine falschen API-Deklarationen enthält.

```
#If VBA7 Then
Private Declare PtrSafe Function CoCreateGuid Lib "ole32.dll" _
    (tGUIDStructure As GUID) As Long
Private Declare PtrSafe Function StringFromGUID2 Lib "ole32.dll" _
    (rGUID As Any, ByVal lpstrClsId As LongPtr, ByVal cbMax As Long) As Long
#Else
Private Declare Function CoCreateGuid Lib "ole32.dll" _
    (tGUIDStructure As GUID) As Long
Private Declare Function StringFromGUID2 Lib "ole32.dll" _
    (rGUID As Any, ByVal lpstrClsId As Long, ByVal cbMax As Long) As Long
#End If
```

Listing 1: Unterscheidung zwischen VBA 7 und VBA 6

Zusammenfassung und Ausblick

Dieser Beitrag zeigt, wie Sie die **CreateGUID**-Funktion samt API-Funktionen 64-Bit-fähig machen. In weiteren Beiträgen zeigen wir die Änderungen für andere oft verwendete VBA-Elemente.

Wechselkurse per Webservice

Für viele finanzwirtschaftliche Anwendungen spielt die Umwandlung von Beträgen in verschiedene Währungen eine Rolle. Da wäre es doch praktisch, wenn man immer die aktuellen Umrechnungskurse parat hätte. Am besten jedoch nicht so, dass man diese immer aus dem Internet oder der Tageszeitung entnimmt und in die Anwendung eintippt – sondern eher als Information, die eine Anwendung automatisch aus dem Internet bezieht. Wir schauen uns an, wie wir den Webservice der europäischen Zentralbank für unsere Zwecke nutzen können.

Fertige XML-Dateien der europäischen Zentralbank

Für den schnellen Gebrauch bietet die europäische Zentralbank XML-Dokumente an. Die täglichen Wechselkurse von EUR in viele andere Währung bietet beispielsweise diese Webseite an:

`http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml`

Das Ergebnis sieht im Browser wie in Bild 1 aus. Nun müssen wir es nur noch schaffen, den Inhalt dieser Datei per VBA abzurufen und auf die Daten im XML-Format zuzugreifen.

Bild 1: XML-Dokument mit Wechselkursen

XML-Verweis

Dazu fügen wir dem VBA-Projekt unserer Beispieldatenbank erst einmal einen Verweis auf die Bibliothek **Microsoft XML 6.0 Object Library** hinzu (siehe Bild 2).

Außerdem benötigen wir noch einen Verweis auf die Bibliothek **Microsoft WinHTTP Services, version 5.1**.

Damit können wir nun eine erste kleine Testprozedur bauen, die uns den Inhalt der obigen URL herunterlädt und diesen

in ein XML-Dokument überträgt, bevor wir dessen Inhalt im Direktfenster ausgeben lassen.

Diese Prozedur finden Sie in Listing 1. Sie deklariert zwei Objekte – eines mit dem Typ **WinHttp.WinHttpRequest** und eines des Typs **MSXML2.DOMDocument60**. Das erste dient dazu, die Anfrage an die URL zu schicken und die zweite zum Verarbeiten des zurückgelieferten XML-Dokuments.

Die Prozedur speichert die URL in der Variablen **strURL**. Dann erstellt sie ein neues Objekt des Typs **WinHttp.WinHttpRequest** und referenziert dieses mit der Variablen **objWinHTTP**. Die **Open**-Anweisung erwartet die Art des Zugriffs, die zu öffnende URL und einen **Boolean**-Wert, der angibt, ob die Anfrage synchron oder asynchron ausgeführt werden soll.

Wir benötigen die synchrone Variante, damit der Code solange angehalten wird, bis der Inhalt abgerufen wurde. Die **send**-Methode schließlich schickt die Abfrage ab.

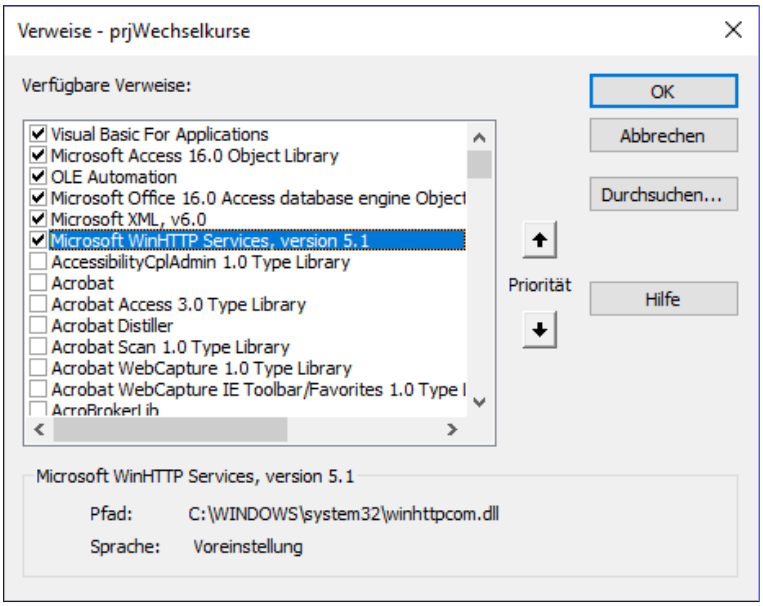


Bild 2: Verweis auf die XML-Bibliothek

```
Public Sub WechselkurseInsDirektfenster()
    Dim objWinHTTP As WinHttp.WinHttpRequest
    Dim objXML As MSXML2.DOMDocument60
    Dim strURL As String
    strURL = "http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml"
    Set objWinHTTP = New WinHttp.WinHttpRequest
    objWinHTTP.Open "GET", strURL, False
    objWinHTTP.send
    If objWinHTTP.status = 200 Then
        Set objXML = New MSXML2.DOMDocument60
        If objXML.loadXML(objWinHTTP.responseText) Then
            Debug.Print objXML.XML
        End If
    End If
End Sub
```

Listing 1: Prozedur zum Herunterladen eines XML-Dokuments

Danach prüfen wir in einer **If...Then**-Bedingung, ob die Eigenschaft **Status** den Wert **200** enthält, was bedeutet, dass die Abfrage erfolgreich war. In diesem Fall erstellt die Prozedur ein neues Objekt des Typs **MSXML2.DOMDocument60**.

Mit der Methode **LoadXML** füllen wir das XML-Dokument mit der durch den HTTP-Request gelieferten Antwort, ermittelt mit der Eigenschaft **responseText**. Wenn der Aufruf den Wert **True** zurückgibt, wurde der Text erfolgreich als XML-Dokument eingelesen und wir geben diesen mit der XML-Eigenschaft des **MSXML2.DOMDocument**-Objekts im Direktbereich des VBA-Editors aus.

Das Ergebnis sieht wie in Bild 3 aus.

Tabellen zum Speichern der Wechselkurse

Nun können wir uns an die Aufgabe begeben, die gewünschten Daten aus dem XML-Dokument zu extrahieren.

Wir wollen das Datum ermitteln und die Wechselkurse von Euro in die verschiedenen anderen Währungen in eine Tabelle eintragen. Die Tabelle sieht in der Entwurfsansicht wie in Bild 4 aus.

Neben dem Primärschlüsselfeld **WechselkursID** verwenden wir ein Fremdschlüsselfeld namens **WaehrungID**, mit dem wir einen Eintrag einer weiteren Tabelle namens **tblWaehrungen** auswählen können. Diese Tabelle enthält nur die beiden Felder **WaehrungID** und **Waehrung**, wobei wir das letztere mit einem eindeutigen Index ausgestattet haben, damit jede Währung nur einmal in die Tabelle eingetragen werden kann.

Das Feld **Wechselkursdatum** wollen wir mit dem Datum füllen, für das der Wechselkurs abgefragt wurde. Schließlich fehlt noch das Feld **Wechselkurs**, das den eigentlichen Wechselkurs aufnehmen soll. Dieses haben wir als Währungsfeld mit dem Format **Allgemeine Zahl** definiert.

Für die beiden Felder **WaehrungID** und **Wechselkursdatum** haben wir außerdem noch einen zusammengesetzten, eindeutigen Index definiert, damit für jede Kombination aus Währung und Datum nur ein Wechselkurs eingetragen werden kann.

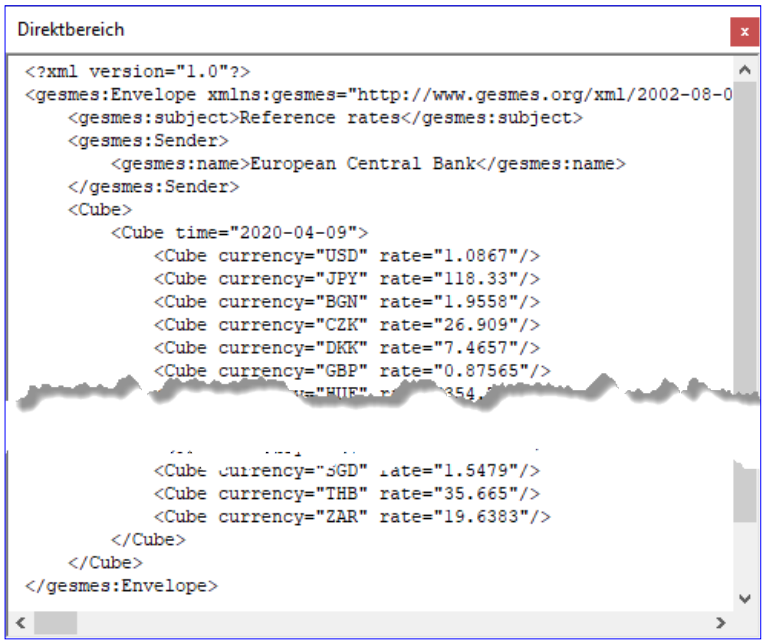


Bild 3: Ausgabe des soeben noch im Webbrowser betrachteten XML-Dokuments im Direktbereich des VBA-Editors

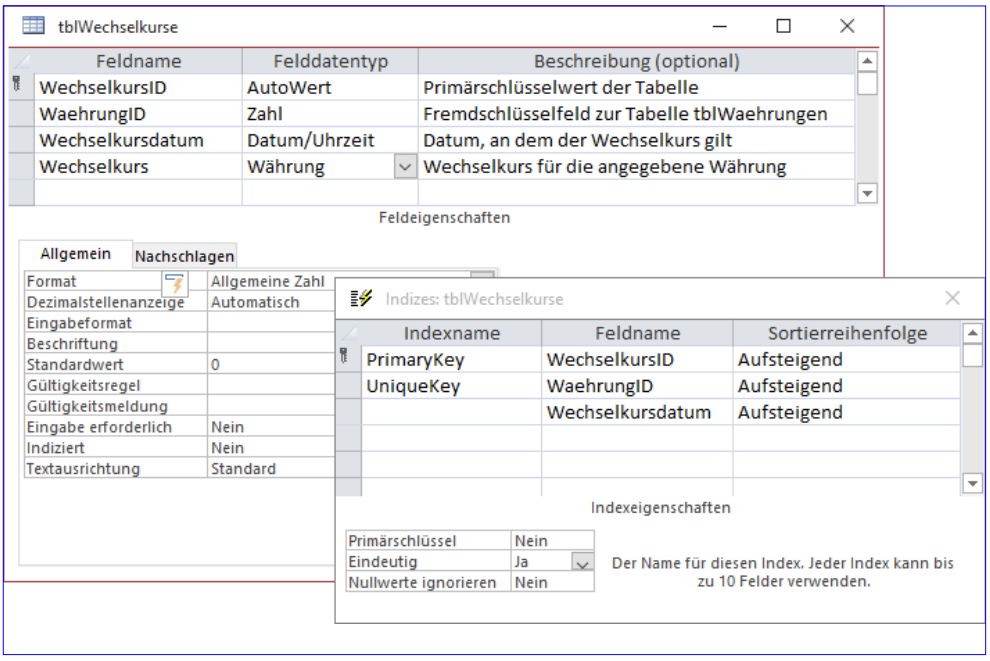


Bild 4: Entwurf der Tabelle **tblWechselkurse**

XML-Dokument auslesen

Nun wollen wir das ermittelte XML-Dokument auslesen und die ermittelten Informationen in die beiden Tabellen **tblWaehrungen** und **tblWechselkurse** speichern. Die erste Prozedur, die wir hierzu einsetzen, heißt **WechselkurseSpeichern** (siehe Listing 2). Diese ist prinzipiell wie die weiter oben vorgestellte Prozedur **WechselkurseIns-Direktfenster** aufgebaut, ruft aber nach dem Einlesen

des XML-Dokuments die Prozedur **Wechselkurse-Auslesen** auf und übergibt dieser als Parameter das **MSXML2.DOMDocument60**-Element.

```
Public Sub WechselkurseSpeichern()
    Dim objWinHTTP As WinHttp.WinHttpRequest
    Dim objXML As MSXML2.DOMDocument60
    Dim strURL As String
    strURL = "http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml"
    Set objWinHTTP = New WinHttp.WinHttpRequest
    objWinHTTP.Open "GET", strURL, False
    objWinHTTP.send
    If objWinHTTP.status = 200 Then
        Set objXML = New MSXML2.DOMDocument60
        If objXML.loadXML(objWinHTTP.responseText) Then
            WechselkurseAuslesen objXML
        End If
    End If
End Sub
```

Listing 2: Prozedur zum Speichern der Wechselkurse

Prozedur zum Auslesen der Wechselkurse

Diese Prozedur nimmt den Parameter namens **objXML** mit dem Typ **MSXML2.DOMDocument60** entgegen und deklariert zunächst einige Variablen (siehe Listing 3).

In **objDatum** wollen wir das XML-Element speichern, welches das Datum der Wechselkurse enthält. In **objWechselkurs** wollen wir die XML-Elemente speichern, das die Währung und den Wechselkurs liefern. **datDatum**, **strWaehrung** und **curWechselkurs** nehmen die dabei ermittelten Informationen auf. **strXMLNamespace** benötigen wir für eine Spezialaufgabe, die wir gleich beschreiben.

Für das XML-Dokument sind nämlich verschiedene Namespaces festgelegt – dies können wir dem Kopf des Dokuments entnehmen, das wie folgt aussieht:

```
Public Sub WaehrungenAuslesen(objXML As MSXML2.DOMDocument60)
    Dim objDatum As MSXML2.IXMLDOMNode
    Dim datDatum As Date
    Dim objWechselkurs As MSXML2.IXMLDOMNode
    Dim strWaehrung As String
    Dim curWechselkurs As Currency
    Dim strXMLNamespaces As String
    strXMLNamespaces = "xmlns:d=""http://www.ecb.int/vocabulary/2002-08-01/eurofxref"""
    objXML.SetProperty "SelectionNamespaces", strXMLNamespaces

    Set objDatum = objXML.selectSingleNode("//d:Cube[@time]")
    datDatum = objDatum.Attributes.getNamedItem("time").nodeTypedValue
    For Each objWechselkurs In objXML.selectNodes("//d:Cube[@currency]")
        strWaehrung = objWechselkurs.Attributes.getNamedItem("currency").nodeTypedValue
        curWechselkurs = Replace(objWechselkurs.Attributes.getNamedItem("rate").nodeTypedValue, ".", ",")
        WechselkursSpeichern datDatum, strWaehrung, curWechselkurs
    Next objWechselkurs
End Sub
```

Listing 3: Prozedur zum Auslesen der Daten aus der XML-Datei

```
<gesmes:Envelope xmlns:gesmes="http://www.gesmes.org/
xml/2002-08-01" xmlns="http://www.ecb.int/vocabula-
ry/2002-08-01/eurofxref">
```

Wir haben also den Namespace **http://www.gesmes.org/xml/2002-08-01**, dessen Element mit dem Präfix **gesmes** markiert werden. Das gilt beispielsweise gleich für das erste Element, das die **Namespace**-Angaben enthält.

Außerdem haben wir den Namespace **http://www.ecb.int/vocabulary/2002-08-01/eurofxref**, der ohne Präfix kommt. Dementsprechend bezieht er sich auf alle Elemente, die ohne Präfix angegeben werden. Dabei handelt es sich um die **Cube**-Elemente – und diese enthalten genau die Informationen, die wir einlesen und verarbeiten wollen:

```
<Cube>
    <Cube time="2020-04-09">
        <Cube currency="USD" rate="1.0867"/>
        ...
    </Cube>
</Cube>
```

Damit gleich beim Aufruf der Methoden **selectNodes** und **selectSingleNodes** klar ist, in welchem Namespace sich die Elemente beziehen, die wir suchen wollen, legen wir diesen zuvor mit einer Property namens **SelectionNamespaces** fest:

```
strXMLNamespaces = "xmlns:d=""http://www.ecb.int/vocabula-
ry/2002-08-01/eurofxref"""
objXML.SetProperty "SelectionNamespaces", strXMLNamespaces
```

Damit können wir nun das Element einlesen, welches uns das Datum liefert und so aussieht:

```
<Cube time="2020-04-09">
```

In der Hierarchie des XML-Dokuments befindet es sich unterhalb eines weiteren **Cube**-Elements.

Wir könnten mit folgendem Suchausdruck darauf zugreifen:

```
Set objDatum = objXML.selectSingleNode("//d:Cube/d:Cube")
```

Dies liefert das erste **Cube**-Element, das ein übergeordnetes **Cube**-Element enthält. Wir verwenden aber den folgenden Ausdruck:

```
Set objDatum = objXML.selectSingleNode("//d:Cube[@time]")
```

Damit suchen wir etwas gezielter nach dem ersten Cube-Element, welches das **time**-Attribut enthält. Dieses lesen wir mit der nächsten Anweisung in die Varible **datDatum** ein:

```
datDatum = objDatum.Attributes.getNamedItem("time").node-
TypedValue
```

Anschließend durchlaufen wir in einer **For Each**-Schleife alle Elemente, die dem Schema **//d:Cube[@currency]** entsprechen und somit ein **Cube**-Element enthalten, welches das Attribut **currency** enthält:

```
For Each objWechselkurs In 
                objXML.selectNodes("//d:Cube[@currency]")
```

Innerhalb dieser Schleife ermitteln wir zunächst den Wert des Attributs **currency** und schreiben diesen in die Variable **strWaehrung**:

```
strWaehrung = objWechselkurs.Attributes.
                    getNamedItem("currency").nodeTypedValue
```

Danach lesen wir den Wechselkurs aus dem Attribut **rate** ein und ersetzen hier direkt den Punkt durch das Komma als Dezimaltrennzeichen:

```
curWechselkurs = Replace(objWechselkurs.Attributes.
            getNamedItem("rate").nodeTypedValue, ".", ",")
```

Schließlich rufen wir die nächste Prozedur namens **WechselkursSpeichern** auf, der wir das Datum, die Währung

und den Wechselkurs mit den Variablen **datDatum**, **strWaehrung** und **curWechselkurs** übergeben:

```
WechselkursSpeichern datDatum, strWaehrung, curWechselkurs
```

Diese Anweisungen erledigen wir für alle **Cube**-Elemente mit dem Attribut **rate**, welche die **SelectNodes**-Methode liefert.

Daten in den Tabellen speichern

Das Speichern der Daten in den Tabellen erledigt die Prozedur **WechselkursSpeichern**, die für jede Kombination aus Datum, Währung und Wechselkurs einmal von der Prozedur **WechselkurseAuslesen** aus aufgerufen wird (siehe Listing 4).

Diese Prozedur deklariert ein **Database**-Objekt namens **db**, mit dessen **Execute**-Methode gleich die Währung aus **strWaehrung** in die Tabelle **tblWaehrungen** eingetragen wird:

```
db.Execute "INSERT INTO tblWaehrungen(Waehrung) 7
            VALUES('" & strWaehrung & "')", dbFailOnError
```

Zuvor deaktivieren wir die Fehlerbehandlung, damit im Falle eines Fehlers nicht die eingebaute Fehlermeldung

```
Public Sub WechselkursSpeichern(datDatum As Date, strWaehrung As String, curWechselkurs As Currency)
    Dim db As DAO.Database
    Dim lngWaehrungID As Long
    Dim strSQL As String
    Set db = CurrentDb
    On Error Resume Next
    strSQL = "INSERT INTO tblWaehrungen(Waehrung) VALUES('" & strWaehrung & "')"
    db.Execute strSQL, dbFailOnError
    Select Case Err.Number
        Case 3022
            lngWaehrungID = DLookup("WaehrungID", "tblWaehrungen", "Waehrung = '" & strWaehrung & "'")
        Case 0
            lngWaehrungID = db.OpenRecordset("SELECT @@IDENTITY", dbOpenDynaset).Fields(0)
        Case Else
            MsgBox "Fehler " & Err.Number & "'" & Err.Description & "'"
    End Select
    On Error GoTo 0
    On Error Resume Next
    strSQL = "INSERT INTO tblWechselkurse(WaehrungID, Wechselkursdatum, Wechselkurs) VALUES(" & lngWaehrungID & ", " _
        & SQLDatum(datDatum) & ", " & Replace(curWechselkurs, ",", ".") & ")"
    db.Execute strSQL, dbFailOnError
    Select Case Err.Number
        Case 3022
            Debug.Print "Wechselkurs für EUR/" & strWaehrung; " am " & datDatum & " bereits vorhanden."
        Case 0
        Case Else
            MsgBox "Fehler " & Err.Number & "'" & Err.Description & "'"
    End Select
End Sub
```

Listing 4: Prozedur zum Speichern der Wechselkurse mit Parametern

erscheint. Stattdessen werten wir die Fehlernummer aus und prüfen, ob diese den Wert **3022** enthält. Das würde darauf hindeuten, dass bereits ein Datensatz mit der zu speichernden Währungsbezeichnung in der Tabelle **tblWaehrung** vorhanden ist.

Der Fehler wird ausgelöst, weil wir für das Feld **Waehrung** einen eindeutigen Index definiert haben.

Egal, ob wir den Datensatz in der Tabelle **tblWaehrungen** neu anlegen oder nicht, wir benötigen den Wert des Primärschlüsselfeldes der Tabelle **tblWaehrungen** für den Datensatz mit der übergebenen Währung. Falls die Währung bereits vorhanden war, ermitteln wir den Primärschlüsselwert über eine **DLookup**-Funktion, der wir den Namen der Währung als Vergleichswert für das Feld **Waehrung** übergeben und tragen den zurückge-

tblWaehrungen

WaehrungID	Waehrung	Zum H
1	USD	
2	JPY	
3	BGN	
4	CZK	
5	DKK	
6	GBP	
7	HUF	
8	PLN	
9	RON	
10	SEK	
11	CHF	
12	ISK	
13	NOK	
14	HRK	
15	RUB	
16	TRY	
17	AUD	
18	BRL	
19	CAD	
20	CNY	
21	HKD	
22	IDR	
23	ILS	
24	INR	
25	KRW	
26	MXN	
27	MYR	
28	NZD	
29	PHP	
30	SGD	
31	THB	
32	ZAR	
	(Neu)	

Datensatz: 1 von 32 Kein Filt

Bild 5: Daten in der Tabelle **tblWaehrungen**

tblWechselkurse

WechselkursID	WaehrungID	Wechselkursdatum	Wechselkurs
1	ZAR	09.04.2020	1,0867
2	USD	09.04.2020	118,33
3	JPY	09.04.2020	1,9558
4	BGN	09.04.2020	26,909
5	CZK	09.04.2020	7,4657
6	DKK	09.04.2020	0,8756
7	GBP	09.04.2020	354,76
8	HUF	09.04.2020	4,5586
9	PLN	09.04.2020	4,833
10	RON	09.04.2020	10,9455
11	SEK	09.04.2020	1,0558
12	CHF	09.04.2020	155,9
13	ISK	09.04.2020	11,2143
14	NOK	09.04.2020	7,6175
15	HRK	09.04.2020	80,69
16	RUB	09.04.2020	7,3233
17	TRY	09.04.2020	1,7444
18	AUD	09.04.2020	5,5956
19	BRL	09.04.2020	1,5265
20	CAD	09.04.2020	7,6709
21	CNY	09.04.2020	8,4259
22	HKD	09.04.2020	17243,21
23	IDR	09.04.2020	3,8919
24	ILS	09.04.2020	82,9275
25	INR	09.04.2020	1322,49
26	KRW	09.04.2020	26,0321
27	MXN	09.04.2020	4,7136
28	MYR	09.04.2020	1,8128
29	NZD	09.04.2020	54,939
30	PHP	09.04.2020	1,5479
31	SGD	09.04.2020	35,665
32	THB	09.04.2020	19,6383
(Neu)			0

Datensatz: 1 von 32 Kein Filter Suchen

Bild 6: Daten in der Tabelle **tblWechselkurse**

Verweise aus anderer Datenbank importieren

Wenn eine Datenbank beschädigt ist, können Sie diese oft retten, indem Sie eine neue Datenbank aufsetzen und alle Objekte der alten Datenbank in die neue Datenbank importieren. Leider berücksichtigt die Import-Funktion nicht die Verweise des VBA-Projekts. Je nach der Anzahl der Verweise kann das Übertragen der Verweise recht mühsam werden. Zum Glück kann man fast alles automatisieren, was sich sonst über die Benutzeroberfläche von Access erledigen lässt. In diesem Fall bauen wir uns ein Add-In, mit dem wir der aktuell geöffneten Datenbank die Verweise einer ausgewählten Datenbank hinzufügen können.

Gründe dafür, Verweise aus anderen Datenbanken zu importieren, gibt es genug. Einen haben wir bereits genannt – wenn Sie eine Datenbank bestehend auf einer anderen, gegebenenfalls beschädigten Datenbank neu aufbauen wollen, dann sind die Verweise eines der wenigen Elemente, die Sie nicht mit der eingebauten Importieren-Funktion übertragen können.

Ein anderer Grund ist die Unzulänglichkeit des **Verweise**-Dialogs: Wenn dieser für das Quellprojekt beispielsweise Einträge enthält, die nicht in der Liste der Verweise enthalten sind, sondern die über den **Durchsuchen**-Dialog hinzugefügt wurden, dann müssen Sie diese auch in der Zieldatenbank wieder über den **Durchsuchen**-Dialog hinzufügen.

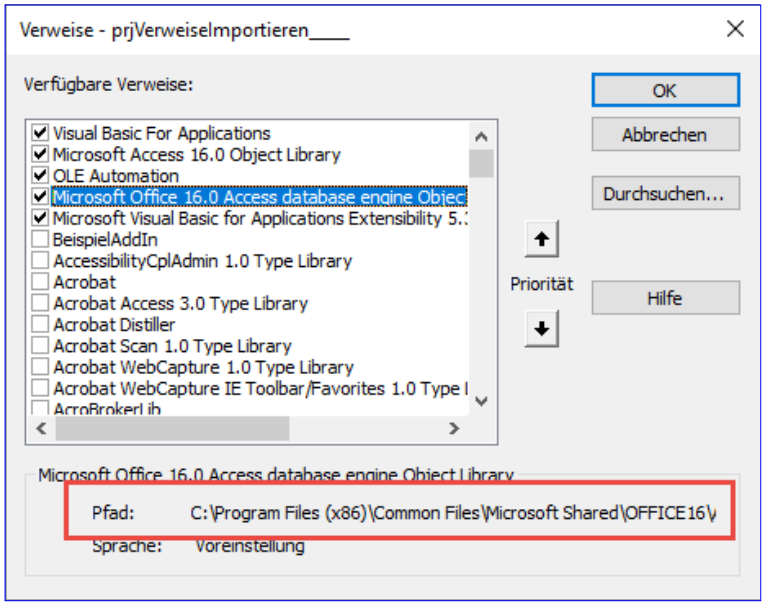


Bild 1: Zu kleines Textfeld für den Pfad zur Verweisdatei

Wenn der Pfad der Datei hinter dem Verweis wie in Bild 1 etwas länger ist, dann können Sie den hinteren Teil nicht mehr lesen, weil nur ein geringer Teil des Pfades angezeigt wird.

Add-In zum Importieren von Verweisen

Deshalb haben wir ein Add-In entwickelt, das alle gängigen Verweise aus einem VBA-Projekt in ein anderes übertragen kann. Dieses sieht in Aktion wie in Bild 2 aus. Das Add-In zeigt zunächst nur die Verweise des VBA-Projekts der Datenbank an, für die das Add-In geöffnet wurde und in welche die Verweise aus einem anderen VBA-Projekt importiert werden sollen.

Diese Verweise finden Sie in der rechten Liste unter **Verweise der aktuellen Datenbank**. Wenn Sie mit dem **Datei auswählen**-Dialog die Datenbankdatei ausgewählt haben, aus der die Verweise eingelesen werden sollen, zeigt das linke Listenfeld unter **Verweise der gewählten Datenbank** alle Verweise der Quelldatenbank an.

Nun haben Sie verschiedene Möglichkeiten, die Verweise aus der linken Liste in die rechte Liste zu übertragen –

und diese gleichzeitig zur aktuell geöffneten Datenbank hinzuzufügen:

- Doppelklick auf den jeweiligen Listeneintrag
- Markieren des zu übertragenden Verweises und Betätigen der Schaltfläche mit dem Pfeil
- Klicken auf die Schaltfläche mit dem Doppelpfeil, um alle Verweise zu übertragen, die in der Quelldatenbank, aber nicht in der Zieldatenbank enthalten sind.

Und Sie können auch noch Verweise aus der Zieldatenbank entfernen, indem Sie doppelt auf diese klicken.

Entwerfen des Formulars

Das Formular **frmVerweiseImportieren** sieht in der Entwurfsansicht wie in Bild 3 aus. Das Textfeld zur Eingabe des Pfades der Quelldatenbank heißt **txtImportVon**, die Schaltfläche zum Öffnen des **Datei auswählen**-Dialogs heißt **cmdDateiauswahl**. Die beiden Listenfelder nennen wir kurz **lstQuelle** und **lstZiel**. Die beiden Schaltflächen zum Übertragen eines oder aller Verweise heißen **cmdHinzufuegen** und **cmdAlleHinzufuegen**.

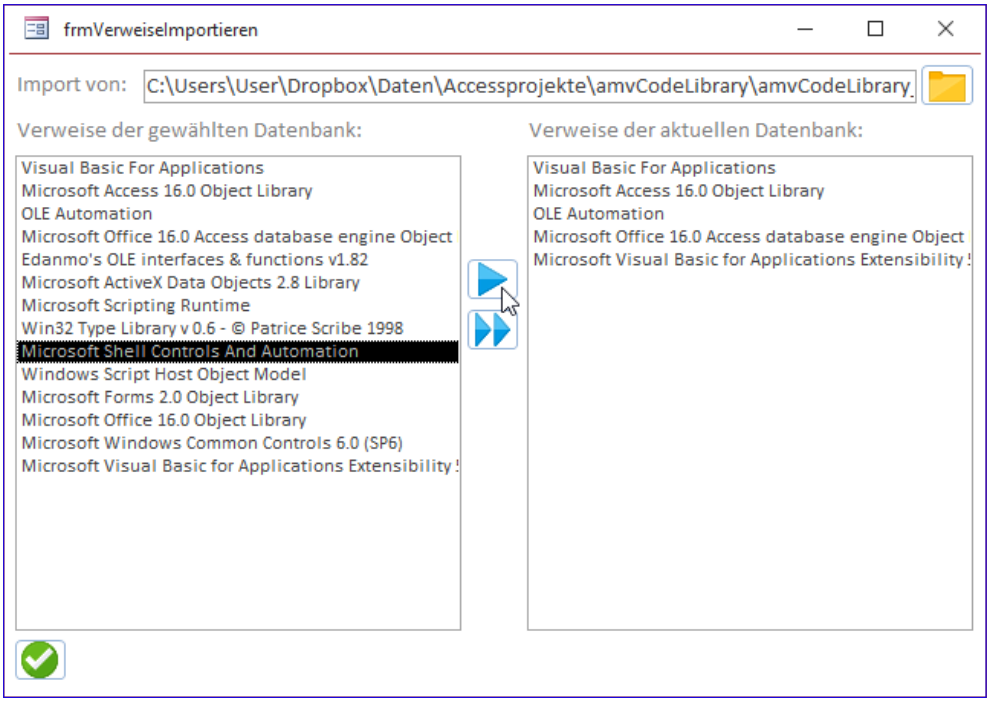


Bild 2: Das Add-In zum Importieren von Verweisen

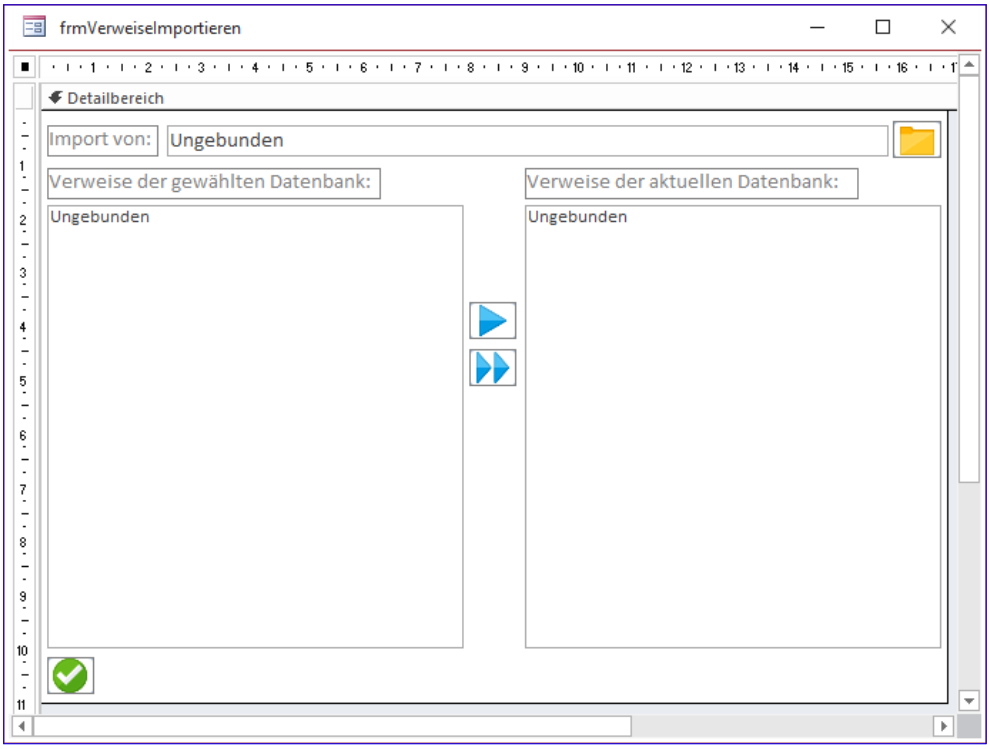


Bild 3: Entwurf des Formulars

Variablen vorbereiten

Im oberen Teil des Klassenmoduls des Formulars **frmVerweiseImportieren** deklarieren wir einige modulweit verfügbare Variablen. Einige davon werden mit dem Typ **VBIDE.VBProject** versehen. Um diese nutzen zu können, benötigen wir einen Verweis auf die Bibliothek **Microsoft Visual Basic for Applications Extensibility 5.3** (siehe Bild 4).

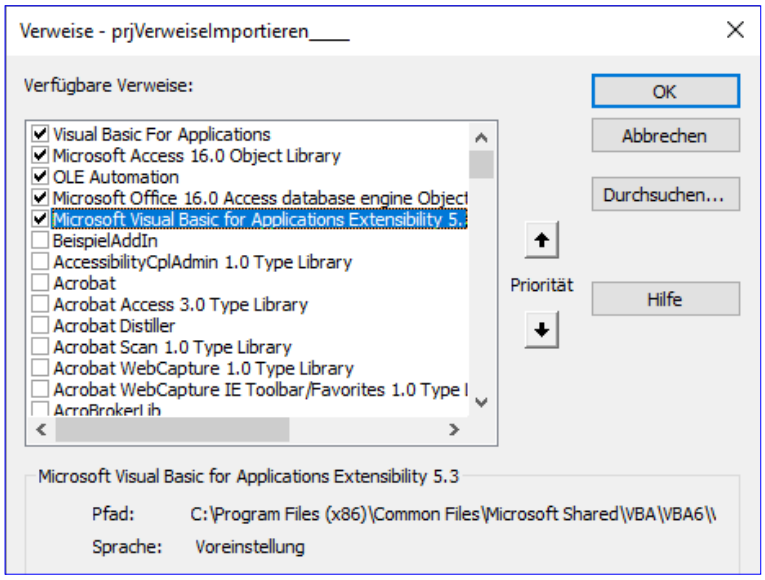


Bild 4: Verweis auf die Bibliothek zur Programmierung der VBIDE hinzufügen

Die Variablendeklarationen sehen wie folgt aus:

```
Dim strAddInName As String
Private m_VBProjectAddIn As VBIDE.VBProject
Private m_VBProjectZiel As VBIDE.VBProject
Dim objVBProjectQuelle As VBIDE.VBProject
```

Die beiden Variablen zum Speichern der Verweise auf die VBA-Projekte von Add-In und Zieldatenbank deklarieren wir als Member-Variablen, weil wir diese später über **Property Get**-Prozeduren füllen und bereitstellen.

Füllen des Listenfeldes mit den vorhandenen Verweisen

Beim Laden des Formulars wird die folgende Prozedur ausgelöst, die wiederum einige weitere Routinen ansteuert:

```
Private Sub Form_Load()
    VerweiseEinlesen objVBProjectZiel, Me!lstZiel
End Sub
```

Referenzieren von AddIn- und Zieldatenbank

Wir greifen in den nachfolgend vorgestellten Prozeduren auf die VBA-Projekte der Add-In-Datenbank, der Datenbank, von der aus wir das Add-In starten und der Datenbank, aus der wir die Verweise importieren wollen, zu.

Zwei davon referenzieren wir durch die folgenden **Property Get**-Prozeduren, die sicherstellen, dass auch im Falle eines Fehlers immer eine Instanz geholt werden kann.

Die Prozeduren sehen wir folgt aus:

```
Public Property Get VBProjectZiel() As VBIDE.VBProject
    If m_VBProjectZiel Is Nothing Then
        Set m_VBProjectZiel = GetVBProject(CurrentDb.Name)
    End If
    Set VBProjectZiel = m_VBProjectZiel
End Property

Public Property Get VBProjectAddIn() As VBIDE.VBProject
```

```
Private Function GetVBProject(strPath As String) As VBIDE.VBProject
    Dim objVBProject As VBIDE.VBProject
    For Each objVBProject In VBE.VBProjects
        If objVBProject.FileName = strPath Then
            Exit For
        End If
    Next objVBProject
    Set GetVBProject = objVBProject
End Function
```

Listing 1: Referenzieren eines VBA-Projekts

```
    If m_VBProjectAddIn Is Nothing Then
        Set m_VBProjectAddIn = GetVBProject(CodeDb.Name)
    End If
    Set VBProjectAddIn = m_VBProjectAddIn
End Property
```

Die Variablen **m_VBProjectZiel** und **m_VBProjectAddIn** füllen wir mit der Funktion **GetVBProject**, der wir den Pfad zu der Datenbankdatei übergeben, zu der das VBA-Projekt gehört (siehe Listing 1).

Wenn Sie von einer Datenbankdatei aus einer Add-In-Datenbank öffnen, befinden sich im VBA-Editor bereits zwei VBA-Projekte (siehe Bild 5).

Um herauszufinden, welches VBA-Projekt zum Add-In gehört, durchläuft die Funktion in einer **For Each**-Schleife alle **VBProject**-Elemente, die aktuell im VBA-Editor verfügbar sind.

Dabei vergleicht sie jeweils den Dateinamen des VBA-Projekts, der mit dem Pfad der Datenbankdatei, den wir mit dem Parameter **strPath** übergeben haben, übereinstimmen muss. Ist das in der **For Each**-Schleife der Fall, verlassen wir diese. Das aktuell in **objVBProject** befindliche VBA-Projekt wird dann als das zu der Datenbank aus **strPath** gehörende VBA-Projekt zurückgegeben.

Auf diese Weise holen wir zwei Verweise auf das VBA-Projekt der Add-In-Datenbank und zu der Datenbank, von der aus das Add-In gestartet wurde und in welche die Verweise einer anderen Datenbank importiert werden sollen.

Für das Ermitteln des Verweises auf die Add-In-Datenbank übernehmen wir den Pfad der Datenbank, den wir mit dem Ausdruck **CodeDb.Name** ermitteln.

Das VBA-Projekt der Datenbank, die das Add-In geöffnet hat, ermitteln wir durch die Übergabe des Pfades aus **CurrentDb.Name** an die Funktion **GetVBProject**.

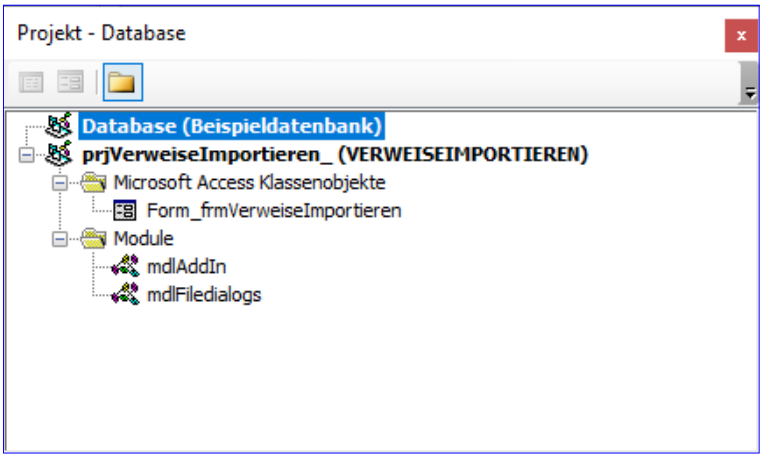


Bild 5: VBA-Projekte der geöffneten Datenbank und der Add-In-Datenbank

```
Private Sub VerweiseEinlesen(objVBProject As VBIDE.VBProject, lst As ListBox)
    Dim objReference As VBIDE.Reference
    Dim strEintrag As String
    lst.RowSource = ""
    For Each objReference In objVBProject.References
        strEintrag = objReference.Description
        On Error Resume Next
        strEintrag = strEintrag & ";" & objReference.Guid
        If Not Err.Number = 0 Then
            strEintrag = strEintrag & ";"
        End If
        Err.Clear
        strEintrag = strEintrag & ";" & objReference.FullPath
        If Not Err.Number = 0 Then
            strEintrag = strEintrag & ";"
        End If
        On Error GoTo 0
        lst.AddItem strEintrag
    Next objReference
End Sub
```

Listing 2: Einlesen der Verweise des VBA-Projekts einer Datenbank

Einlesen der aktuellen Verweise

Die Prozedur **VerweiseEinlesen** liest Verweise des VBA-Projekts der mit dem ersten Parameter angegebenen VBA-Projekt ein und fügt diese zu dem mit dem zweiten Parameter angegebenen Listenfeld hinzu.

Hierzu müssen wir zunächst die Listenfelder vorbereiten. Diese werden nicht an Tabellen oder Abfragen gebunden, sondern sollen ihre Daten aus Wertlisten beziehen.

Daher stellen wir die Eigenschaft **Herkunftstyp** für beide Listenfelder auf **Wertliste** ein. Diese Wertlisten sollen in der ersten Spalte den Namen des Verweises anzeigen und in der zweiten und dritten Spalte die GUID und den Pfad zur referenzierten Bibliothek.

Deshalb stellen wir die Eigenschaft **Spaltenanzahl** auf **3** und **Spaltenbreiten** auf **;0cm;0cm** ein – so wird die erste Spalte über die komplette Breite angezeigt und die zweite und die dritte Spalte werden ausgeblendet.

Die Prozedur **VerweiseEinlesen** aus Listing 2 leert das mit dem Parameter **lst** übergebene Listenfeld zunächst. Dann durchläuft es alle Verweise der Auflistung **Reference** des VBA-Projekts aus **objVBProject**, das wieder mit **GetVBProject** ermittelt wird.

Dann fügt sie der Variablen **strEintrag** den Beschreibungstext aus der Eigenschaft **Description** des Verweises hinzu – dies ist der Inhalt der ersten Spalte. Danach schalten wir die eingebaute Fehlerbehandlung aus, um **strEintrag** um ein Semikolon und den Wert der Eigenschaft **Guid** zu erweitern. Ist dieser leer, wird ein Fehler ausgelöst. In diesem Fall fügen wir einfach nur ein Semikolon zu **strEintrag** hinzu – die zweite Spalte dieses Listenfeldeintrags bliebe also leer. Danach gehen wir auf die gleiche Weise vor, um den Inhalt der Eigenschaft **FullPath** des Verweises zu **strEintrag** hinzuzufügen.

Danach fügt die Prozedur den Inhalt von **strEintrag** mit der Methode **AddItem** des Listenfeldes als neue Zeile zum Listenfeld hinzu.

Damit haben wir nun das rechte Listenfeld gefüllt und der Zeitpunkt ist gekommen, wo der Benutzer die Datenbank auswählen soll, aus der Verweise importiert werden sollen.

Datenbank mit zu importierenden Verweisen auswählen

Die Schaltfläche **cmdDateiauswahl** löst beim Anklicken die Prozedur aus Listing 3 aus. Diese merkt sich zunächst den Namen des VBA-Projekts der Add-In-Datenbank in

```
Private Sub cmdDateiauswahl_Click()
    strAddInName = VBProjectAddIn.Name
    Me!txtImportVon = OpenFileName(CurrentProject.Path, "Datenbank mit zu importierenden Verweisen wählen", _
        "Access-DB (*.mdb,*.accdb,*.mda,*.accda)|Alle DAteien (*.*)")
    On Error Resume Next
    VBProjectAddIn.References.AddFromFile Me!txtImportVon
    If Not Err.Number = 0 Then
        VBProjectAddIn.Name = strAddInName & "_"
        VBProjectAddIn.References.AddFromFile Me!txtImportVon
    End If
    On Error GoTo 0
    Set objVBProjectQuelle = GetVBProject(Me!txtImportVon)
    VerweiseEinlesen objVBProjectQuelle, Me!lstQuelle
End Sub
```

Listing 3: Auswählen der Datenbank zum Importieren der Verweise

der Variablen **strAddInName**. Dann öffnet sie einen **Datei auswählen**-Dialog, der nur Access-Datenbanken anzeigt. Die dazu notwendige Funktion **OpenFileName** finden Sie im Modul **mdlTools**. Das Ergebnis des **OpenFileName**-Aufrufs nach der Auswahl durch den Benutzer landet im Textfeld **txtImportVon**.

Nun wollen wir die Verweise des VBA-Projekts dieser Datenbank auslesen. Dazu binden wir das VBA-Projekt dieser Datenbank als Verweis in das VBA-Projekt der Add-In-Datenbank ein.

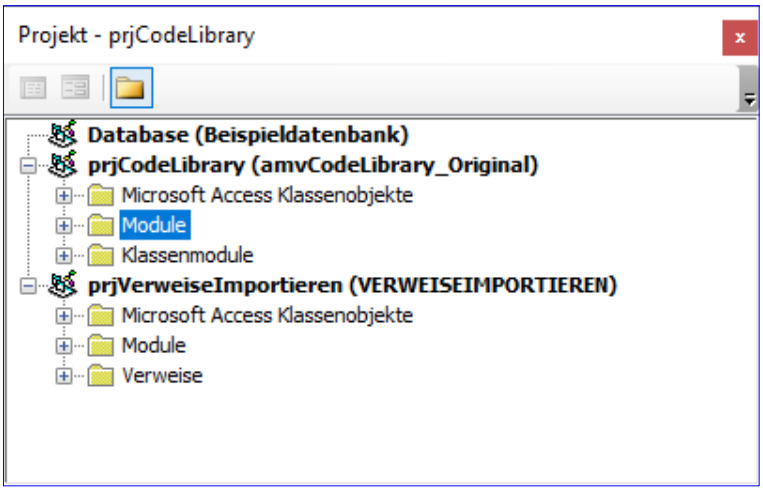


Bild 6: Verweise auf drei VBA-Projekte gleichzeitig im VBA-Editor

Das erledigen wir in der folgenden Anweisung, wo wir der **References**-Auflistung des aktuellen VBA-Projekts aus **objVBProjectAddIn**, also des Add-In-Projekts, einen Verweis auf die Datenbank zu, deren Verweise wir importieren wollen.

Tritt dabei ein Fehler auf, liegt es vermutlich daran, dass der Name des zu referenzierenden Projekts mit dem Namen des Add-In-VBA-Projekts übereinstimmt – was normalerweise nicht passieren sollte, außer die Datenbank heißt ebenfalls prjVerweiseImportieren wie das VBA-Projekt unseres Add-Ins. Der resultierende Fehler lautet **Name steht in Konflikt mit vorhandenem Modul, Projekt oder vorhandener Objektbibliothek**.

In diesem Fall passen wir den Namen des VBA-Projekts aus **objVBProjectAddIn** an, indem wir einen Unterstrich an den Projektnamen anhängen. Und nun wird auch klar, warum wir weiter oben den Projektnamen in der Variablen **strAddInName** gespeichert haben – damit wir den Namen des VBA-Projekts nach Abschluss der Arbeit mit dem Add-In wieder in den vorherigen Zustand zurückversetzen können.

Im Projekt-Explorer im VBA-Editor finden Sie nun sogar gleich drei VBA-Projekte vor – das der aktuell geöffneten Datenbank, das für das Add-In und das der Datenbank, deren Verweise importiert werden sollen (siehe Bild 6).

Dann weisen wir der Variablen **objVBProjectQuelle** das VBA-Projekt zu aus der Liste der aktuell verfügbaren VBA-Projekte zu, das den gleichen Dateinamen hat wie die Datei, die der Benutzer über den **Datei auswählen**-Dialog selektiert hat.

Schließlich rufen wir noch die Prozedur **VerweiseEinlesen** auf. Dabei geben wir mit dem ersten Parameter einen Verweis auf das VBA-Projekt mit den Verweisen und als zweiten Parameter den Verweis auf das linke Listenfeld an (**lstQuelle**).

Danach zeigen nun beide Listenfelder die Verweise der jeweiligen Datenbanken an und wir können uns um die Programmierung für den Import oder für das Entfernen von Verweisen aus der aktuell geöffneten Datenbank kümmern.

Einzelnen Verweis importieren

Den Import eines einzelnen Verweises aus der Quelldatenbank erledigen Sie beispielsweise durch einen Klick auf die Schaltfläche **cmdHinzufuegen**.

Dies löst die folgende Ereignisprozedur aus:

```
Private Sub cmdHinzufuegen_Click()
    If Not IsNull(Me!lstQuelle) Then
```