HAUFE.
Ausgabe 03/2023

ACCESS
IM UNTERNEHMEN

DATUM UND ZEIT IM GRIFF

Keine Datenbank kommt ohne das Speichern von Datums- und Zeitangaben aus. Ab Seite 2 und Seite 34 finden Sie wichtige grundlegende Informationen zu diesem Thema in Microsoft Access.

In diesem Heft:

Datum und Zeit

Es gibt sicher nur wenige Datenbankanwendungen, in denen nicht mindestens in einer Tabelle Felder zum Speichern von Datum und Uhrzeit vorkommen. Sei es, um Geburtstage zu speichern oder einfach nur den Zeitpunkt, wann ein Datensatz hinzugefügt oder gelöscht wurde. In dieser Ausgabe von Access im Unternehmen kümmern wir uns gleich in zwei Beiträgen um dieses Thema: Wir schauen uns an, wie Datum und Zeit überhaupt gespeichert werden und wie man mit diesen Daten rechnen kann.

Wie bereits in der Einleitung erwähnt, dreht es sich in dieser Ausgabe um das Thema Datum und Zeit. Der erste Beitrag dazu heißt **Datum und Zeit mit Access** und ist ab Seite 2 zu finden. Die wichtigste Information ist, wie Datum und Zeit überhaupt gespeichert werden. Dahinter verbirgt sich nämlich kein spezieller Datentyp, sondern eine einfache **Double**-Zahl, deren Stellen vor dem Komma das Datum und deren Nachkommastellen die Uhrzeit angeben.

Ausgestattet mit dieser Information untersuchen wir im Beitrag **Mit Zeiten rechnen** ab Seite 34 in Abfragen und schließlich auch in Berichten, wie wir mit Datums- und Zeitwerten rechnen. Hier gehen wir auch auf die oft gestellten Fragen ein, wie man unter Access die Differenz zwischen verschiedenen Datums- und Zeitangaben ermittelt und wie man Zeiten aufsummieren kann.

Wer bereits einmal Schaltflächen mit Icons ausgestattet hat, nutzt dabei vermutlich die Möglichkeit, die dazu notwendigen Bilddateien in der jeweiligen Access-Datenbank in einer internen Tabelle namens **MSysResources** zu speichern. Öffnet man den dazu notwendigen Befehl über das Ribbon, kann man jedoch nur jeweils eine einzige Bilddatei zur Datenbank hinzufügen. Verwendet man immer einen Satz bestimmter Bilddateien, muss man diese jeder neuen Datenbank einzeln hinzufügen. Das erleichtern wir mit den Techniken, die wir im Beitrag **Bilder für Buttons und Co. schnell hinzufügen** ab Seite 12 erläutern. Noch einen Schritt weiter gehen wir im Beitrag **Bilder per COM-Add-In hinzufügen** ab Seite 65. Hier zeigen wir eine Lösung, mit der Sie die Funktion des eingebauten Ribbonbefehls durch die im vorherigen Beitrag erläuterte Technik ersetzen können. Damit ist das mühselige Einfügen einzelner Bilddateien obsolet.

Im Beitrag **Ribbon: Controls erkennen und ausblenden** (ab Seite 18) wollten wir eigentlich nur erklären, wie man die Ribbon-Einträge im Dateimenü von Access ausblendet. Wir haben dabei aber einige spannende Methoden gefunden, um die Namen der eingebauten Elemente herauszufinden, damit man diese gezielt referenzieren und anpassen kann.

Ein oft gemeldetes Problem ist eine Datensatzsperrung, obwohl scheinbar niemand sonst auf diesen Datensatz zugreift. Die Ursache kann ein Memofeld sein. Wie Sie das Problem erkennen und lösen, zeigen wir im Beitrag **Sperrung durch Memofeld statt anderer Sitzung** ab Seite 26.

Schließlich liefert diese Ausgabe noch eine Lösung zum Anzeigen von Bezahlcodes in Rechnungen (**Rechnungsbericht mit EPC-QR-Code**, ab Seite 39), einen Satz zur Behebung von Verweisproblemen (**Probleme bei Verweisen mit verschiedenen Versionen**, ab Seite 51) und Techniken, um Dateien automatisiert aus dem Internet herunterzuladen (**Dateien aus dem Web herunterladen per VBA**, ab Seite 58).

Viel Spaß beim Erkunden der neuen Ausgabe!

Ihr André Minhorst

Datum und Zeit mit Access

Das Speichern von Datum- und Zeitwerten mit Access birgt wenige Geheimnisse. Sie fügen einer Tabelle ein Feld vom Datentyp »Datum/Zeit« hinzu und schon können Sie Werte in dieses Feld eingeben. Die tägliche Praxis im Umgang mit Datums- und Zeitwerten stellt jedoch höhere Anforderungen. Mit den Tipps aus dem folgenden Beitrag können Sie die meisten der sich stellenden Fragen beantworten.

Ein Wert mit vielen Gesichtern

Innerhalb Deutschlands geben die meisten Menschen einen Datumswert intuitiv als 5.3.2023, als 05.03.2023 oder schreibfaul als 5.3.23 oder mit Mischformen ein. Access versteht all diese Schreibweisen und interpretiert sie als den gleichen Tag, den 5.3.2023.

Selbst Angaben im Klartext wie beispielsweise 5. März 2023 sind bei der Dateneingabe erlaubt. Sie müssen lediglich darauf achten, dass Ihre Eingabe den in der Systemsteuerung von Windows getätigten Ländereinstellungen entspricht.

Wie speichert Access Datum- und Zeitwerte?

Dass Access Ihre Eingabe richtig versteht, können Sie leicht feststellen, indem Sie ein Formular mit einem Textfeld namens **txtDatumZeit** anlegen.

Fügen Sie dem Formular ein weiteres Textfeld namens **txtDatum** mit dem folgenden Steuerelementinhalt hinzu:

```
=Format([txtDatumZeit];"tt.mm.jjjj")
```

Mit einem weiteren Textfeld namens **txtZeit** können Sie die Zeitangabe aus dem Eingabewert ermitteln. Geben Sie für dieses den folgenden Steuerelementinhalt ein:

```
=Format([txtDatumZeit];"hh:nn:ss")
```

Access erkennt selbstständig, dass es sich bei dem eingegebenen Text um einen Datum- und Zeitwert handelt und wandelt ihn in automatisch in diesen Datentyp um. Die **Format**-Funktion formatiert den Wert aus dem Steuerelement **txtDatumZeit** im ersten Fall als Datums- und im zweiten Fall als Zeitwert. Mit **tt.mm.jjjj** erhalten Sie die Anzeige mit zweistelligem Tages-, zweistelligem Monats- und vierstelligem Jahreswert. Das Zeitformat zeigt Stunden, Minuten und Sekunden ebenfalls zweistellig an.

Nun können Sie nacheinander verschiedene Datumswerte in das Textfeld eingeben und beobachten, welchen Datumswert und welchen Zeitwert Access ermittelt.

Ein Beispiel hierfür sehen Sie in Bild 1.

Sie können allerlei verschiedene Eingaben für den gleichen Tag tätigen. Wenn Sie wissen wollen, wie Access die Werte intern speichert, dann fügen Sie dem Formular drei weitere Steuerelemente mit folgenden Formeln als Steuerelementinhalt hinzu:

```
=ZDouble(ZDate([txtDatumZeit]))
=DatWert([txtDatumZeit])
=ZeitSeriellStr([txtDatumZeit])
```

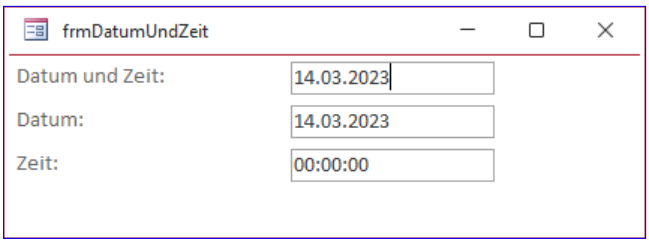

Bild 1: Mit diesem Formular können Sie prüfen, ob Access Ihre Datums- und Zeiteingabe versteht.

Besonders interessant ist die **ZDate**-Funktion (englisch: **CDate**). Diese Funktion wandelt irgendeinen Wert, in diesem Fall die eingegebene Zeichenkette, in den Datentyp **DatumZeit** um. Der anschließende Aufruf von **ZDouble** (englisch: **CDbl**) führt dazu, dass Access den berechneten Wert als Zahl mit Nachkommastellen anzeigt. Ein Beispiel hierfür sehen Sie in Bild 2.

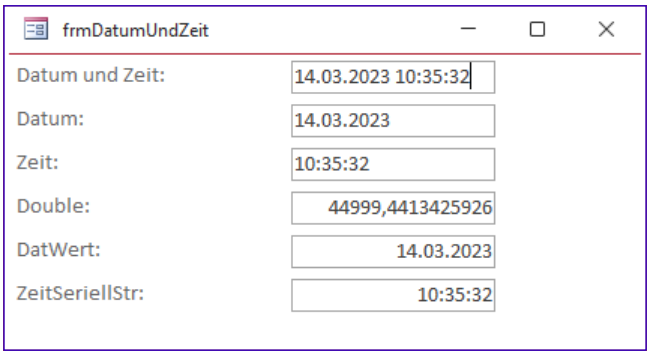

Bild 2: Beispiel für die Umrechnung eines Datums- und Zeitwertes in einen Double-Wert

Access wandelt beispielsweise **14.03.2023 10:35:32** in den **Double**-Wert **44999,4413425926** um. Der Grund hierfür besteht darin, dass Access Datums- und Zeitwerte als **Double**-Werte kodiert.

Der ganzzahlige Anteil speichert die Anzahl der Tage in Bezug auf den **30.12.1899**. Negative Werte geben die Anzahl der Tage vor diesem Datum an und positive Werte die Anzahl der Tage nach diesem Datum.

Der Nachkommanteil entspricht jeweils der seit 0 Uhr vergangen Zeit. Der Zeitpunkt **30.12.1899 0:00:00** entspricht dem Double-Wert **0,0**.

Rechnen mit Datum und Zeit

Aufgrund der internen Umrechnung in **Double**-Werte können Sie mit Datums- und Zeitwerten rechnen. Bei der Addition von zwei Werten erhalten Sie absolut sinnvolle Ergebnisse. Das in Bild 3 gezeigte Beispielformular enthält zwei Textfelder zur Dateneingabe.

Der eingegebene Text wird jeweils wie oben beschrieben mit **ZDate()** und **ZDouble()** in einen **Double**-Wert umgerechnet. Das Textfeld **txtSumme** berechnet dann die Summe aus den beiden Werten. Das unterste Textfeld zeigt den berechneten Wert als Standarddatum formatiert an.

Im Beispiel wurden zu dem **Datum/Zeit**-Wert **14.03.2023 23:00:00** zwei Stunden hinzuaddiert. Als Ergebnis erhalten Sie den **Datum/Zeit**-Wert **15.03.2023 01:00:00**.

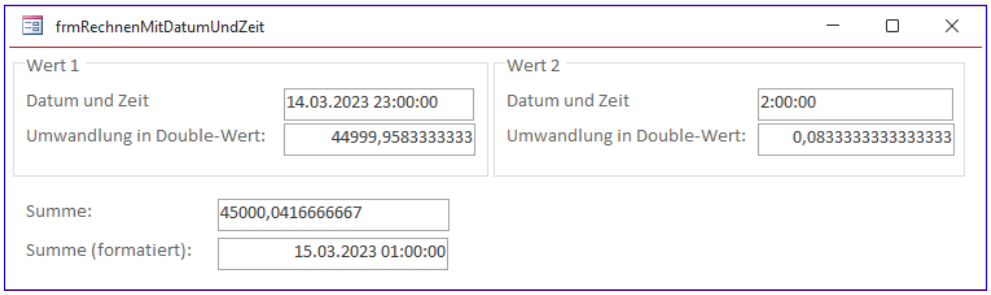

Bild 3: Beispiel für die Addition von zwei Datum-/Zeit-Werten

Hinweis: Beim Rechnen mit **Datum/Zeit**-Werten dürfen Sie nur positive Zahlen oder nur ganze Zahlen verwenden. Bei Datumswerten vor dem **30.12.1899** wird dem negativen ganzzahligen Anteil der positive Nachkommaanteil hinzugerechnet.

Wenn Sie beispielsweise dem **Datum/Zeit**-Wert **15.12.1899 23:00:00** (entspricht **-15,9583333333333**) zwei Stunden (entspricht **0,0833333333333333**) hinzuaddieren, dann erhalten Sie das fehlerhafte Ergebnis **15.12.1899 21:00:00**.

Funktionen für Datum und Zeit

Access verfügt über eine ganze Reihe eingebauter Funktionen für die Berechnung von Datum- und Zeit-Werten. Mit der **Jetzt**-Funktion erhalten Sie die aktuelle Systemzeit inklusive Datum. Die **Datum**-Funktion liefert nur das Datum, die **Jetzt**-Funktion nur die Zeit.

Addieren von Zeitintervallen

Mit der **DatAdd**-Funktion (englisch: **DateAdd**) können Sie eine Sekunde (**s**), eine Minute (**n**), eine Stunde (**h**), einen Tag (**d**), einen Wochentag (**w**), einen Tag des Jahres (**y**), eine Woche (**ww**), einen Monat (**m**), ein Quartal (**q**) oder ein Jahr (**yyyy**) zu einem Datumswert hinzuaddieren. Die in Klammern angegebenen Werte können Sie als Parameter für die **DatAdd**-Funktion verwenden. Bild 4 zeigt exemplarisch das Addieren einer Woche und eines Monats zu einem Datumswert.

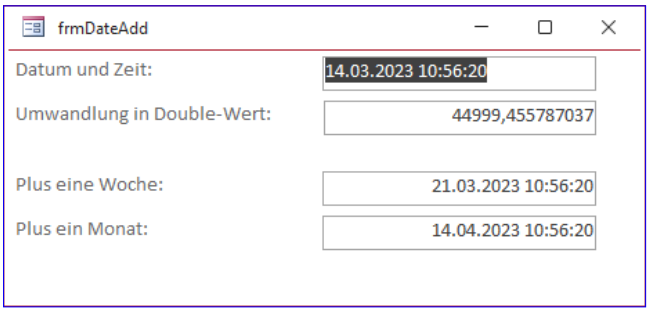

Bild 4: Addition einer Woche und eines Monats zu einem Datumswert

Die in der Abbildung gezeigten Werte können Sie mit den folgenden beiden Ausdrücken berechnen:

```
=DatAdd("ww";1;[txtDouble1])
=DatAdd("m";1;[txtDouble1])
```

Differenz in Tagen, Wochen und Monaten

Für viele Berechnungen benötigen Sie die Differenz zwischen zwei Datumswerten nicht einfach nur als **Double**-Zahl, sondern als konkret benanntes Intervall.

Mit der **DateDiff**-Funktion ermitteln Sie die Differenz zwischen zwei Zeitpunkten in Sekunden (**s**), Minuten (**n**), Stunden (**h**), Wochen (**ww**), Wochentagen (**w**), Tagen (**d**), Tagen des Jahres (**y**), Monaten (**m**), Quartalen (**q**) und Jahren (**y**) . So berechnet beispielsweise der Ausdruck

```
=DatDiff("d";[txtDouble1];[txtDouble2];2;2)
```

die Differenz zwischen zwei Datumswerten in Tagen. Der erste Parameter bezeichnet das Intervall. Das **d** für englisch **Day** veranlasst Access, die Differenz in Tagen zu berechnen. Der zweite und dritte Parameter bezeichnen zwei Steuerelemente mit Datumswerten.

Parameter Nummer vier und fünf sind optional, sollten aber im deutschsprachigen Raum angegeben werden.

Diese beiden Parameter bestimmen, wie Access die Kalenderwochen zählt. Mit der ersten **2** legen Sie fest, dass der erste Tag einer Woche der Montag (**2** = **vbMonday**) ist. Mit der zweiten **2** legen Sie fest, dass die erste Kalenderwoche eines Jahres die erste Woche im neuen Jahr mit mindestens 4 Tagen ist.

Etwas seltsam ist die Unterscheidung zwischen der Differenz in Wochentagen (**w**) und Wochen (**ww**). Falls Sie dieses Thema interessiert, sollten Sie sich den genauen Text in der Hilfe durchlesen.

Eine verständliche Interpretation des Hilfetextes könnte wie folgt lauten: Bei Verwendung des Parameters **w** berechnet Access die Anzahl voller Kalenderwochen mit 7 Tagen, die zwischen den beiden Datumswerten liegen.

Bei Verwendung des Parameters **ww** zählt Access die Anzahl der Sonntage zwischen den beiden Datumswerten. Dabei wird das zweite Datum nicht mitgezählt, sofern es auf einen Sonntag fällt. Dieser Wert ist entweder gleich oder größer als der mit **w** ermittelte.

Wenn Sie wie in Bild 5 beispielsweise die Wochen von Sonntag, den 5.3.2023 bis Montag, den 13.3.2023 mit **DateDiff** berechnen, dann erhalten Sie bei Verwendung des Parameters **w** ein Ergebnis von einer Woche und bei Verwendung des Parameters **ww** ein Ergebnis von zwei Wochen. Tatsächlich liegen zwischen den beiden Zeitpunkten acht Tage.

Wenn Sie hingegen die Differenz von Sonntag, den 5.3.2023 und Sonntag, den 12.3.2023 berechnen, dann

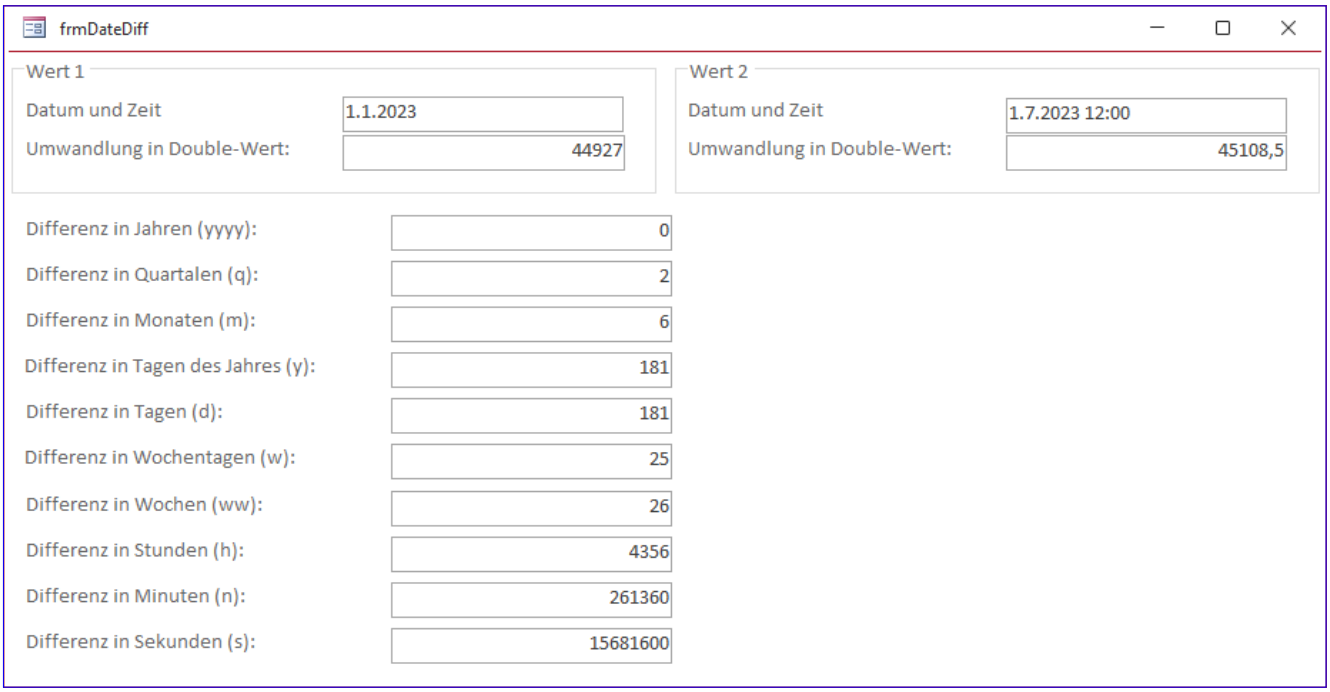

Bild 5: Berechnungen mit **DateDiff**

ermitteln beide Parameter den gleichen Wert **1** als Ergebnis. Der Grund besteht darin, dass auch bei Verwendung von **ww** der zweite Sonntag nicht mitgezählt wird.

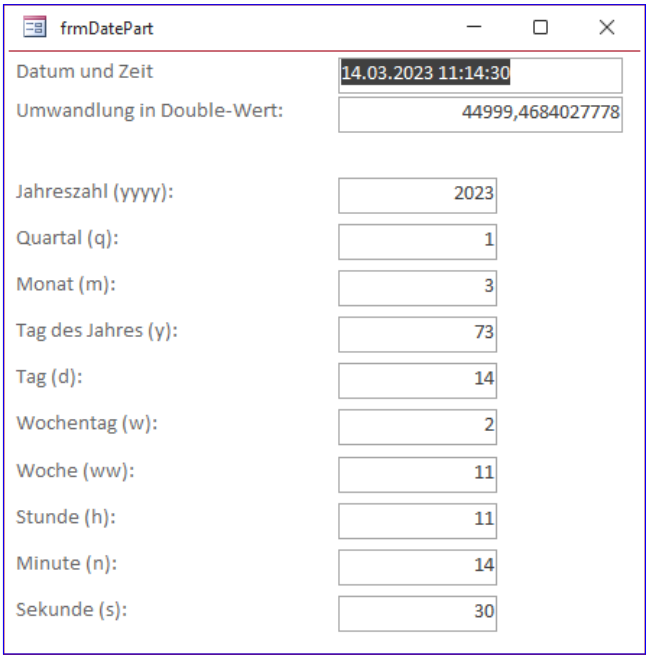

Bild 6: Ergebnisse der **DatTeil**-Funktion

Datumsteile ermitteln

Mit der **DatTeil**-Funktion (englisch: **DatePart**) können Sie die Jahreszahl (**yyyy**), das Quartal (**q**), den Monat (**m**), den Tag des Jahres (**y**), den Tag (**d**), den Wochentag (**w**), die Woche (**ww**), die Stunde (**h**), die Minute (**n**) oder die Sekunde (**s**) aus einem Zeitpunkt berechnen. Mit dem folgenden Ausdruck berechnen Sie beispielsweise den Tag des Jahres des im Steuerelement **txtZeitpunkt** gespeicherten Zeitpunkts:

```
=DatTeil("y";[txtZeitpunkt];2;2)
```

Am 8.3.2023 ist dies der Wert **73**, am 31.12 des Schaltjahres 2024 ist es der Wert **366**. Bild 6 enthält eine Übersicht der verschiedenen Datumsteile.

Typen umwandeln

Die **DatSeriell**-Funktion (englisch: **DateSerial**) ermittelt einen Datumswert aus den in dieser Reihenfolge angegebenen Werten **Jahr**, **Monat** und **Tag**.

Ein Beispiel lautet:

Bilder für Buttons und Co. schnell hinzufügen

Wenn Sie mit Access ein neues Bild zu einem Formular hinzufügen oder dieses für eine Schaltfläche hinterlegen, speichert Access dieses in einer Tabelle namens MSysResources. Das Hinzufügen ist recht einfach und lässt sich per Ribbonbefehl und anschließender Auswahl per Datei-öffnen-Dialog realisieren. Allerdings kann man auf diese Weise immer nur ein Bild gleichzeitig zur Anwendung hinzufügen. Wenn man einer Schaltfläche schnell ein Icon zuweisen möchte, wäre es praktisch, wenn alle üblicherweise benötigten Bilder bereits verfügbar sind. Daher schauen wir uns in diesem Beitrag an, wie wir mehrere Bilder auf einen Schlag zur Tabelle MSysResources hinzufügen können.

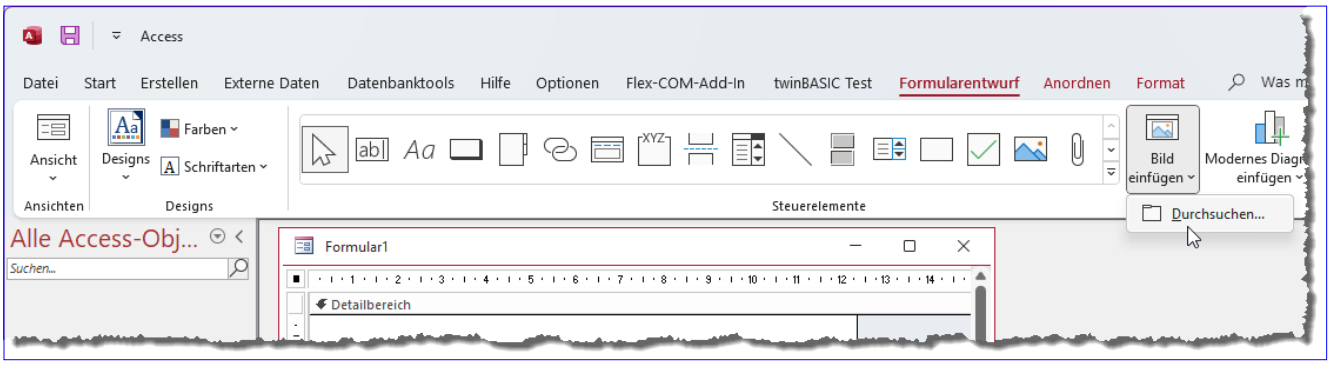

Bild 1: Bild zu einem Formular hinzufügen

Wo landen die Bilder für Schaltflächen und Co.?

Bevor wir uns ansehen, wie wir schnell viele Bilder zur Anwendung hinzufügen, werfen wir einen Blick auf den Speicherort dieser Bilddateien.

Gemeint sind solche Bilder, die wir beispielsweise im Entwurf eines Formulars oder Berichts hinzufügen. Dazu gehen wir normalerweise wie folgt vor:

- Wir öffnen das Zielformular in der Entwurfsansicht.
- Dann klicken wir im Ribbon auf den Befehl **Bild einfügen\|Durchsuchen** im Bereich **Steuerelemente** (siehe Bild 1).
- Bewegen wir den Mauszeiger dann über den Formularentwurf, befindet sich der Mauszeiger im Modus zum

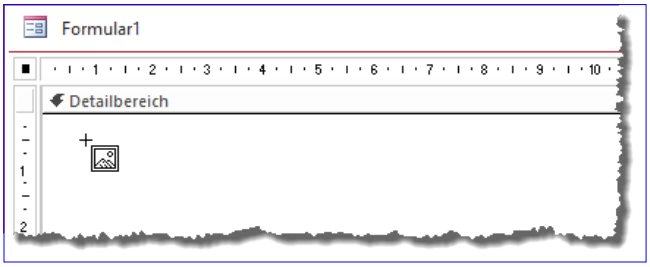

Bild 2: Bildsteuerelement einfügen

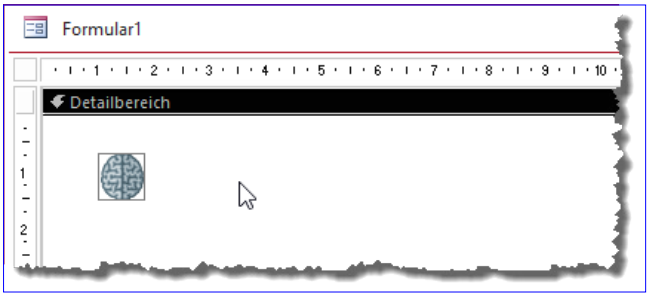

Bild 3: Das eingefügte **Bild**-Steuerelement

Bild 4: Eigenschaften des Bild-Steuerelements

Einfügen eines **Bild**-Steuerelements (siehe Bild 2).

- Nach dem Anklicken des Zielpunktes für das Steuerelement öffnet sich der Dialog **Grafik einfügen**. Hier wählen wir die gewünschte Bilddatei, zum Beispiel mit dem Format **.png**.

- Nach der Auswahl erscheint das **Bild**-Steuerelement mit dem gewünschten Icon im Formularentwurf (siehe Bild 3).

Wo ist das Bild nun gelandet? Ist es eingebettet? Oder verknüpft? Nun: Es gibt noch einen dritten Zustand, den wir sehen, wenn wir in den Eigenschaften des **Bild**-Steuerelements auf der Seite **Format** die Eigenschaft **Bildtyp** betrachten.

Diese hat den Wert **Freigegeben** (siehe Bild 4).

Was heißt das? Das bedeutet, dass die Bilddatei an zentraler Stelle gespeichert wurde und auch für die Verwendung in anderen Steuerelementen in anderen Formularen und Berichten geeignet ist. Um das nachzuweisen, fügen wir dem Formular eine neue Schaltfläche hinzu. Für diese können wir nun auch das Bild **brain** auswählen (siehe Bild 5).

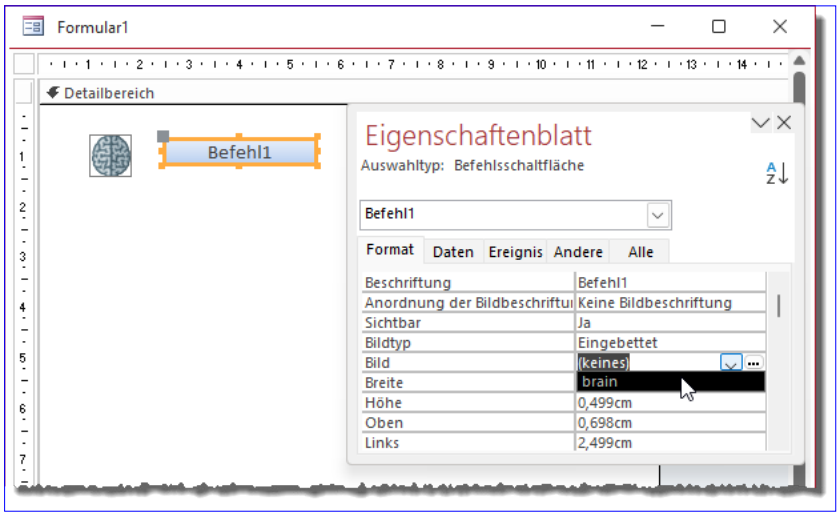

Bild 5: Gleiches Bild für eine Schaltfläche auswählen

Schließlich finden wir auch noch im Ribbon das bereits hinzugefügte Bild als Auswahl für neu anzulegende Bildsteuerelemente – siehe Bild 6.

Speicherort der »freigegebenen« Bilddateien

Wo aber werden diese Bilder gespeichert? Diese landen in einer Systemtabelle, die üblicherweise nicht im Navigationsbereich angezeigt wird. Um diese sichtbar zu machen, klickt man mit der rechten Maustaste auf den Titel des Navigationsbereichs und wählt aus dem Kontextmenü den Befehl **Navigationsoptionen...** aus. Im nun erscheinenden Dialog wählt man im Bereich **Anzeigeoptionen** die beiden Optionen **Ausgeblendete Objekte anzeigen** und **Systemobjekte anzeigen** aus.

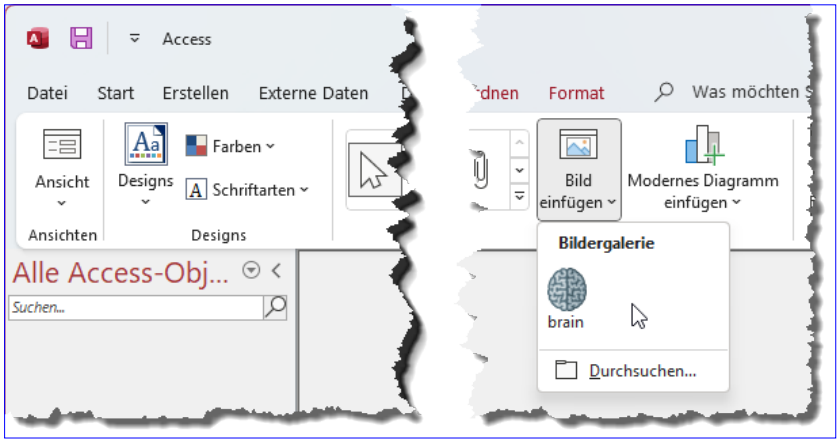

Bild 6: Das selektierte Bild erscheint im Ausklappmenü.

Danach sehen wir im Navigationsbereich die Tabelle **MSysResources**, die in der Datenblattansicht nach dem Öffnen wie in Bild 7 aussieht.

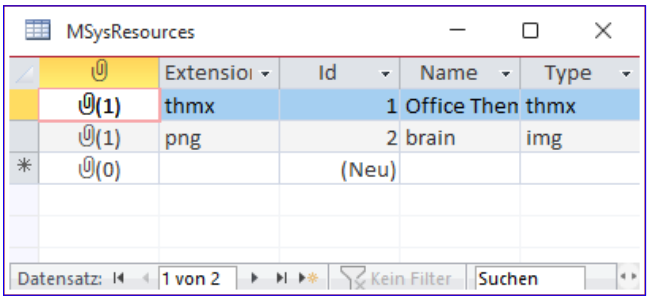

Bild 7: Bilddatei in der Tabelle **MSysResources**

Zwischenstand

Wir wissen also nun, dass von uns zu Schaltflächen oder **Bild**-Steuerelementen hinzugefügte Bilder in der Tabelle **MSysResources** landen. Wenn wir eine speziellen Sammlung an Bildern vorab in diese Tabelle laden wollen, damit wir beim Entwerfen von Formularen und Berichten aus diesem Fundus schöpfen können, müssen wir jedes Bild einzeln zur Anwendung und damit zur Tabelle **MSysResources** hinzufügen.

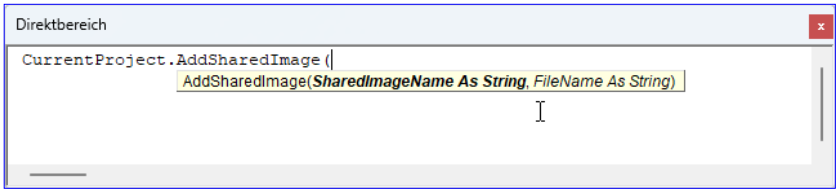

Bild 8: Die Methode **AddSharedImage** der **CurrentProject**-Klasse

Bilder schnell hinzufügen per VBA

Ich hatte mal ein Add-In entwickelt, mit dem dies schneller ging, aber dieses hat aufwendige Prozeduren verwendet, um dies zu realisieren.

Zwischendurch hat Microsoft aber still und heimlich einen Befehl zur Klasse **CurrentProject** hinzugefügt (ich bin mir zumindest sicher, dass dieser früher noch nicht vorhanden war).

Diese Methode erwartet zwei Parameter:

- Name des Bildes, wie er in der Liste erscheinen soll
- Pfad zur Bilddatei, die importiert werden soll

Im Direktbereich kann man diese Methode wie in Bild 8 aufrufen.

Wir haben einige **.png**-Dateien direkt im Verzeichnis der Datenbank gespeichert und für den ersten Versuch die Methode wie folgt aufgerufen:

```
CurrentProject.AddSharedImage "add", _
    CurrentProject.Path & "\add.png"
```

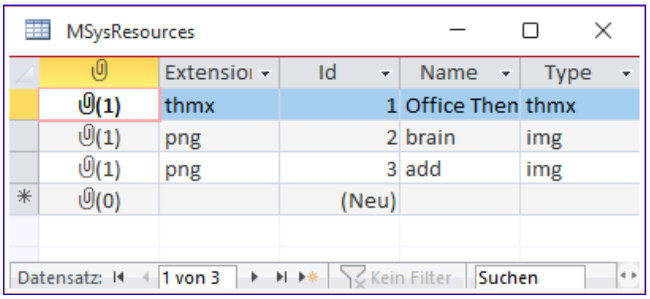

Bild 9: Die neue Bilddatei in der Tabelle **MSysResources**

Schauen wir uns das Ergebnis in der Tabelle **MSysResources** an, war das Einfügen erfolgreich (siehe Bild 9). Und eine weitere Probe zeigte auch, dass das neue Bild an den verschiedenen Stellen zur Auswahl bereitsteht.

Es steht also nichts mehr im Wege, eine kleine Prozedur zu schreiben, mit der wir schnell viele Icons in die Tabelle **MSysResources** laden können!

Viele Icons per Prozedur laden

Dazu benötigen wir nicht viel:

- Dialog zur Auswahl der einzufügenden Bilder
- Code, der die dort ausgewählten Dateien durchläuft und mit **AddSharedImage** zur Datenbank hinzufügt

Ribbon: Controls erkennen und ausblenden

Das Dateimenü von Access bietet eine ganze Reihe praktischer Elemente. Diese sind im Alltagsgebrauch eines Entwicklers sehr hilfreich, bieten sie doch schnellen Zugriff auf wichtige Funktionen. Wollen wir eine Anwendung entwickeln und diese an den Benutzer weitergeben, sollen jedoch unter Umständen gar nicht all diese Befehle verfügbar sein. Vielleicht wollen wir zusätzlich oder statt dieser Einträge sogar eigene Elemente hinzufügen, beispielsweise für das Speichern von Anwendungsoptionen. In diesem Fall ist der erste Schritt, einen oder mehrere der vorhandenen Einträge auszublenden. Wie das gelingt, zeigen wir in diesem Beitrag.

Genau wie die Elemente des Ribbons können wir auch die Einträge im Backstage-Bereich manipulieren, indem wir diese ausblenden, ergänzen oder ihre Funktion verändern.

Standardmäßig finden wir hier einige hilfreiche Funktionen, die wir in Bild 1 sehen – hier am Beispiel von Access in der Version, die mit Microsoft 365 geliefert wird.

In den folgenden Abschnitten schauen wir uns an, wie wir die eingebauten Elemente für die hier verwendete Access-Version ausblenden können. Das erledigen wir wie folgt:

- Erstellen einer Tabelle zum Speichern der benötigten Ribbondefinition
- Zusammenstellen der Ribbondefinition und Speichern der Definition in der Tabelle
- Einstellen der Ribbondefinition als Anwendungsribbon

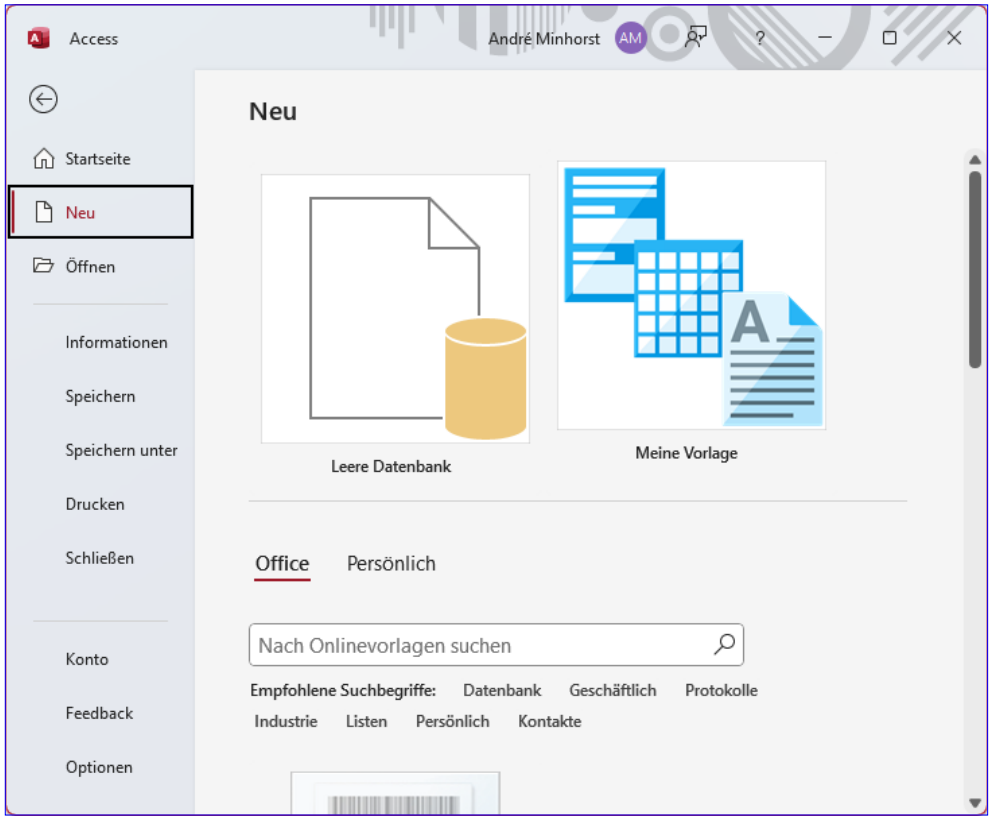

Bild 1: Das Dateimenü von Microsoft Access

Tabelle für die Ribbondefinition

Die Tabelle für die Ribbondefinition legen wir unter dem Namen **USysRibbons** an. Sie enthält drei Felder: **RibbonID** (Autowert, Primärschlüsselfeld), **RibbonName** (kurzer

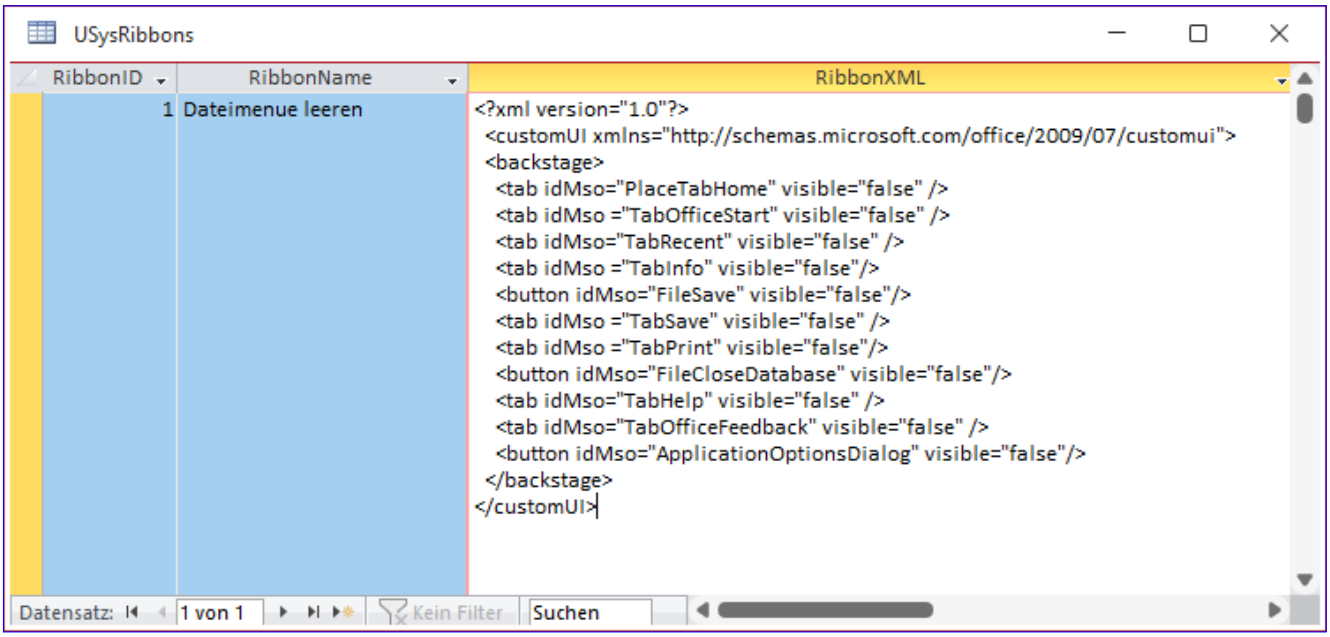

Bild 2: Tabelle mit der Ribbondefinition zum Leeren des Dateimenüs

Text) und **RibbonXML** (langer Text). Nach dem Anlegen und Speichern verschwindet diese Tabelle scheinbar im Nirwana – zumindest wird sie normalerweise nicht im Navigationsbereich angezeigt. Das ist jedoch so gewollt: Der Name der Tabelle beginnt mit **USys...** – und damit interpretiert Access, dass es sich um eine Systemtabelle handelt, die nicht angezeigt werden soll. Um diese zu sehen, aktivieren wir die Anzeige der Systemtabellen. Dazu klicken wir mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs und wählen aus dem Kontextmenü den Eintrag **Navigationsoptionen...** aus. Im nun erscheinenden Dialog aktivieren wir die Option **Systemobjekte anzeigen**.

```
<?xml version="1.0"?>
  <customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <tab idMso="PlaceTabHome" visible="false" />
    <tab idMso ="TabOfficeStart" visible="false" />
    <tab idMso="TabRecent" visible="false" />
    <tab idMso ="TabInfo" visible="false"/>
    <button idMso="FileSave" visible="false"/>
    <tab idMso ="TabSave" visible="false" />
    <tab idMso ="TabPrint" visible="false"/>
    <button idMso="FileCloseDatabase" visible="false"/>
    <tab idMso="TabHelp" visible="false" />
    <tab idMso="TabOfficeFeedback" visible="false" />
    <button idMso="ApplicationOptionsDialog" visible="false"/>
  </backstage>
</customUI>
```

Listing 1: Ausblenden der Dateimenü-Einträge

Nun öffnen wir die Tabelle und fügen einen neuen Datensatz wie in Bild 2 ein.

Ribbondefinition zusammenstellen

Diesem fügen wir im Feld **RibbonXML** den Code aus Listing 1 hinzu. Er enthält ein **customUI**-Element, dem direkt das **backstage**-Element untergeordnet ist. Unmittelbar im **backstage**-Element finden wir dann verschiedene Definitionen für **tab**- beziehungsweise **button**-Elemente.

Im Gegensatz zu früheren Access-Versionen kann man im Dateimenü nicht mehr erkennen, ob es sich bei einem Eintrag um ein **button**- oder ein **tab**-Element handelt.

Der Unterschied ist, dass das **button**-Element direkt eine Aktion auslöst und das **tab**-Element den entsprechenden Bereich rechts öffnet.

Wie aber finden wir nun heraus, welche **idMso** zu den angezeigten Einträgen im Dateimenü gehört? Für die **idMsos** gibt es Excel-Tabellen mit allen Elementen. Wir haben uns ein wenig Arbeit gemacht und diese manuell ermittelt. Dabei haben wir die folgenden Zuordnungen gefunden, die zum Teil nicht gerade intuitiv zu finden sind:

- **PlaceTabHome**: **Startseite**
- **TabOfficeStart**: **Neu**
- **TabRecent**: **Öffnen**
- **TabInfo**: **Informationen**
- **FileSave**: **Speichern**
- **TabSave**: **Speichern unter**
- **TabPrint**: **Drucken**
- **FileCloseDatabase**: **Schließen**
- **TabHelp**: **Konto**
- **TabOfficeFeedback**: **Feedback**

tblAccessControls

ID	ControlName	ControlType	TabSet	Tab	GroupContextMe
1442	SkyDriveUpsellGroup	group	None (Backstage View)	TabRecent	GroupOpenPlace
1443		button	None (Backstage View)	TabRecent	GroupOpenPlace
1444	LoadingServicesGroup	group	None (Backstage View)	TabRecent	GroupOpenPlace
1445	GroupAddLocation	group	None (Backstage View)	TabRecent	GroupOpenPlace
1446	FileSave	button	None (Backstage View)		
1447	TabSave	tab	None (Backstage View)		
1448	SaveGroup	taskFormGroup	None (Backstage View)	TabSave	
1449	Save	category	None (Backstage View)	TabSave	SaveGroup
1450	PublishToAccessServices	task	None (Backstage View)	TabSave	SaveGroup
1451	PublishToAccessServices	group	None (Backstage View)	TabSave	SaveGroup
1452	GroupPrepareDatabaseForWeb	group	None (Backstage View)	TabSave	SaveGroup
1453	GroupPublishToAccessServices	group	None (Backstage View)	TabSave	SaveGroup
1454	FileTypes	category	None (Backstage View)	TabSave	SaveGroup
1455	SaveDatabaseAs	task	None (Backstage View)	TabSave	SaveGroup
1456	GroupSaveDatabase	group	None (Backstage View)	TabSave	SaveGroup
1457	SaveObjectAs	task	None (Backstage View)	TabSave	SaveGroup
1458	GroupSaveObject	group	None (Backstage View)	TabSave	SaveGroup
1459	TabPrint	tab	None (Backstage View)		

Datensatz: 1446 von 1477 | Kein Filter | Suchen

Bild 3: Daten von Ribbon-Elementen in einer Tabelle

- **ApplicationOptionsDialog**: **Optionen**

Die Zuordnung kann man vereinfachen, indem man mit einem kleinen Trick arbeitet und die Excel-Tabelle mit der Liste der Ribbon-Steuerelemente nutzt. Diese haben wir aus der Excel-Datei als neue Tabelle namens **tblAccessControls** zur Beispieldatenbank hinzugefügt. Sie sieht nach dem Import wie in Bild 3 aus.

Was machen wir nun mit den Daten dieser Tabelle? Wir können anhand des Steuerelementtyps und des TabSets erkennen, welche Elemente infrage kommen.

Um zu identifizieren, welches Element der Tabelle welchem Element im Ribbon entspricht, wenden wir den folgenden Trick an:

Wir definieren die eingebauten Elemente so um, dass sie statt der eingebauten Bezeichnung die **idMso** anzeigen. Das wollen wir nicht von Hand erledigen, sondern wir verwenden eine Prozedur dazu. Diese finden Sie in Listing 2.

Die Prozedur erstellt ein Recordset auf Basis der Tabelle **tblAccessControls** und ruft alle Datensätze ab, deren Feld **TabSet** den Wert **None (Backstage View)** enthält, deren Feld **Tab** den Wert **Null** hat und wo **ControlType** einen der Werte **button** oder **tab** enthält.

Diese Datensatzgruppe durchlaufen wir in einer **Do While**-Schleife und stellen innerhalb dieser Schleife Anweisungen wie diese zusammen:

```
<tab idMso=""TabInfo"" label=""TabInfo"" />
```

```
Public Sub CreateXMLBackstage()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strRibbon As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblAccessControls WHERE TabSet = 'None (Backstage View)' " _
        & "AND Tab IS NULL AND ControlType IN ('button', 'tab')", dbOpenDynaset)
    strRibbon = strRibbon & "<?xml version=""1.0""?>" & vbCrLf
    strRibbon = strRibbon & "<customUI xmlns=""http://schemas.microsoft.com/office/2009/07/customui"">" & vbCrLf
    strRibbon = strRibbon & "  <backstage>" & vbCrLf
    Do While Not rst.EOF
        strRibbon = strRibbon & "    <" & rst!ControlType & " idMso=""" & rst!ControlName & """ label=""" _
            & rst!ControlName & """ />" & vbCrLf
        rst.MoveNext
    Loop
    strRibbon = strRibbon & "  </backstage>" & vbCrLf
    strRibbon = strRibbon & "</customUI>" & vbCrLf
    DoCmd.Close acTable, "USysRibbons"
    db.Execute "UPDATE USysRibbons SET RibbonXML = '" & strRibbon & "' WHERE RibbonName = 'ClearBackstage'", _
        dbFailOnError
    If db.RecordsAffected = 0 Then
        db.Execute "INSERT INTO USysRibbons(RibbonName, RibbonXML) VALUES('ClearBackstage', '" & strRibbon & "')", _
            dbFailOnError
    End If
End Sub
```

Listing 2: Prozedur zum Zusammenstellen von neuen Bezeichnungen

Sperrung durch Memofeld statt anderer Sitzung

Neulich bat mich ein Kunde, mir einmal ein Netzwerkproblem bei ihm anzusehen. Aufgrund der Fehlermeldung konnte es keine andere Ursache geben – sie lautete »Aktualisieren nicht möglich, momentane Sperrung durch eine andere Sitzung auf diesem Rechner«. Also haben wir uns diese Sache gemeinsam angesehen und zunächst schien die Meldung plausibel – immerhin arbeitete eine andere Mitarbeiterin gerade mit der Datenbank und es konnte gut sein, dass der Datensatz gesperrt war. Allerdings trat das Problem später immer noch auf, obwohl niemand sonst mehr mit der Datenbank arbeitete. Gemeinsam sind wir dem Problem dann auf die Schliche gekommen. Die Überschrift deutet es bereits an: Ein Memofeld spielte eine große Rolle bei der Lösung des Problems.

Wer mag der Fehlermeldung aus Bild 1 nicht glauben, vor allem, wenn die Daten der betroffenen Tabelle tatsächlich von anderen Nutzern verwendet werden?

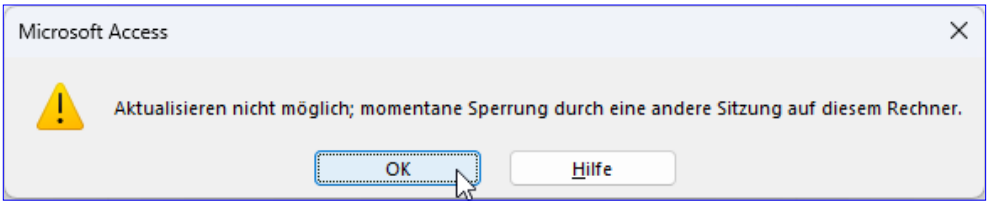

Bild 1: Fehlermeldung

Stutzig macht dann bei genauerem Hinsehen der Hinweis, dass es sich um eine andere Sitzung auf dem gleichen Rechner handelt. Und da waren wir uns doch recht sicher, dass aktuell nur eine Instanz der Datenbank geöffnet war.

Was als konnte dieses Problem verursachen? Um es etwas genauer zu beschreiben: Es trat immer auf, wenn wir versucht haben, einen der Datensätze der Tabelle in einem Detailformular zu speichern – also in einem Formular, das von einem Übersichtsformular geöffnet wurde, das wiederum die Daten zur Auswahl in einem Unterformular anzeigt.

Hier konnten wir zumindest erkennen, dass die Daten gleichzeitig in zwei verschiedenen Formularen angezeigt wurden. Aber das ist gang und gäbe – ich selbst habe hunderte Datenbanken programmiert, bei denen ein Übersichtsformular und ein

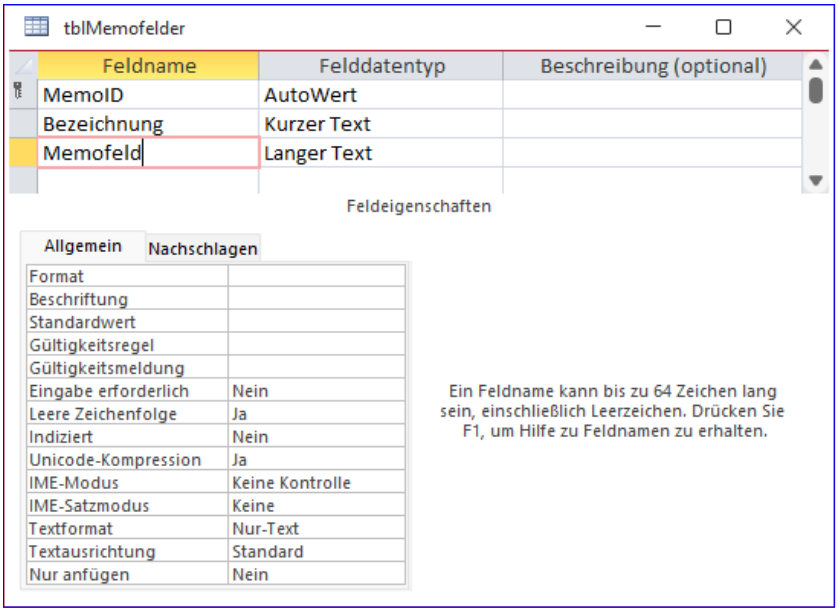

Bild 2: Tabelle zum Reproduzieren des Problems

Detailformular den gleichen Datensatz anzeigten und nie gab es Probleme beim Bearbeiten dieser Daten.

Und es wurde immer ominöser: Wir stellten fest, dass das Problem nicht immer auftrat, sondern nur dann, wenn das Memofeld in dem betroffenen Datensatz bereits eine Menge Daten enthielt.

Konstellation des Problems

Schauen wir uns an, wie das Problem genau aussieht. Wir haben dieses in einer minimalen Version nachgebaut, um zu experimentieren und eine Lösung zu finden.

Dazu haben wir zunächst eine Tabelle namens **tblMemofelder** erstellt, die neben dem Primärschlüsselfeld **MemoID** und einem Feld mit der Bezeichnung des Datensatzes namens **Bezeichnung** noch ein Memofeld namens **Memofeld** enthält. Die Tabelle sieht in der Entwurfsansicht wie in Bild 2 aus.

Dazu haben wir ein Formular mit einem Unterformular hinzugefügt. Das Hauptformular heißt **frmMemofeldUebersicht** und enthält neben einer Schaltfläche namens **cmdDetails** noch ein Unterformular namens **sfmMemofeldUebersicht**. Dieses Unterformular ist über die Eigenschaft **Datensatzquelle** an die Tabelle **tblMemofelder** gebunden und zeigt alle drei Felder dieser Tabelle an (siehe Bild 3).

Wechseln wir zur Formularansicht, sehen wir hier die Daten der Tabelle **tblMemofelder** (siehe Bild 4).

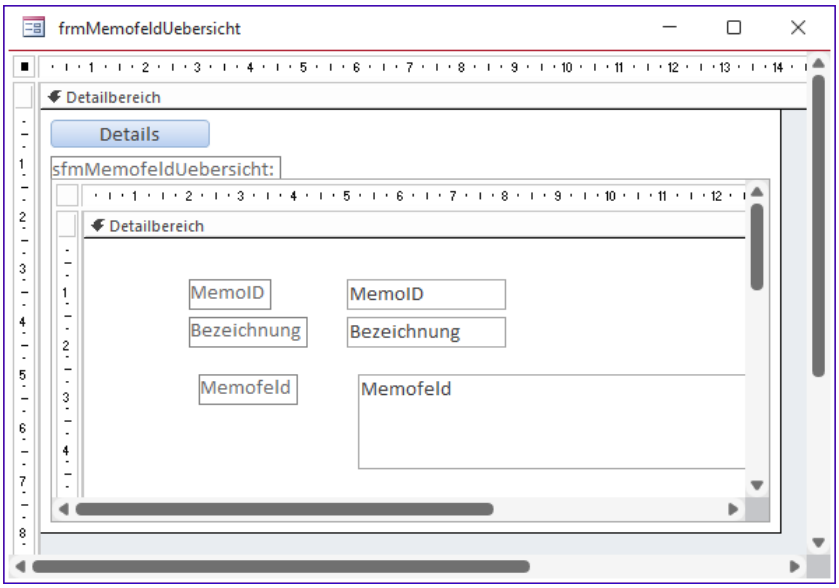

Bild 3: Formular mit einem Unterformular zur Anzeige aller Datensätze der Tabelle **tblMemofelder**

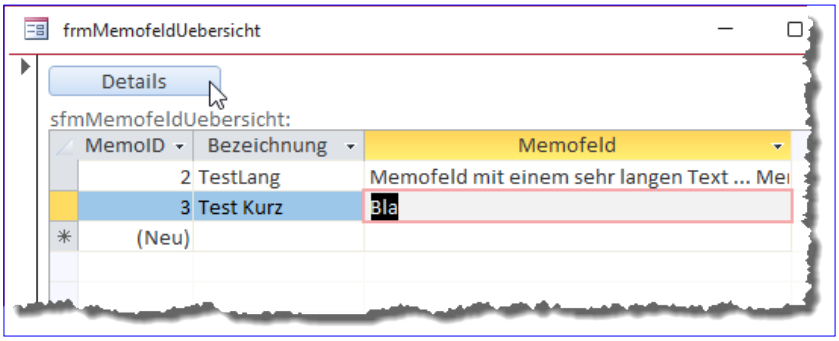

Bild 4: Formular mit den Daten der Tabelle **tblMemofelder** in der Formularansicht

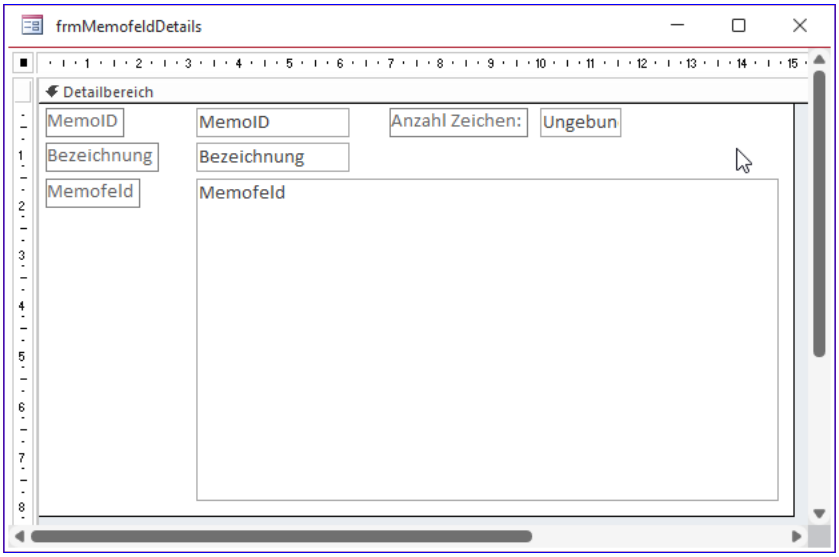

Bild 5: Formular zur Anzeige eines Datensatzes der Tabelle **tblMemofelder**

Detailformular für die Tabelle tblMemofelder

Um das Szenario zu vervollständigen, haben wir noch ein Formular erstellt, das direkt an die Tabelle **tblMemofelder** gebunden ist und alle drei Felder dieser Tabelle anzeigt (siehe Bild 5).

Hier haben wir noch ein Textfeld namens **txtMenge** hinzugefügt, mit dem wir die Menge der Zeichen des im Memofeld enthaltenen Textes ausgeben können.

Diese ermitteln wir zu zwei Gelegenheiten:

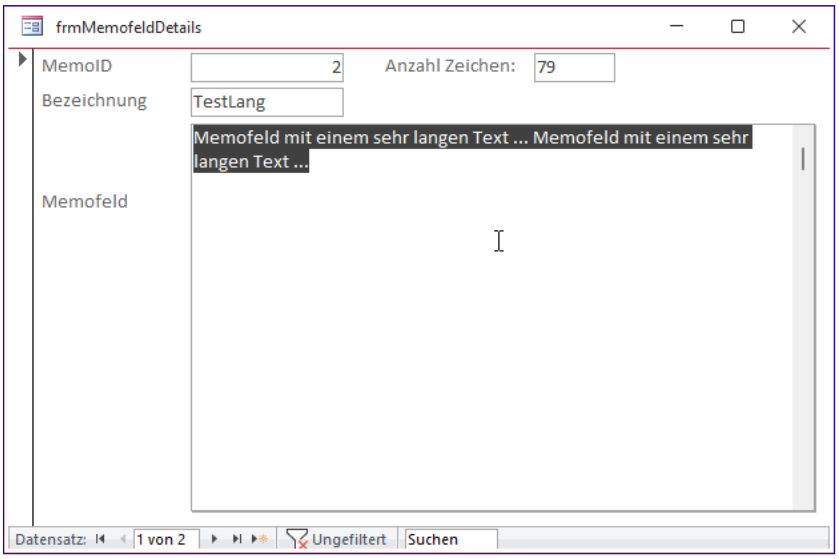

Bild 6: Anzeige von Memofeldinhalten und ihrer Länge

- Beim Anzeigen eines neuen Datensatzes, also beim Datensatzwechsel und
- beim Ändern des enthaltenen Textes.

Dazu haben wir die folgenden beiden Prozeduren für die entsprechenden Ereignisse hinterlegt:

```
Private Sub Form_Current()
    Me!txtMenge = Len(Me!Memofeld)
End Sub

Private Sub Memofeld_Change()
    Me!txtMenge = Len(Me!Memofeld.Text)
End Sub
```

Damit sorgen wir dafür, dass wir nicht nur über das Textfeld den Inhalt des Memofeldes sehen, sondern auch noch die aktuelle Länge (siehe Bild 6). Diese spielt, wie wir gleich sehen werden, eine wichtige Rolle bei unserem Problem.

Maximale Anzahl Zeichen ermitteln

Bevor wir uns um die Probleme kümmern, die durch die oben erwähnte Sperrung ausgelöst werden, schauen wir uns zuerst einmal an, wie viele Zeichen wir einem Memofeld über die Benutzeroberfläche zuweisen können. Dazu fügen wir dem Formular eine Schaltfläche hinzu, die so lange den Inhalt des Memofeldes vergrößert und speichert, bis es zu einem Fehler kommt. Die Schaltfläche heißt **cmdMemofeldAusreizen** und löst die folgende Prozedur aus:

```
Private Sub cmdMemofeldAusreizen_Click()
    On Error Resume Next
    Do While Err.Number = 0
        Me!Memofeld = Me.Memofeld _
            & " Memofeld mit einem sehr langen Text ..."
        RunCommand acCmdSaveRecord
        Me!txtMenge = Len(Me!Memofeld)
        DoEvents
    Loop
    On Error GoTo 0
End Sub
```

Diese fügt solange die Zeichenfolge **Memofeld mit einem sehr langen Text ...** zum Memofeld hinzu, bis ein Fehler ausgelöst wird. Nach jedem Hinzufügen ruft die Prozedur die Anweisung zum Speichern des aktuell im Formular angezeigten Datensatzes auf. Wird ein Fehler ausgelöst,

endet die **Do While**-Schleife, die solange läuft, bis **Err.Number** nicht mehr den Wert **0** enthält.

Nach den ersten paar Durchläufen haben wir allerdings gesehen, dass wir bei der Länge der hinzuzufügenden Zeichenkette ewig brauchen werden, um das Memofeld zu füllen und haben die Zeichenkette um ein Vielfaches verlängert. Bei etwas über 7.000.000 Zeichen erschien dann allerdings nicht etwa eine Fehlermeldung, sondern Access konnte die Anforderung nicht mehr verarbeiten und fror ein. Das geschieht allerdings auch, wenn wir in diesem Zustand versuchen, manuell weitere Zeichen einzugeben. Das Schließen der Anwendung gelang anschließend nur über das Beenden des Tasks im Taskmanager.

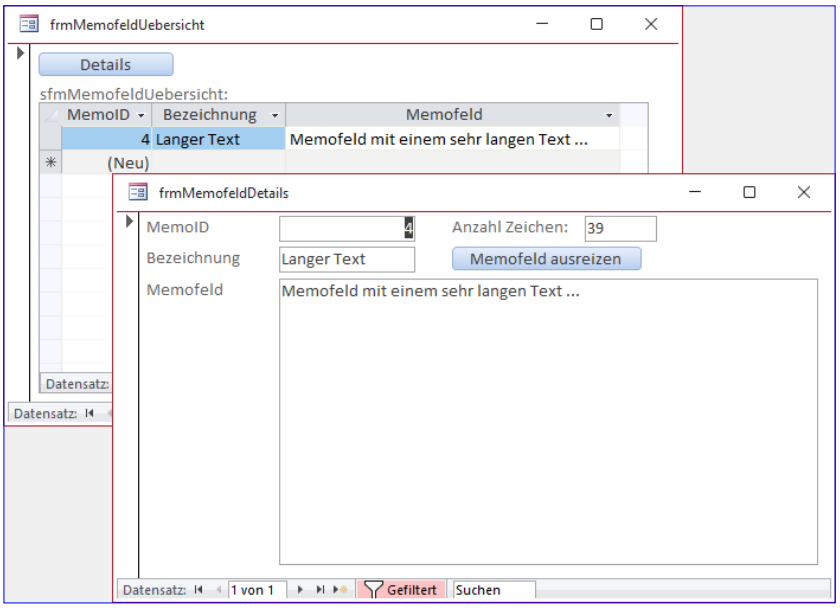

Bild 7: Anzeige des Inhalts eines Memofeldes in zwei verschiedenen Formularen gleichzeitig

Diese Menge an Zeichen erreichen wir allerdings längst nicht, wenn wir uns das oben angerissene Problem ansehen.

Memofeld in zwei Formularen gleichzeitig geöffnet

Nun nutzen wir die oben erstellten Formulare zum Reproduzieren unseres Problems. Dabei öffnen wir zuerst das Formular **frmMemofeldUebersicht**, markieren den zu untersuchenden Eintrag und öffnen dann mit der Schaltfläche **Details** das Detailformular **frmMemofeldDetails** (siehe Bild 7). Der zu untersuchende Eintrag enthält aktuell nur wenige Zeichen, in diesem Fall **39**.

Dort fügen wir nun einige weitere Zeichen ein, sodass das Memofeld beispielsweise 3.000 Zeichen anzeigt. Auch das ist kein Problem! Also was führt nun zu dem eingangs erwähnten Fehler?

Schließen wir einfach nochmal das Detailformular und aktualisieren den Inhalt des Übersichtsformulars mit **F5**. Dann öffnen wir den Detaildatensatz erneut und versuchen, diesen zu bearbeiten. Und dann tritt der Fehler auf – siehe Bild 8.

Was hat sich nun geändert? Im Gegensatz zum vorherigen Beispiel versuchen wir nun, einen bereits recht langen Inhalt eines Memofeldes zu ändern. Zuvor war die gespei-

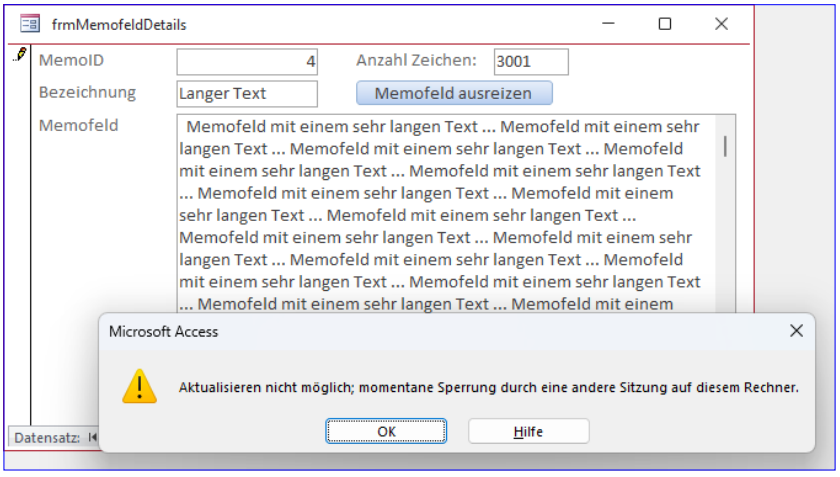

Bild 8: Fehler beim Ändern des Inhalts des Memofeldes

Mit Zeiten rechnen

Zeit- und Datumsangaben sind elementarer Bestandteil vieler Anwendungen. Da sollte man wissen, wie man mit diesen Daten rechnet. Dabei ist es einerlei, ob es um das Ermitteln einer Zeitspanne zwischen zwei Zeitangaben oder um das Summieren von Zeiten geht – Stolperfallen gibt es dabei eine ganze Reihe. Dieser Beitrag zeigt, wie Sie die Klippen beim Rechnen mit Zeiten umgehen, egal ob in VBA, Abfragen, Formularen oder Berichten.

Wie Access Zeiten speichert

Grundlegende Voraussetzung für die Arbeit mit Datums- und Zeitwerten ist die Kenntnis der Art der internen Speicherung dieser Daten unter Access. Es speichert Zeiten im **Double**-Format, wobei die Stellen vor dem Komma die Anzahl der Tage seit dem **30.12.1899** angeben (**1** bedeutet **31.12.1899**, **44998** steht für den **12.3.2023**) und die Stellen hinter dem Komma die Uhrzeit, wobei **0** genau Mitternacht entspricht und **0,5** zwölf Uhr mittags. Entgegen den üblichen Vorstellungen kann Access auch mit Zeiten vor dem 30.12.1899 arbeiten. Dazu verwenden Sie einfach ganze Zahlen mit negativem Vorzeichen. Experimente zeigten, dass beim 1.1.100 aber definitiv Schluss ist und der Geburtstag von Jesus Christus zumindest nicht als Jahreszahl in Ihre Datenbank Einzug halten wird. Grundlegende Informationen zu diesem Thema erhalten Sie im Beitrag **Datum und Zeit mit Access** (**www.access-im-unternehmen.de/1432**). Dort erläutern wir auch die von VBA bereitgestellten Funktionen etwa zur formatierten Ausgabe von Zeitangaben oder zum Rechnen mit den Funktionen **DateAdd** oder **DateDiff**.

Zeitspannen berechnen

Die **DateDiff**-Funktion berechnet die Zeitspanne zwischen zwei Daten, wobei die Zeitspanne in einer beliebigen Zeiteinheit wie beispielsweise Sekunden, Stunden oder auch Tagen angegeben werden kann. Ein separates Pendant für die Berechnung von Differenzen zwischen zwei Uhrzeiten findet sich unter VBA nicht. Unter VBA berechnen Sie Zeitspannen etwa ganz einfach als Differenz zweier Literale. Im Direktfenster würde eine Beispieleingabe folgendes Ergebnis liefern:

```
? #10:00:00# - #8:00:00#
 8,33333333333334E-02
```

Das Ergebnis ist eine in wissenschaftlicher Schreibweise dargestellte Zahl, die in ein Datum umgewandelt so aussieht:

```
? CDate(8.33333333284827E-02 )
02:00:00
```

Beachten Sie, dass Access die Differenz im Direktfenster mit einem Komma als Dezimaltrennzeichen ausgibt, bei Berechnungen jedoch einen Punkt als solches erwartet. Schwieriger wird es bei Zeitspannen, deren Anfangs- und Endzeit nicht das gleiche Datum haben. Die Berechnung einer typischen Nachtschicht von 22 Uhr bis 6 Uhr am folgenden Tag führt so zu einer negativen Zahl:

```
? #6:00:00# - #22:00:00#
-0,666666666666667
```

Das ist auch logisch, wenn man sich ansieht, wie die **Double**-Zahlen aussehen, mit denen Access intern rechnet:

```
? CDbl(#6:00:00#)
 0,25
? CDbl(#22:00:00#)
 0,916666666666667
```

Was also tun, wenn Sie Zeitspannen zweier Zeiten ermitteln möchten, die nicht an einem Tag liegen? Das Problem lässt sich leicht lösen, wenn es um Zeiten geht, die auf

den gleichen Tag fallen oder maximal einen Tag auseinanderliegen – also beispielsweise Arbeitszeiten. Obiges Beispiel würde man mit folgender **IIf**-Bedingung abrunden:

```
? IIf(#6:00:00# - #22:00:00# > 0, #6:00:00# - #22:00:00#,
#6:00:00# - #22:00:00# + 1)
 0,333333333333333
```

Diese prüft, ob die Differenz negativ ist, und addiert in dem Fall den Wert **1** hinzu. Dies entspricht genau einem Tag – Sie erinnern sich, die Stellen vor dem Komma sind mit der Anzahl der Tage seit dem 30.12.1899 gleichzusetzen. Einfacher und zuverlässiger, auch für größere Zeitspannen, funktioniert dies, wenn Sie das Datum mit angeben:

```
? #1/3/2008 6:00:00# - #1/2/2008 22:00:00#
 0,333333333335759
```

Aber halt: Das Ergebnis stimmt in den hinteren Nachkommastellen nicht mit dem tatsächlichen Ergebnis überein – was ist dort schiefgelaufen? Der Grund liegt ganz einfach in Ungenauigkeiten beim Rechnen mit Gleitkommazahlen. Für genaue Ergebnisse gibt es zwei Möglichkeiten: Entweder man rechnet getrennt nach Datum und Zeit oder man verwendet doch die **DateDiff**-Funktion.

Die getrennte Rechnung sieht so aus:

```
? #1/3/2008# - #1/2/2008# + #6:00:00# - #22:00:00#
 0,333333333333333
```

Mit **DateDiff** geht es folgendermaßen:

```
Function ZeitdifferenzHMS(dat1 As Date, dat2 As Date) As
String
    Dim s As Long
    Dim h As Long
    Dim n As Long
    s = DateDiff("s", dat1, dat2)
    h = Int(s / 3600)
    n = Int((s Mod 3600) / 60)
    s = Int(n Mod 60)
    ZeitdifferenzHMS = Format(h, "00") & ":" _
        & Format(n, "00") & ":" & Format(s, "00")
End Function
```

Dabei wird **DateDiff** in eine weitere Funktion gepackt. Der Hintergrund ist, dass **DateDiff** die Zeitdifferenz als Ergebnis nur in Form einer einzigen Zahl zurückgibt, wobei diese Zahl mit der im Funktionsaufruf angegebenen Einheit zu verknüpfen ist – hier also beispielsweise Sekunden. Die Funktion **ZeitdifferenzHMS** liefert diese Differenz im Format **hh:nn:ss**, wobei auch Differenzen größer als 24 Stunden möglich sind. Dafür kommt der Wert auch als **String** und nicht als **Date**. Der Versuch, das Resultat mit der **CDate**-Funktion in einen Datumswert umzuwandeln, schlägt fehl, weil **CDate** nicht mit Stundenwerten größer als 23 umgehen kann.

Zeitdifferenzen in Abfragen

Beim Ermitteln von Zeitdifferenzen in Abfragen ist es am einfachsten, wenn Datum und Zeit im gleichen Feld gespeichert sind. In einem weiteren Feld ermitteln Sie dann mit einem Ausdruck wie **Dauer: [Endzeit]-[Startzeit]** die Zeitdifferenz (siehe Bild 1). In der Formularansicht zeigt Access die Differenz dann als Dezimalzahl an (siehe Bild 2). Verpasst man diesem Ausdruck noch die oben vorgestellte Funktion **ZeitdifferenzHMS**, wird ein Schuh draus: Die Anzahl der Stunden wird in der folgenden Spalte im richtigen Format angezeigt, auch für Zeitspannen, die sich über Mitternacht oder mehr als 24 Stunden erstrecken:

```
DauerHMW: ZeitdifferenzHMS([Startzeit];[Endzeit])
```

Zeiten addieren

Wenn Sie die Arbeitszeiten für einzelne Tage ermittelt haben, möchten Sie diese vielleicht auch summieren, etwa um die Wochen- oder Monatsarbeitszeit zu ermitteln. Das ist prinzipiell einfach:

```
? #8:00:00# + #8:00:00#
16:00:00
```

Rechnungsbericht mit EPC-QR-Code

Nachdem wir in einigen vorherigen Beiträgen eine feine Verwaltung für Kunden, Produkte und Bestellungen programmiert haben, wagen wir uns nun an ein Thema heran, das noch recht neu ist und das auch erst durch die wachsende Anzahl von Anwendungen im Finanzbereich für mobile Geräte entstanden ist. Wir wollen Rechnungen nun nicht mehr nur auf einem schnöden Blatt Papier oder als PDF-Dokument erzeugen, sondern diese auch noch mit einem schicken QR-Code versehen, der die für die Begleichung der Rechnung über die Banking-App notwendige Bilddatei anzeigt. Und wenn wir schon dabei sind, erklären wir auch direkt, wie Sie auf Basis der bisherigen Teile dieser Beitragsreihe einen Rechnungsbericht erstellen.

Hinweis

Die Downloads zu diesem Beitrag enthalten die DLL zur Generierung des hier verwendeten EPC-QR-Codes. Diese DLL kann hier erworben werden:

https://shop.minhorst.com/access-tools/372/amvepcqrcode?c=78

Als Abonnent dieses Magazins erhalten Sie eine Kopie der DLL, die Sie für den privaten Gebrauch nutzen können. Für geschäftliche Zwecke oder für den Einbau in Anwendungen für Ihre Kunden können Sie eine oder mehrere Lizenzen erwerben.

Vorbereitungen

Bevor wir uns mit der Erstellung des eigentlichen Berichts beschäftigen, erledigen wir ein paar vorbereitende Arbeiten:

- Hinzufügen einer Schaltfläche zum Formular **frmBestellungDetails** zum Öffnen der Rechnung
- Erstellen einer Optionentabelle mit den Daten des Rechnungsversenders
- Erstellen eines Formulars zum Verwalten der Optionen dieses Formulars

Danach kümmern wir uns um den Bericht selbst.

Hinzufügen einer Schaltfläche zum Öffnen des Rechnungsberichts

Zum Erstellen von Rechnungen eignen sich verschiedene Orte in einer Datenbank. Wir könnten in der Übersicht aller Bestellungen eine Schaltfläche hinzufügen, die alle

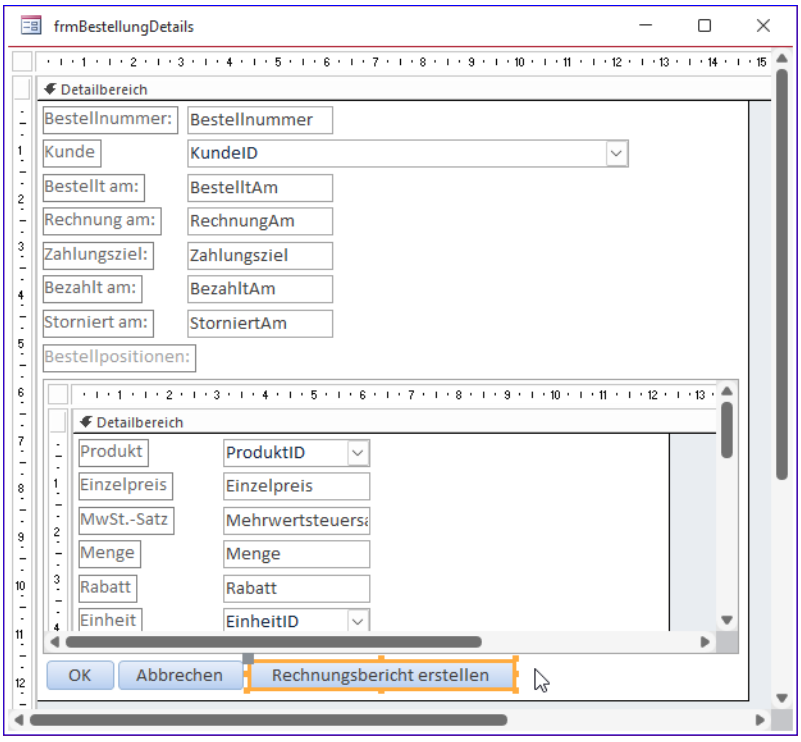

Bild 1: Schaltfläche zum Anzeigen des Rechnungsberichts

aktuell angezeigten Bestellungen in Rechnung stellt, aber wir wollen es erst einmal einfacher halten und nur eine Rechnung gleichzeitig anzeigen. Dazu verwenden wir das Formular **frmBestellungDetails**, das jeweils eine Bestellung anzeigt. Diesem fügen wir eine Schaltfläche namens **cmdRechnungsberichtErstellen** hinzu (siehe Bild 1).

Für diese Schaltfläche hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub cmdRechnungsberichtErstellen_Click()
    DoCmd.OpenReport "rptRechnung", _
        View:=acViewPreview, _
        WhereCondition:="ID = " & Me!ID
End Sub
```

Dies öffnet den noch zu erstellenden Bericht namens **rptRechnung** in der Seitenansicht – das ist die Ansicht, welche die Druckansicht anzeigt. Außerdem übergeben wir dem Bericht die **ID** der aktuell im Formular angezeigten Bestellung.

Tabelle für die Optionen des Rechnungssenders erstellen

Unter Optionen des Rechnungssenders verstehen wir solche Daten, die im Bericht landen sollen, aber bisher nirgends gespeichert werden. Dazu gehören Informationen wie Firmenname, Briefkopf, Bankverbindung, Umsatzsteuer-Identifikationsnummer et cetera.

Diese könnten wir auch direkt im Bericht anlegen, aber wenn der Benutzer diese dann ändern möchte, müsste er den Entwurf des Berichts anpassen – und das wollen wir vermeiden. Also wollen wir solche Informationen in einer Optionentabelle speichern, deren Daten in einem entsprechenden Formular angezeigt werden.

Der Entwurf dieser Tabelle wird in Bild 2 dargestellt.

Formular zum Verwalten der Optionen

Das Formular, mit dem wir die Optionen für den Bericht eingeben wollen, finden Sie in Bild 3. Es verwendet die Tabelle **tblOptionen** als Datensatzquelle. Wir ziehen alle Felder außer dem Feld **ID** in den Detailbereich des Formularentwurfs.

Dadurch, dass wir in den vorherigen Teilen der Beitragsreihe bereits ein Formular namens **Normal** erstellt haben, in dem wir auch einige Standardeinstellungen für Beschriftungsfelder vorgenommen haben, werden die Beschriftungen direkt mit Doppelpunkten abgeschlossen.

Da das Formular immer nur einen Datensatz anzeigen und keinen Wechsel zu einem neuen, leeren Datensatz ermöglichen soll, stellen wir die Eigenschaft **Zyklus** auf **Aktuel-**

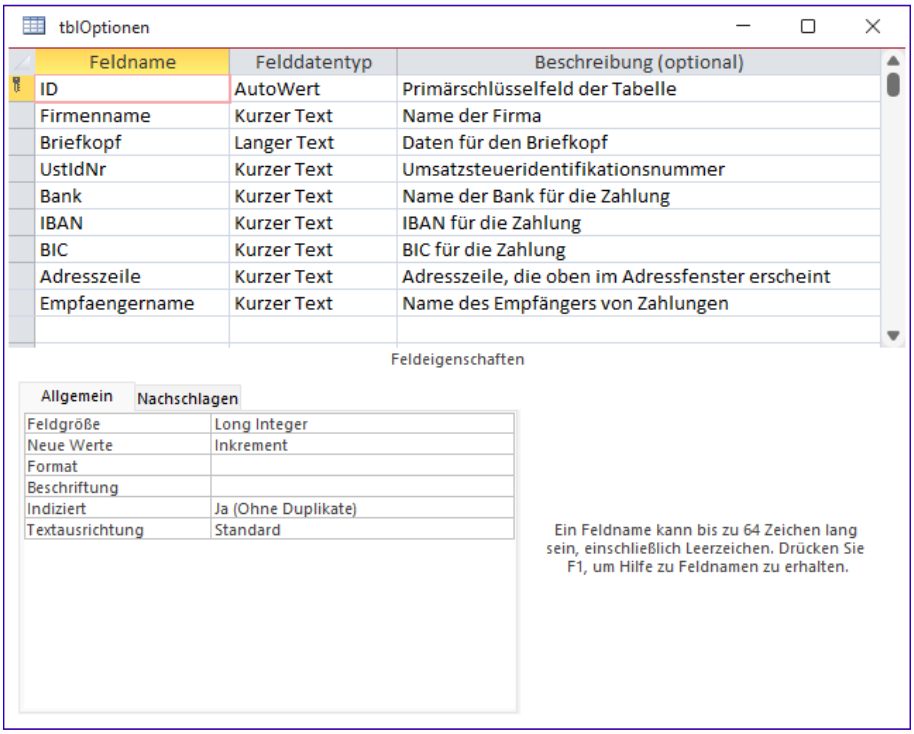

Bild 2: Tabelle zum Speichern der Optionen für den Bericht

ler Datensatz ein. Außerdem stellen wir die Eigenschaften **Bildlaufleisten**, **Trennlinien**, **Navigationsschaltflächen** und **Datensatzmarkierer** auf den Wert **Nein** und **Automatisch zentrieren** auf den Wert **Ja** ein.

Schließlich legen wir den Namen der Schaltfläche mit der Beschriftung **OK** auf **cmdOK** fest und hinterlegen für diese die folgende Ereignisprozedur:

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

Außerdem legen wir für die Textfelder **Firmenname**, **Briefkopf**, **Empfaengername** und **Adresszeile** die Eigenschaft **Horizontaler Anker** auf **Beide** ein, damit die Größe dieser Steuerelemente mit dem Vergrößern des Formulars ebenfalls vergrößert wird. Diese Eigenschaft müssen wir für die dazugehörigen Bezeichnungsfelder anschließend wieder auf Links zurücksetzen.

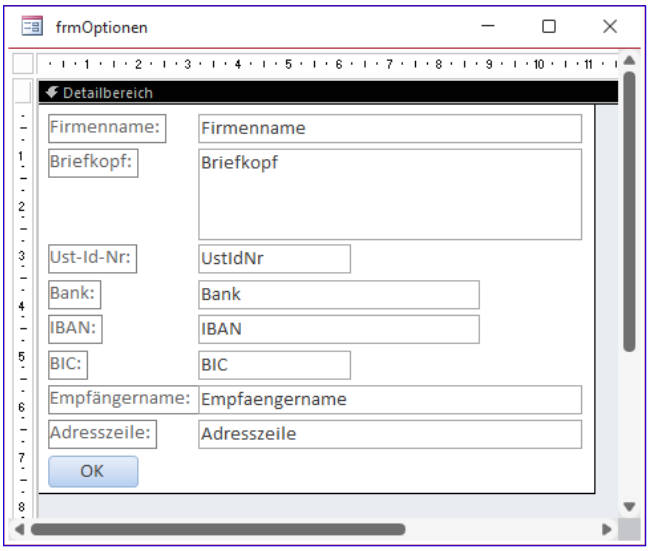

Bild 3: Formular zum Eingeben der Optionen in der Entwurfsansicht

Wechseln wir zur Formularansicht und geben wir einige Daten in der Formular ein, sieht dieses wie in Bild 4 aus. Nach der Eingabe der Daten können wir es mit einem Klick auf **OK** schließen.

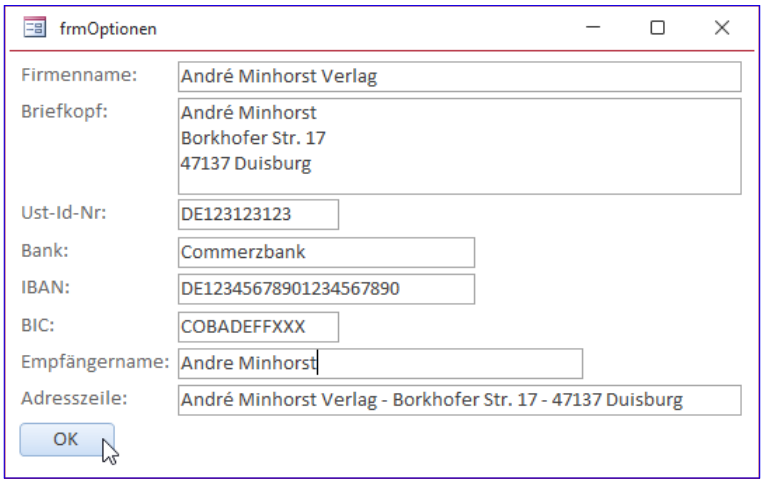

Bild 4: Formular zum Eingeben der Optionen in Aktion

Neuen Bericht erstellen

Den neuen Bericht erstellen wir mit dem Ribbonbefehl **Erstellen|Berichte|Berichtsentwurf**. Danach landen wir in der Entwurfsansicht und nehmen direkt die erste Einstellung vor.

Dabei legen wir für den Detailbereich die Eigenschaft **Alternative Hintergrundfarbe** auf den Wert der Eigenschaft **Hintergrundfarbe** ein, in diesem Fall von **Hintergrund 1, Dunkler 5%** zu **Hintergrund 1** (siehe Bild 5).

Abfrage für den Bericht

Nun fügen wir die Datensatzquelle für den Bericht hinzu. Und im Gegensatz zu Formularen, wo man immer möglichst eine Tabelle pro Haupt-/Unterformular verwendet, damit die Daten noch bearbeitet werden können, statten wir die Abfrage für den Bericht mit allen Daten aus, die wir darin anzeigen wollen.

Da wir wollen, dass die Abfrage optimiert wird, um schnell die benötigten Daten zusammenstellen zu können, speichern wir ihn als eigene Abfrage, die auch im Navigationsbereich angezeigt wird. Diese Abfrage nennen wir **qryRechnung**. Wir fügen dieser fast alle Tabellen der Datenbank hinzu (siehe Bild 6).

Aus diesen Tabellen ziehen wir die folgenden Felder in den Abfrageentwurf, beginnend mit der Tabelle **tblBestellungen**, dann nach rechts und anschließend nach links:

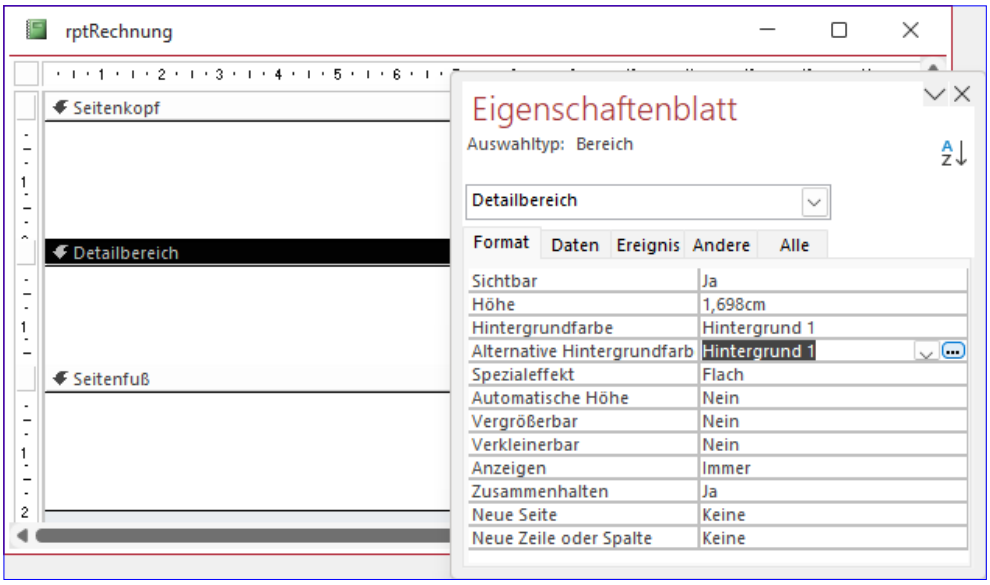

Bild 5: Einstellen der alternativen Hintergrundfarbe für den Detailbereich

- **tblBestellungen**: **ID**, **Bestellnummer**, **BestelltAm**, **RechnungAm**, **Zahlungsziel**
- **tblBestellpositionen**: **Einzelpreis**, **Mehrwertsteuersatz**, **Menge**, **Rabatt**
- **tblProdukte**: **Produktbezeichnung**
- **tblEinheiten**: **Einheitsbezeichnung**
- **tblKunden**: **Kundennummer**, **Firma**, **Vorname**, **Nachname**, **Strasse**, **PLZ**, **Ort**, **UstIDNr**
- **tblAnrede**: **AnredeBrief**

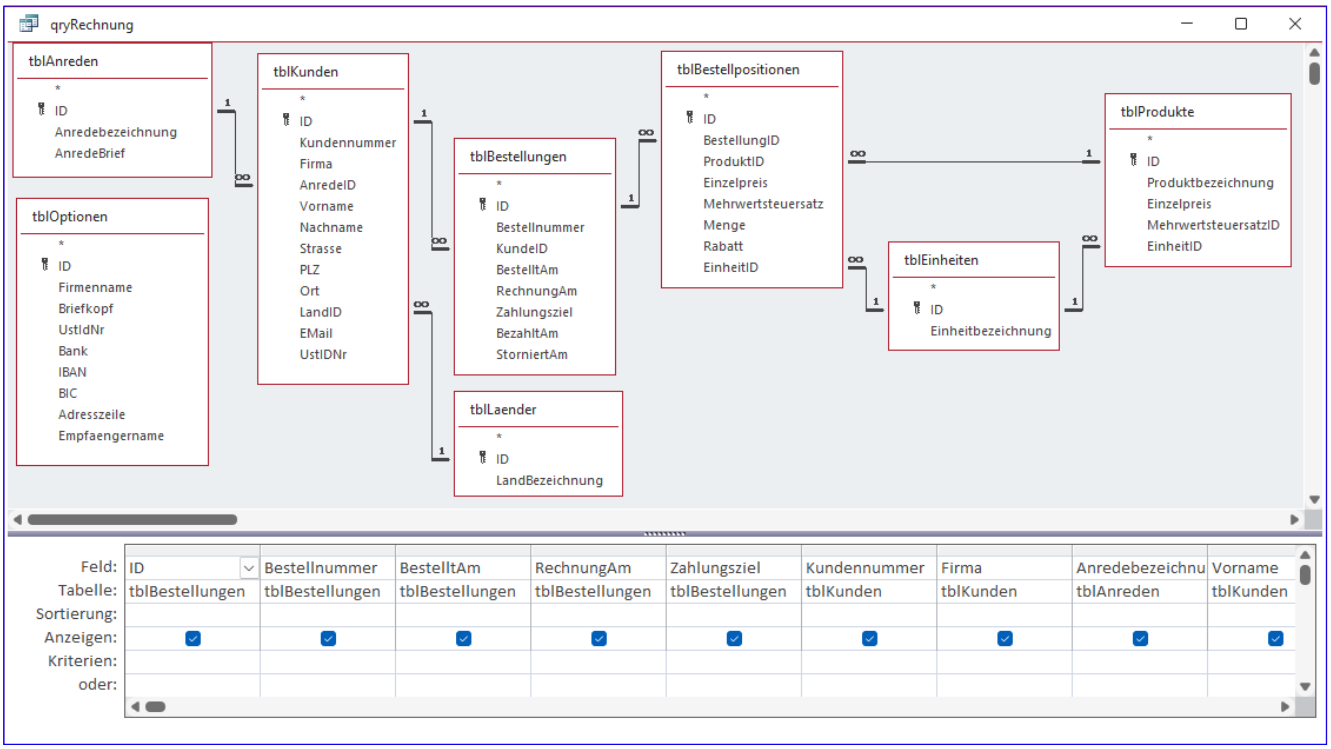

Bild 6: Abfrage als Datensatzquelle des Rechnungsberichts

- **tblLaender**: **LandBezeichnung**

- **tblOptionen**: alle außer **ID**

Wir fügen also selbst alle Felder der Tabelle **tblOptionen** hinzu, obwohl diese mit keiner der hier vorliegenden Tabellen verknüpft ist. Aber ist das ein Problem, weil ja dann alle Kombinationen der Datensätze der Tabellen **tblOptionen** und der Datensätze der übrigen Tabellen ausgegeben werden? Nein, das ist kein Problem – solange die Tabelle **tblOptionen** nur den einen vorgesehenen Datensatz enthält.

Sollte man einmal mehrere Sätze von **Optionen** verwenden wollen, um Rechnungen mit verschiedenen Erstellern zu erzeugen, dann muss man noch eine entsprechende Verknüpfung hinzufügen. Das wollen wir im Rahmen dieses Beitrags jedoch nicht tun.

Hinzufügen der Felder zum Bericht

Nun stellen wir für den Bericht die Abfrage **qryRechnung** als **Datensatzquelle** ein. Außerdem blenden wir den Bereich Berichtskopf/-fuß ein. Diesen ziehen wir in der Höhe ein wenig größer und stellen die Hintergrundfarbe auf **Automatisch** ein. Die Bereiche **Seitenkopf** und **Seitenfuß** minimieren wir zunächst in der Höhe – diese holen wir später wieder dazu.

Danach können wir beginnen, die Felder in der gewünschten Anordnung in den Bericht zu ziehen. Dabei landen zu-

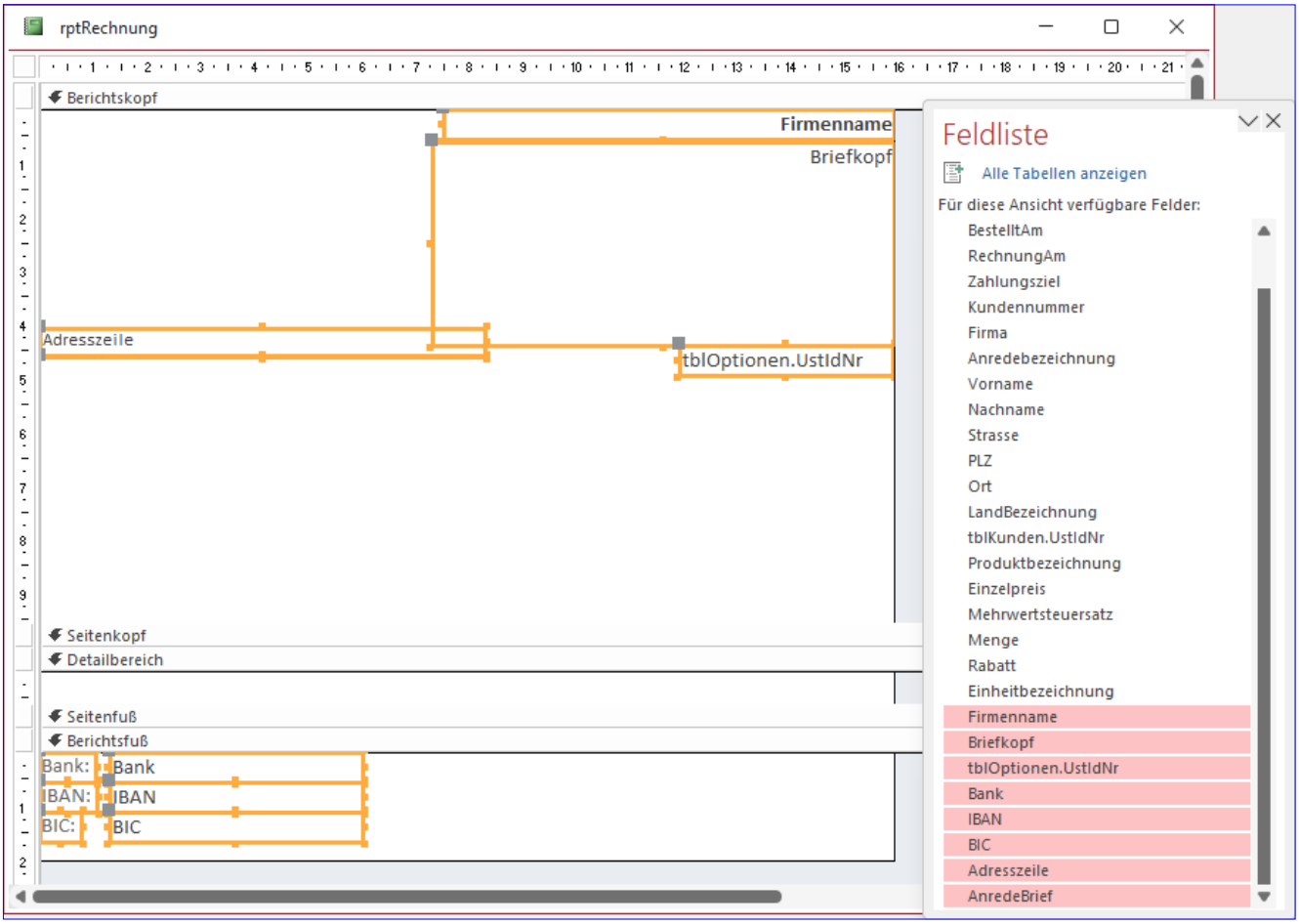

Bild 7: Hinzufügen der Elemente der Tabelle **tblOptionen**

Probleme bei Verweisen mit verschiedenen Versionen

Wenn Sie Datenbanken entwickeln, die auf Rechnern eingesetzt werden sollen, die eine ältere Version von Office verwenden, kann es zu Problemen kommen. Abgesehen davon, dass die Datenbank Funktionen enthalten könnte, die auf der Zielplattform nicht verfügbar sind, kommt es vor allem zu Problemen mit Verweisen. Ein Beispiel sind Verweise auf die Office-Anwendungen. Haben Sie Verweise zu den Bibliotheken von Word, Excel, Outlook et cetera, die sich auf die aktuelle Office-Version beziehen, und stellt der Zielrechner diese nicht bereit, kann es sein, dass die Anwendung nicht wie gewünscht funktioniert. Wir schauen uns an, wie die Probleme entstehen und wie wir diese lösen können.

Ausgangssituation

Es gibt verschiedene Konstellationen, die Probleme mit Verweisen begünstigen. Wir schauen uns in diesem Beitrag eine davon an – dabei entwickeln wir eine Access-Datenbank, welche Verweise der Version 16.0 von Office hinzufügt und geben diese dann weiter an einen Rechner, bei dem noch die Version 14.0 verwendet wird.

Entsprechend der Version 16.0 von Office landen auch die mit Office gelieferten Bibliotheken in dieser Version in der Datenbank, genauer gesagt im **Verweise**-Dialog des VBA-Projekts (siehe Bild 1). Hier haben wir Verweise auf die Bibliotheken für Excel, Office, Outlook und Word jeweils in der Version 16.0 hinzugefügt – eine andere war auch nicht verfügbar.

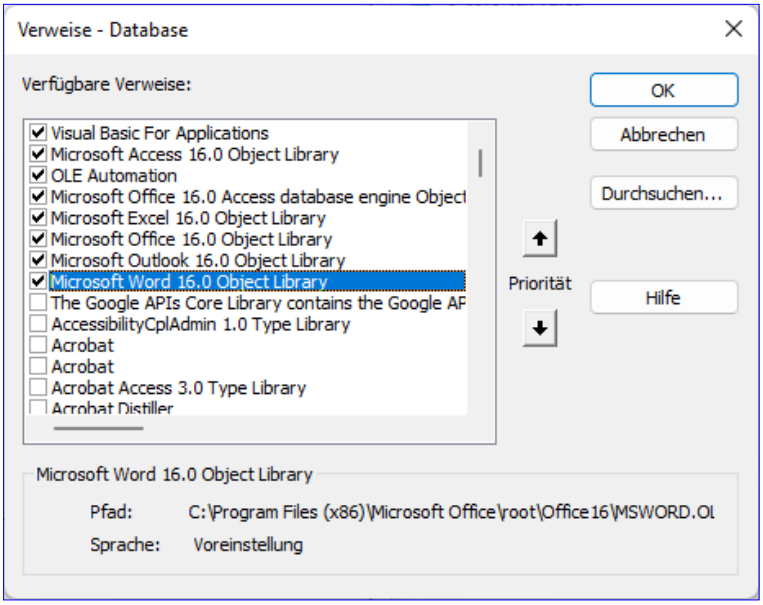

Bild 1: Ausgangssituation im **Verweise**-Dialog

Test auf einem älteren System

Danach haben wir diese Datenbankdatei auf eine virtuelle Maschine kopiert und diese dort geöffnet. Da die Datenbank bis auf die hinzugefügten Verweise leer ist, treten beim Öffnen auch keine Probleme auf – wie es sonst der Fall ist, wenn man beispielsweise direkt VBA-Code ausführt, der aufgrund fehlender oder fehlerhafter Verweise zu Fehlern führt. Wechseln wir dann jedoch in den VBA-Editor und öffnen den

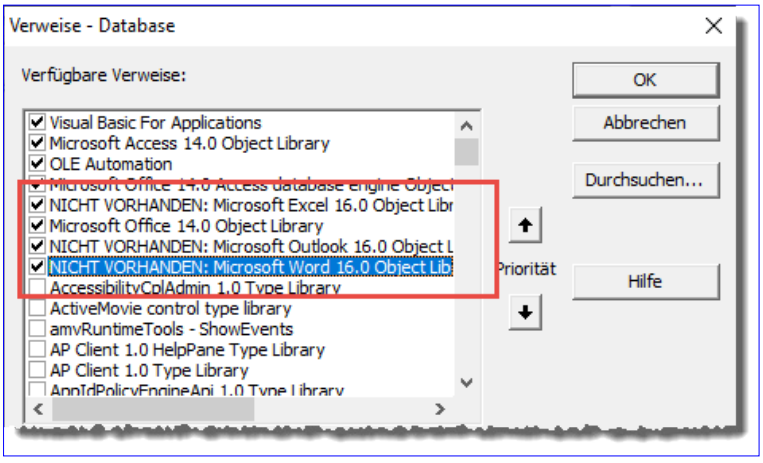

Bild 2: Einige Verweise werden als NICHT VORHANDEN gekennzeichnet.

Verweise-Dialog, finden wir die Situation aus Bild 2 vor.

Einige der hinzugefügten Verweise, namentlich die für die Excel-, Outlook- und Word-Bibliotheken, werden als **NICHT VORHANDEN** markiert. Dementsprechend können wir die Elemente dieser Verweise auch nicht unter VBA nutzen.

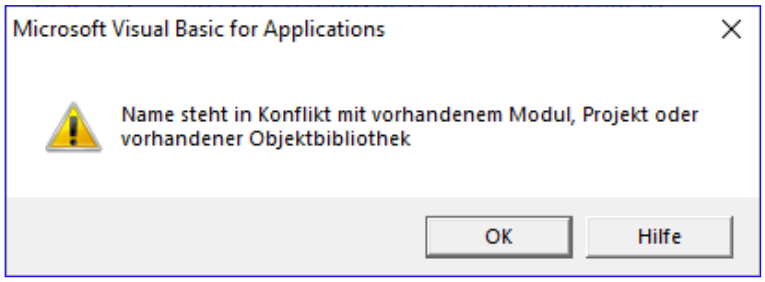

Bild 3: Fehler beim Versuch, einen Verweis durch den älteren Verweis zu ersetzen

Manuelles Beheben des Fehlers

Wenn die Datenbank in der Variante mit offenem Quellcode, also nicht als **.accde**-Datenbank, weitergegeben wurde, können Sie ihm Anweisungen zum Beheben des Problems geben.

Möglicherweise versucht er, einfach die richtigen Verweise hinzuzufügen, also beispielsweise den Verweis auf die Bibliothek **Microsoft Excel 14.0 Object Library**. Der Versuch ist möglich, allerdings mündet dieser in der Anzeige der Fehlermeldung aus Bild 3.

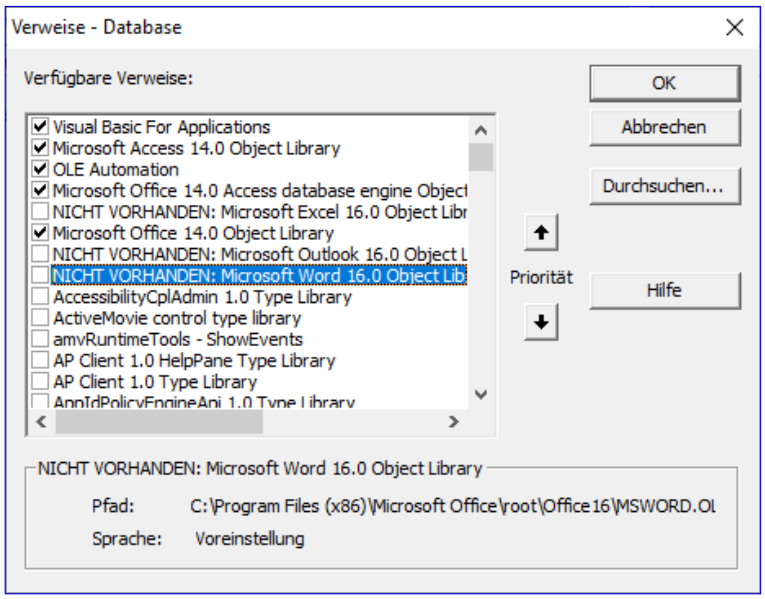

Bild 4: Entfernen der als **NICHT VORHANDEN** markierten Verweise

Das Problem entsteht dadurch, dass die beiden Bibliotheken die gleiche GUID aufweisen und wir nicht zwei Verweise mit der gleichen GUID zu den Verweisen eines Projekts hinzufügen können.

Der korrekte Weg wäre, die als **NICHT VORHANDEN** markierten Einträge zunächst zu entfernen, indem man die Haken aus den entsprechenden Kontrollkästchen entfernt (siehe Bild 4).

Anschließend fügt man einfach die gewünschten Verweise in der verfügbaren Version erneut hinzu (siehe Bild 5). Sofern die übermittelte Datenbank keine Objekte, Eigenschaften oder Methoden einer neueren Version der Bibliotheken verwendet, sollte die Anwendung nun funktionieren.

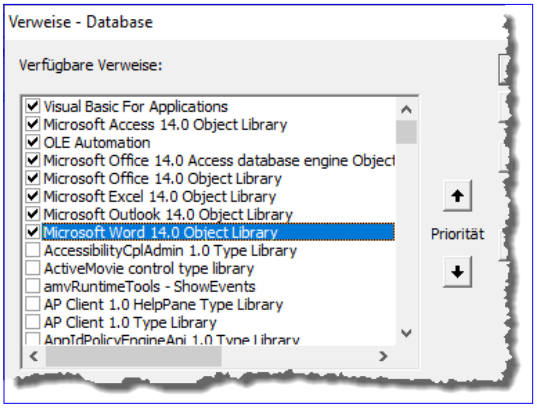

Bild 5: Hinzufügen der Verweise in der benötigten Version, hier 14.0

Automatisiertes Anpassen der Verweise

Allerding wollen wir dem Benutzer kaum zumuten, mit jeder neuen Version unserer Datenbankanwendung die Verweise neu einzustellen. Also überlegen wir uns einen Weg, wie wir dies per VBA automatisieren können. Wie nicht anders zu erwarten, bietet das Objektmodell von Access die Möglichkeit des Zugriffs auf die Verweise.

Verweise per VBA erneuern

Theoretisch sollte es möglich sein, die Verweise einer Anwendung, die als **NICHT VORHANDEN** markiert sind, per VBA zu erneuern – vorausgesetzt, diese sind auf dem Zielrechner vorhanden und installiert. Dabei gibt es allerdings Ausnahmen wie beispielsweise die in diesem Beispiel verwendeten Verweise.

Wir haben einmal versucht, die Einträge für die nicht vorhandenen Verweise auf dem Zielrechner mit Office 14 zu entfernen.

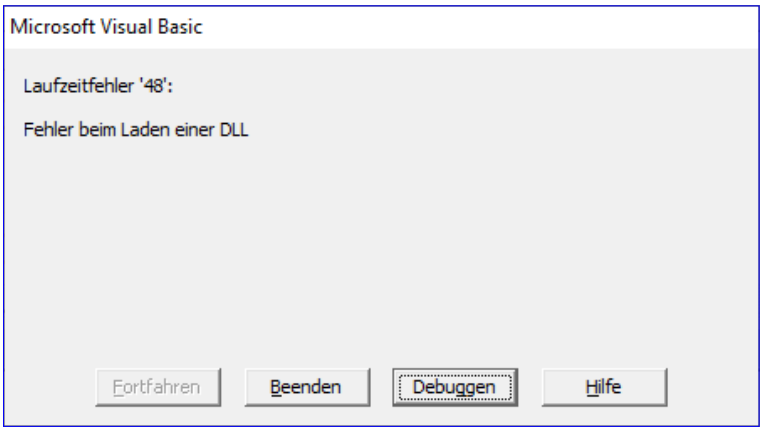

Bild 6: Fehler beim Versuch, einen nicht vorhandene Verweis per VBA zu entfernen

Dazu haben wir die folgende Prozedur verwendet. Diese durchläuft in einer **For...Next**-Schleife mit der Schrittweite **-1** alle Referenzen in einer Schleife von der Anzahl der Verweise bis zum Wert **1**. Innerhalb der Schleife referenziert die Prozedur die Referenz mit dem Index aus der Variablen **lngIndex** mit der Variablen **ref**.

```
Public Sub RemoveBrokenReferences()
    Dim ref As Reference
    Dim lngIndex As Long
    For lngIndex = References.Count To 1 Step -1
        Set ref = References(lngIndex)
        If ref.IsBroken Then
            References.Remove ref
        End If
    Next lngIndex
End Sub
```

Wenn sie auf den ersten Verweis mit dem Vermerk **NICHT VORHANDEN** stößt, löst dies den Fehler aus Bild 6 aus. Was geschieht hier? Offensichtlich kann die Bibliothek hinter diesem Verweis nicht gefunden werden, und das Problem ist, dass VBA den Verweis nicht entfernen kann, wenn die Bibliothek nicht vorhanden ist.

Ehrlich gesagt haben wir keine Methode gefunden, wie man diese Verweise dennoch entfernen kann. Daher empfehlen wir eine von zwei alternativen Methoden.

Alternative Methoden für Verweisprobleme

Es gibt beispielsweise die folgenden beiden Möglichkeiten, um die beschriebenen Probleme mit Verweisen zu umgehen:

- Gar nicht erst mit Verweisen arbeiten und Late Binding nutzen. Dabei arbeitet man nicht mit expliziten Deklarationen wie etwa **Dim objWord As Word.Application**, sondern mit **Dim objWord As Object**. Der Vorteil ist: Man benötigt keine Verweise. Der Nachteil: Die Verwendung von IntelliSense fällt weg.

- Die Verweise erst beim ersten Öffnen der Anwendung nach der Weitergabe an den Zielrechner setzen. Dabei würde man vor der Weitergabe alle relevanten Verweise entfernen und diese durch eine automatisch gestartete VBA-Prozedur beim erneuten Aufrufen der Anwendung neu setzen. Die Verweisdaten würden wir dabei in einer Tabelle speichern und sie beim erneuten Setzen aus der Tabelle auslesen.

Verweise erst nach Weitergabe setzen

Wir schauen uns die zweite Methode im Detail an. Dabei wollen wir uns nur um die Verweise kümmern, die nicht standardmäßig verwendet werden, also in einer neuen Anwendung noch nicht vorhanden sind. Als Erstes legen wir die Tabelle an, in der wir die anzulegenden Verweise speichern. Diese definieren wir wie in Bild 7.

Dateien aus dem Web herunterladen per VBA

Es gibt eine Menge Gründe, warum man per VBA komplette Dateien aus dem Internet herunterladen sollte. Beispielsweise könnte man von dort Listen im Excel- oder .csv-Format herunterladen, um anschließend die enthaltenen Daten in die aktuelle Datenbank einzulesen. Oder man hat eine Anwendung, die beim Kunden läuft, und diese soll in regelmäßigen Abständen prüfen, ob es ein Update für diese Anwendung gibt und die neue Datei bei Bedarf aus dem Internet herunterladen. In diesem Beitrag zeigen wir die Technik, mit der solche Anforderungen umgesetzt werden. Dabei nutzen wir verschiedene Techniken, zum Beispiel per API oder mit dem XMLHTTP-Objekt der Bibliothek Microsoft XML, v6.0.

Download per API

Ein Download via API ist die am schnellsten programmierte und einfachste Methode, um eine Datei aus dem Internet herunterzuladen, die auch frei verfügbar ist (also auch über die Eingabe der URL in einen Webbrowser heruntergeladen werden könnte). Die Lösung zu diesem Ansatz sieht wie in Listing 1 aus. Hier deklarieren wir zunächst eine API-Funktion namens **URLDownloadToFile** in der 32-Bit-Version und in der 64-Bit-Version sowie eine Konstante.

Danach rufen wir die Prozedur **DownloadAPI** auf, in der wir die URL der herunterzuladenden Datei sowie den Zielpfad angeben. Damit ausgestattet rufen wir die API-Funktion auf und übergeben dieser mit dem zweiten

```
#If VBA7 Then
Private Declare Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" _
    (ByVal pCaller As Long, ByVal szURL As String, ByVal szFileName As String, _
    ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long
#Else
Private Declare PtrSafe Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" _
    (ByVal pCaller As LongPtr, ByVal szURL As String, ByVal szFileName As String, _
    ByVal dwReserved As LongPtr, ByVal lpfnCB As LongPtr) As LongPtr
#End If

Const ERROR_SUCCESS = 0

Sub DownloadAPI()
    Dim strUrl As String
    Dim strDatei As String
    strUrl = "http://www.access-im-unternehmen.de/pic001.png"
    strDatei = CurrentProject.Path & "\pic001.png"
    Debug.Print URLDownloadToFile(0&, strUrl, strDatei, BINDF_GETNEWESTVERSION, 0&) = ERROR_SUCCESS
End Sub
```

Listing 1: Einfacher Code zum Herunterladen einer Datei

und dritten Parameter die URL und den Zielpfad. Die übrigen Parameter bleiben leer. Wenn wir die Prozedur nun aufrufen und die angegebene Datei vorhanden ist, wird sie unter dem angegebenen Pfad auf dem lokalen Rechner gespeichert.

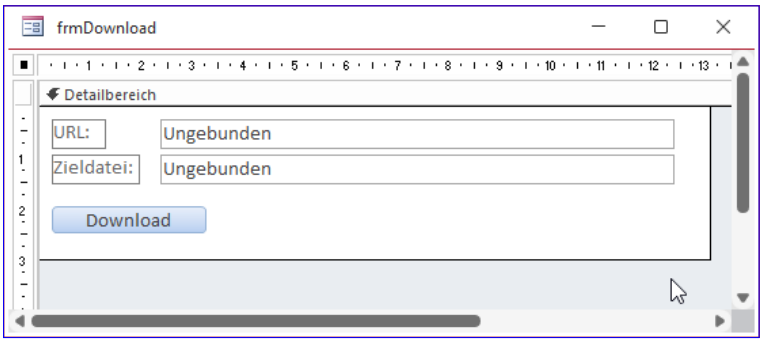

Bild 1: Entwurf des Formulars zum Herunterladen von Dokumenten aus dem Internet

Funktion auf Basis dieser Prozedur

In vielen Fällen wollen wir die Parameter einer solchen Prozedur wie die URL der herunterzuladenden Datei und den Zielpfad dynamisch angeben und diese nicht fest im Code vorgeben.

Dann erstellen wir eine Funktion auf Basis der Prozedur, die wie folgt aussieht:

```
Public Function DownloadAPI(strURL As String, _
        strDatei As String) As Boolean
    If URLDownloadToFile(0&, strURL, strDatei, 0&, 0&) _
            = ERROR_SUCCESS Then
        DownloadAPI = True
    End If
End Function
```

Der Aufruf erfolgt beispielsweise über die folgende Prozedur:

```
Public Sub Test_DownloadAPI()
    Dim strURL As String
    Dim strDatei As String
    strURL = "http://www.access-im-unternehmen.de/pic001.png"
    strDatei = CurrentProject.Path & "\pic001.png"
    Debug.Print DownloadAPI(strURL, strDatei)
End Sub
```

Mehr Komfort per Formular

Wenn wir dem Benutzer die Steuerung des gesamten Vorgangs überlassen wollen, können wir ihm ein Formular wie in Bild 1 hinzufügen. Dieses enthält zwei Textfelder – eines zum Eingeben der URL der Quelldatei und eines für die Zieldatei – und eine Schaltfläche namens **cmdDownload**.

Außerdem hinterlegen wir noch eine Schaltfläche, mit welcher der Benutzer den Namen der Datei angeben kann, unter welcher die heruntergeladene Datei gespeichert werden soll. Diese Schaltfläche heißt **cmdDateiauswahl** und ruft die folgende Prozedur auf:

```
Private Sub cmdDateiauswahl_Click()
    Dim objFileDialog As FileDialog
    Set objFileDialog = FileDialog(msoFileDialogSaveAs)
    objFileDialog.Title = "Zieldatei festlegen"
    objFileDialog.InitialFileName = _
        CurrentProject.Path & "\"
    If objFileDialog.Show = True Then
        Me!txtZieldatei = objFileDialog.SelectedItems(1)
    End If
End Sub
```

Für die Verwendung der **FileDialog**-Klasse benötigen wir einen Verweis auf die Bibliothek **Microsoft Office 16.0**

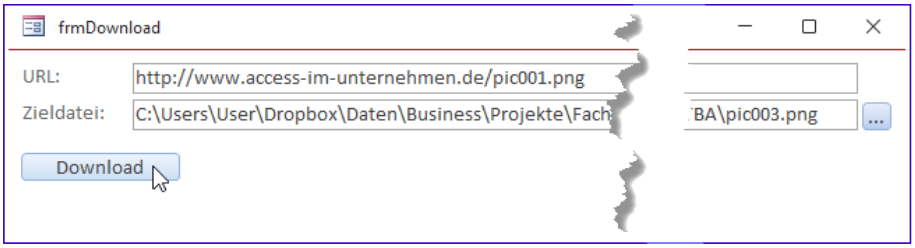

Bild 2: Formular zur Eingabe von Quelle und Ziel beim Download

Object Library (wie wir diesen hinzufügen, zeigen wir weiter unten).

Wenn der Benutzer auch noch die URL wie in Bild 2 ausfüllt, können wir die Schaltfläche **cmdDownload** nutzen, für die wir die folgende Prozedur hinterlegen:

```
Private Sub cmdDownload_Click()
    If DownloadAPI(Me!txtURL, Me!txtZieldatei) Then
        MsgBox "Download erfolgreich!"
    End If
End Sub
```

Für etwas mehr Komfort haben wir die Eigenschaft **Horizontaler Anker** der beiden Textfelder auf **Beide** eingestellt. So können wir die Felder verbreitern, indem wir die Breite des Formulars vergrößern. Die gleiche Eigenschaft stellen wir für die beiden Bezeichnungsfelder dann wieder auf **Links** ein, da diese automatisch auf **Rechts** eingestellt wurden.

Außerdem legen wir diese Eigenschaft für die Schaltfläche **cmdDateiauswahl** auf **Rechts** ein, damit sie beim Vergrößern nach rechts verschoben wird (siehe Bild 3).

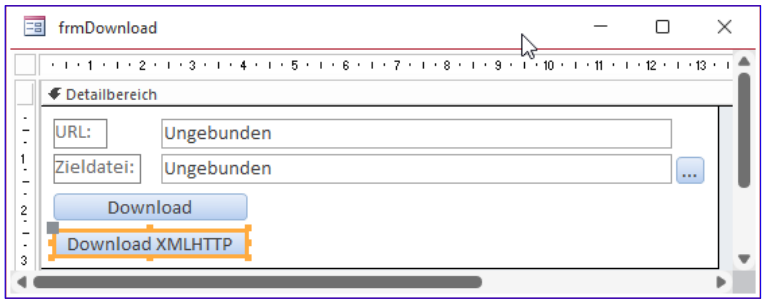

Bild 3: Neue Schaltfläche für eine Download-Alternative

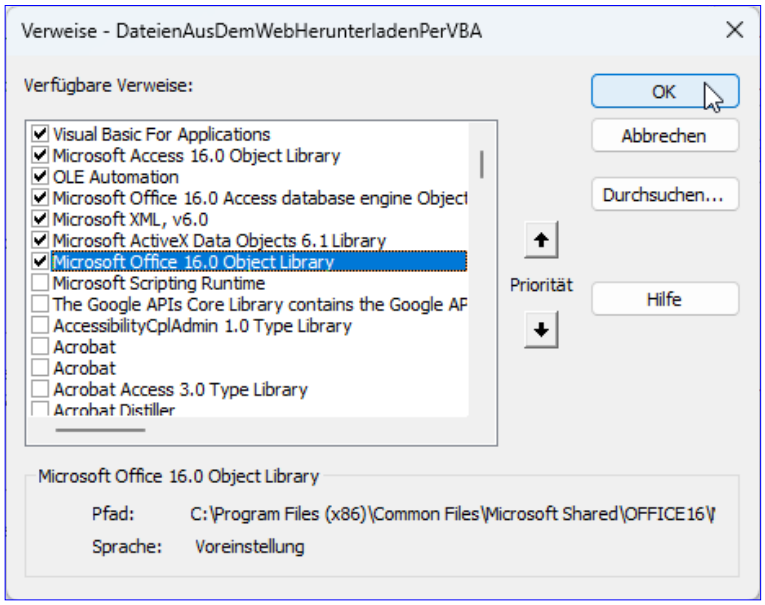

Bild 4: Verweise für die **ServerXMLHTTP**-Variante

Download per ServerXMLHTTP-Objekt

Die zweite Methode, die wir in diesem Beitrag vorstellen, erfolgt mit einem Objekt aus der Bibliothek **Microsoft XML, v6.0**. Um dieses zu verwenden und dabei die Vorteile von IntelliSense zu benutzen, benötigen wir einen Verweis auf diese Bibliothek. Dazu wählen wir im VBA-Editor den Menübefehl **Extras|Verweise** aus und fügen im Dialog **Verweise** den entsprechenden Eintrag durch Anklicken des Kontrollkästchens hinzu (siehe Bild 4).

Das Gleiche erledigen wir, wenn noch nicht geschehen, für die Bibliothek Microsoft **Office 16.0 Object Library**. Schließlich benötigen wir noch einen dritten Verweis, nämlich den auf die Bibliothek **Microsoft ActiveX Data Objects 6.1 Library**. Diese enthält die Elemente, die wir für das Speichern der heruntergeladenen Datei benötigen, denn im Gegensatz zur API-Funktion **URLDownloadToFile** hat das Objekt **ServerXMLHTTP** keine Methode zum direkten Speichern des heruntergeladenen Objekts im Dateisystem.

Für die neue Variante des Downloads erstellen wir eine weitere Funktion namens **DownloadXML**. Diese sieht wie in Listing 2 aus und enthält die gleichen beiden Parameter wie die Funktion **DownloadAPI**.

Um nun die Klasse **ServerXMLHTTP** zu nutzen, deklarieren wir zunächst eine Objektvariable des Typs **MSXML2.**

Bilder per COM-Add-In hinzufügen

Im Beitrag »Bilder für Buttons und Co. schnell hinzufügen« haben wir eine Prozedur und eine Funktion vorgestellt, mit der Sie mehrere Bilder für die Anzeige in Schaltflächen et cetera gleichzeitig zu einer Access-Datenbank hinzufügen können. Allerdings ist es unpraktisch, eine solche Funktion in jede Datenbank kopieren zu müssen, in der wir sie sehen wollen. Also bauen wir uns ein COM-Add-In, mit dem wir nicht eine weitere Schaltfläche mit der neuen Funktion zum Ribbon hinzufügen – sondern gleich die vorhandene Schaltfläche mit der neuen Funktion belegen! Der Benutzer braucht sich also gar nicht umzugewöhnen und kann wie gewohnt Bilder zur Anwendung hinzufügen.

Hinweis

Die hier verwendete Entwicklungsumgebung **twinBASIC** ist für die Erstellung von DLLs für 32-Bit-Office kostenlos, bei der 64-Bit-Version wird ein Splash-Screen eingeblendet.

Die 64-Bit-Version kann unter folgendem Link erworben werden (in einer Zeile):

```
https://shop.minhorst.com/access-tools/364/twinbasic-professional-edition-jahreslizenz?c=78
```

Der Download enthält jedoch die 64-Bit-DLL zum Einsatz auf Ihrem System inklusive Setup.

Ausgangssituation

Wenn wir wie in Bild 1 auf den Ribbonbefehl **Durchsuchen...** klicken, erscheint ein Dateiauswahldialog, mit dem allerdings nur eine Bilddatei gleichzeitig ausgewählt werden kann. Im Beitrag **Bilder für Buttons und Co. schnell hinzufügen** (**www.access-im-unternehmen.de/1436**) haben wir eine Funktion vorgestellt, mit der wir auf die gleiche Weise mehrere Bilddateien zur Datenbank und somit zur Tabelle **MSysResources** hinzufügen können.

Funktion per Ribbonanpassung »überschreiben«

Diese Funktion wollen wir nun so in Access integrieren, dass sie durch Anklicken der Schaltfläche **Durchsu-**

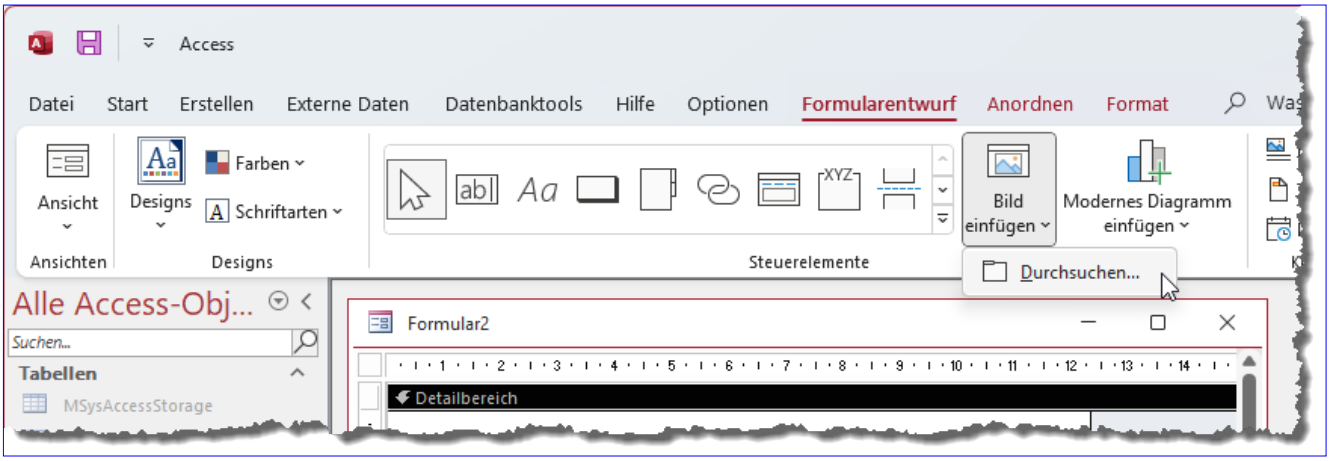

Bild 1: Hinzufügen eines Bildes als Inhalt eines Bildsteuerelements oder für andere Belange

chen... statt der eingebauten Funktion ausgelöst wird. Das können wir durch eine entsprechende Ribbon-Definition erreichen, in der wir in einem **command**-Element einen Verweis auf das entsprechende Steuerelement und einen Aufruf einer alternativen Prozedur angeben.

Die Ribbon-Definition sieht wie folgt aus:

```
<customUI xmlns="http://schemas.microsoft.com/
                                office/2006/01/customui">
  <commands>
    <command onAction="onAction" idMso="SharedImageBrowse"/>
  </commands>
</customUI>
```

Die hier angegebene Prozedur für das Attribut **onAction** können wir wie folgt definieren:

```
Public Sub OnAction(control As IRibbonControl, _
        ByRef CancelDefault)
    BilderImportieren True
End Sub
```

Diese ruft die Prozedur **BilderImportieren** auf, die wir im oben genannten Beitrag beschrieben haben.

Wenn wir diese Ribbondefinition in der Anwendung unterbringen, zum Beispiel für ein Formular oder als Anwendungsribbon, steht die Funktion zwar zur Verfügung und überschreibt auch die eingebaute Funktion des Ribbon-Steuerelements, aber diese hilft uns dann auch nur in der aktuellen Anwendung weiter. Und wer sucht schon in den eingebauten Steuerelementen des Ribbons nach benutzerdefinierten Funktionen? Deshalb machen wir es einen Schritt professioneller und bauen die Funktion in ein COM-Add-In ein, das die Funktion fortan in jeder Access-Instanz zur Verfügung stellt – egal, welche Anwendung wir darin geöffnet haben.

Einbau in ein COM-Add-In

Den Einbau der beiden Routinen aus dem oben genannten Artikel in ein COM-Add-In beziehungsweise dessen Erstellung erfolgt mit dem Tool **twinBASIC**, das wir schon in einigen Beiträgen erwähnt haben. Interessant als Zusatzinformation zu dem, was Sie im vorliegenden Beitrag lesen, dürfte der Beitrag **twinBASIC – COM-Add-Ins für Access** (**www.access-im-unternehmen.de/1306**) sein.

Download und Installation von twinBASIC

Da twinBASIC mittlerweile eine andere Plattform als Entwicklungsumgebung verwendet, weisen wir nochmal auf den Download und die Installation hin.

Der Download der jeweils neuesten Version erfolgt über diesen Link:

```
https://github.com/twinbasic/twinbasic/releases
```

Bild 2: Download der twinBASIC-Entwicklungsumgebung

Hier erweitern Sie für die gewünschte Version den Bereich Assets (siehe Bild 2) und laden die Datei herunter, beispielsweise **twinBASIC_IDE_BETA_277.zip**.

Anschließend stellen Sie die Option **Zulassen** für die heruntergeladene Datei auf **Ja** ein (siehe Bild 3). Die Eigenschaft finden Sie im **Eigenschaften**-Dialog der Datei, den Sie mit dem entsprechenden Kontextmenü-Befehl öffnen.

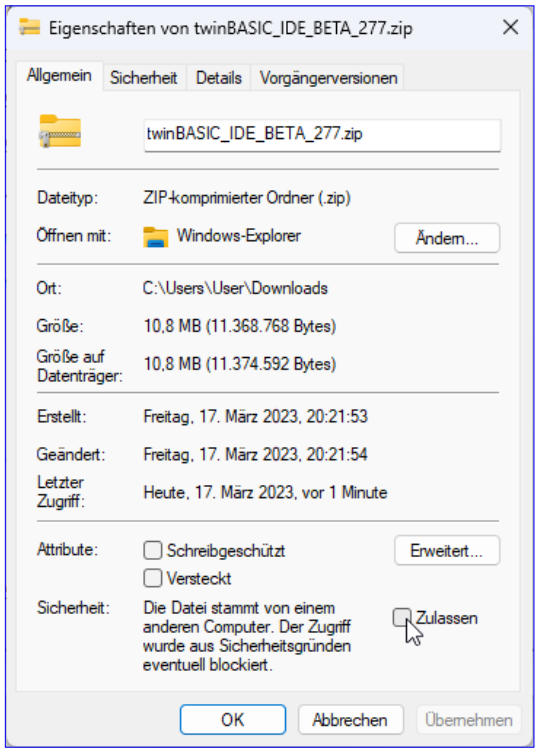

Bild 3: Aktivieren der Sicherheit der Zip-Datei

Danach extrahieren Sie die in der Zip-Datei enthaltenen Dateien in ein Verzeichnis Ihrer Wahl.

Anschließend können Sie durch einen Doppelklick auf die Datei **twinBASIC.exe** direkt mit der Anwendung arbeiten – es ist keine Installation nötig.

Neues COM-Add-In anlegen

Wir legen nach dem Start von twinBASIC dort ein neues Projekt auf Basis der Vorlage **Sample 5. MyCOMAddin** an, das auf der Seite **Samples** der Startseite von **twinBASIC** angezeigt wird.

Dieses Projekt enthält alles, was man für das Erstellen eines eigenen COM-Add-Ins braucht – angefangen von einer Ribbon-Anpassung mit einem individuellen **tab**-Element plus Button, der die ebenfalls integrierte Prozedur auslöst.

Wir wollen unser Projekt **amvAddPictures** benennen und auch die einzige Klasse des Projekts erhält diesen Namen. Der eindeutige Klassen-Identifizierer wurde von twinBASIC automatisch vergeben. Die Klasse implementiert zwei Schnittstellen, nämlich **IDTExtensibility2** und **IRibbonExtensibility**:

```
[ ClassId ("061EF79E-7E28-4BF7-870F-01AF336670CD") ]
Class amvAddPictures
    Implements IDTExtensibility2
    [ WithDispatchForwarding ]
    Implements IRibbonExtensibility
    ... mehr Code
End Class
```

Implementieren der Schnittstelle IDTExtensibility2

Die Schnittstelle **IDTExtensibility2** gibt die Methoden vor, die zu verschiedenen Ereignissen beim Öffnen, Schließen et cetera der Access-Anwendung ausgeführt werden. Uns interessiert nur eine davon, nämlich **OnConnection**. Diese wird nämlich genau dann ausgelöst, wenn der Benutzer eine Access-Instanz öffnet und liefert einige interessante Parameter. Auch hier interessiert uns nur ein Parameter, nämlich **Application**. Dieser liefert einen Verweis auf die aufrufende Anwendung, in diesem Fall **Access.Application**. Da die Funktionen eines COM-Add-Ins meist auf die aktuelle Access-Instanz zugreifen sollen, speichern wir den Verweis auf diese Instanz in einer Objektvariablen, die wir wie folgt deklarieren:

```
Private objAccess As Access.Application
```

Für die Verwendung dieser Klasse benötigen wir, ähnlich wie im VBA-Editor beim Zugriff auf die Objekte anderer Bibliotheken, einen Verweis.

Diesen fügen wir über die Projekteigenschaften hinzu, die wir über den Eintrag **Settings** im Projektexplorer öffnen können. Hier scrollen wir zum Bereich **COM Type Library / ActiveX References**, wo wir im unteren Feld

nach **Access** suchen und den gewünschten Eintrag anhaken (siehe Bild 4). Damit diese Änderung wirksam wird, müssen wir das Projekt speichern, schließen und erneut öffnen.

Schnittstellen implementieren

Die Methoden der Schnittstelle implementieren wir darunter wie in Listing 1. Wir bilden hier auch die weiteren, nicht mit Anweisungen gefüllten Methoden ab, da diese auf jeden Fall im Projekt implementiert werden müssen.

Implementierung der Schnittstelle IRibbonExtensibility

Die zweite Schnittstelle erlaubt das Implementieren einer Funktion, die ebenfalls beim Start von Access ausgelöst

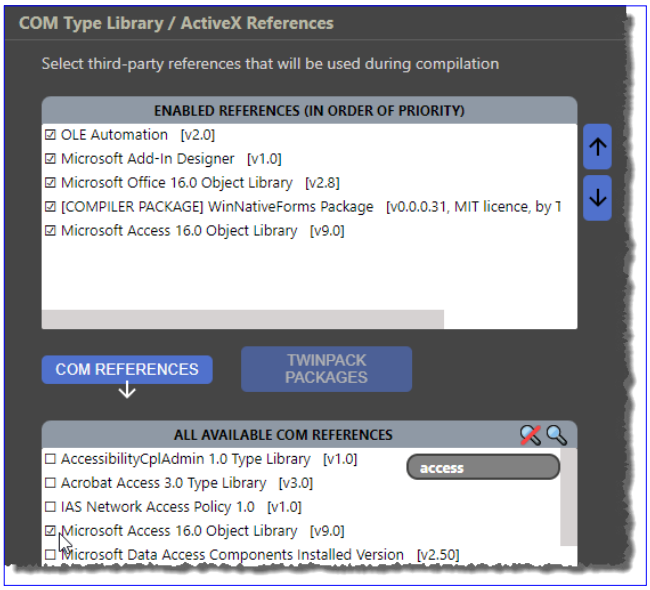

Bild 4: Verweis auf die **Access**-Bibliothek hinzufügen

```
Sub OnConnection(ByVal Application As Object, ByVal ConnectMode As ext_ConnectMode, ByVal AddInInst As Object, _
        ByRef custom As Variant()) Implements IDTExtensibility2.OnConnection
    Set objAccess = Application
End Sub

Sub OnDisconnection(ByVal RemoveMode As ext_DisconnectMode, ByRef custom As Variant()) _
        Implements IDTExtensibility2.OnDisconnection
End Sub

Sub OnAddInsUpdate(ByRef custom As Variant()) Implements IDTExtensibility2.OnAddInsUpdate
End Sub

Sub OnStartupComplete(ByRef custom As Variant()) Implements IDTExtensibility2.OnStartupComplete
End Sub

Sub OnBeginShutdown(ByRef custom As Variant()) Implements IDTExtensibility2.OnBeginShutdown
End Sub
```

Listing 1: Implementierung der Methoden der Schnittstelle **IDTExtensibility2**

```
Private Function GetCustomUI(ByVal RibbonID As String) As String Implements IRibbonExtensibility.GetCustomUI
    Dim strXML As String
    strXML &= "<customUI xmlns=""http://schemas.microsoft.com/office/2006/01/customui"">" & vbCrLf
    strXML &= "  <commands><command onAction=""onAction"" idMso=""SharedImageBrowse""/></commands>"
    strXML &= "</customUI>" & vbCrLf
    Return strXML
End Function
```

Listing 2: Die Funktion, die eine Ribbondefinition entgegennimmt