HAUFE.
Ausgabe 03/2016

ACCESS
IM UNTERNEHMEN

EBAY-ZUGRIFF MIT ACCESS

Erstellen Sie Ihren eigenen Entwickler-Account und durchsuchen Sie die Artikeldatenbank von ebay mit Access (ab S. 59).

In diesem Heft:

3... 2... 1... Meins!

ebay ist nach wie vor die beliebteste Plattform, wenn es um das private Versteigern oder Ersteigern von Dingen geht. Aber es gibt auch mehr und mehr kommerzielle Händler, die ihre Waren auf ebay anbieten und somit eine weitere Alternative zum Onlineshop oder auch zu Amazon nutzen. Wir haben uns einmal angeschaut, wie wir mit Access auf die dort zur Verfügung stehenden Artikel zugreifen können.

Dabei betrachten wir zunächst die reine Artikelsuche und wollen von einer Access-Anwendung aus auf den Webservice von ebay zugreifen. Dazu brauchen wir ein ebay-Konto, einen Entwickler-Account und einiges Know-how, das Ihnen der Beitrag **ebay-Zugriff mit Access** ab S. 59 liefert.

In einem Suchformular geben Sie dort die gewünschten Begriffe ein und erhalten eine Liste der gefundenen Artikel. Dort können Sie dann interessante Einträge auswählen und per Mausklick auf der Webseite von ebay anzeigen lassen.

In der nächsten Ausgabe geht es dann weiter – dort schauen wir uns an, wie Sie eigene Artikel auf ebay zum Verkauf einstellen können.

Wer viel mit dem VBA-Editor programmiert, hat sich an die betagte Benutzeroberfläche gewöhnt. Das ist allerdings kein Grund, nicht einmal über Erweiterungen nachzudenken. Diese kommen in Form einer Software namens **MZ-Tools 8.0**, dem Nachfolger der ebenfalls recht betagten Version 3.0. Im Beitrag **VBA-Editor erweitern mit MZ-Tools 8.0** ab S. 2 beleuchten wir, an welchen Stellen die MZ-Tools Ihnen Arbeit abnehmen können.

Ein etwas stiefmütterliches Dasein fristen die Ereignisse von Berichten, die etwa beim Öffnen eines Berichts ausgelöst werden. Daher widmen wir diesen Ereignissen eine kleine Beitragsreihe. Den Start machen wir ab S. 12 mit dem Beitrag **Berichtsereignisse – Grundlagen**. Hier schauen wir uns an, welche Ereignisse es überhaupt gibt.

Auf S. 15 geht es dann unter dem Titel **Berichtsereignisse: Beim Öffnen** weiter: Hier betrachten wir das Ereignis, das gleich beim Öffnen des Berichts ausgelöst wird. Es ermöglicht etwa die Prüfung, ob der Bericht überhaupt Daten anzeigen wird, und bietet die Möglichkeit, etwa in einem solchen Fall das Öffnen abzubrechen.

Ebenfalls interessant sind die beiden Ereignisse eines Berichts, die wir im Beitrag **Berichtsereignisse: Bei Aktivierung/Deaktivierung** ab S. 20 vorstellen. Diese werden ausgelöst, wenn ein Bericht den Fokus erhält oder verliert. Das ist praktisch, denn so können Sie beispielsweise Menü-Einträge, die zum Ausdrucken eines solchen Berichts dienen, aktivieren oder deaktivieren.

Gelegentlich kommt es vor, dass ein Bericht gar keine Daten liefert, weil das Filterkriterium dies nicht hergibt. Was dann geschieht, zeigen wir im Beitrag **Berichtsereignisse: Bei ohne Daten** ab S. 24.

Oft nachgefragt wurde bei uns in letzter Zeit das Thema Umwandlung von Berichten in das PDF-Format und der Versand eben solcher Dokumente per E-Mail. Der Beitrag **Bericht per PDF und Mail verschicken** zeigt ab S. 26, wie Sie dies mit wenigen Zeilen Code erledigen können.

In der vorherigen Ausgabe haben wir eine Beitragsreihe zum Thema API-Programmierung gestartet. In dieser Ausgabe schauen wir uns an, wie dies in der Praxis funktioniert. Dazu haben wir uns das Thema Windows-Registry herausgesucht. Im Beitrag **API-Praxis: Zugriff auf die Registry** zeigen wir ab S. 31, wie Sie mit API-Befehlen Einträge aus der Registry lesen, anlegen oder löschen.

Schließlich folgt der zweite Teil unserer Ticketverwaltung: Unter der Überschrift **Ticketsystem mit Access, Teil 2** finden Sie ab S. 45 heraus, wie Sie die Ticketverwaltung um weitere Funktionen erweitern. Hier ist einiges an Finetuning zwischen Outlook und Access-Anwendung nötig, damit die Kommunikation reibungslos funktioniert. Außerdem schauen wir uns die Funktionsweise des Formulars zum Erstellen von Tickets auf Basis von Kunden-E-Mails genauer an.

Nun aber: Viel Spaß beim Lesen!

Ihr Michael Forster

VBA-Editor erweitern mit MZ-Tools 8.0

Es gibt nur wenige Tools, mit denen sich der gute, alte VBA-Editor aufpeppen lässt. Bislang war MZ-Tools eines davon, aber auch schienen die Entwickler seit Jahren keine Ambitionen zu zeigen, das Tool noch weiter voranzubringen. Seit kurzem gibt es jedoch eine neue Version namens MZ-Tools 8.0, die gegenüber der Vorgängerversion komplett neu gestaltet zu sein scheint. Wir schauen uns diese im Detail an und sagen Ihnen, ob sich die Invesition lohnt.

Um es vorab kurz zu fassen: Der Nachteil der neuen Version von MZ-Tools ist, dass es im Gegensatz zur vorherigen, veralteten Version nicht mehr kostenlos ist. Der Hersteller bittet mit 79,95 EUR plus Mehrwertsteuer zur Kasse. Allerdings erhält man MZ-Tools in Versionen für Visual Studio von 2005 bis 2015, Office 2000 bis 2016 in der 32- und 64-bit-Version sowie für VB 5.0 und VB 6.0.

Größere Mengen für mehr als einen Entwickler werden entsprechend rabattiert.

Bis Sie sich für den Kauf entscheiden, erhalten Sie allerdings eine 30-Tage-Testversion. Den Download dazu finden Sie unter folgendem Link:

```
http://www.mztools.com/v8/download_trial.aspx
```

Nach der Installation brauchen Sie nur den VBA-Editor zu öffnen, um auf die Funktionen von MZ-Tools 8.0 zu stoßen. Diese treten in Form zweier Symbolleisten in Erscheinung (s. Bild 1).

Außerdem finden Sie einen Eintrag namens **MZ-Tools** in der Menüleiste, der wie in Bild 2 aussieht.

Schließlich liefert auch das Kontextmenü des VBA-Editors noch einige Befehle, die schnell erreichbar sein sollen (s. Bild 3).

Von 3.0 nach 8.0

Wenn Sie Access im Unternehmen schon eine Weile verfolgen, haben Sie bereits über MZ-Tools gelesen – zum Beispiel hier:

```
http://www.access-im-unternehmen.de/431.0.html
```

Wir haben dort sogar eine eigene **.ini**-Datei zum Download angeboten, mit der Sie unsere Vorlagen und Einstellungen unter MZ-Tools nutzen konnten. Wenn Sie selbst bereits eine solche Datei mit Ihren eigenen Einstellungen gefüllt haben, wird es Sie vermutlich freuen, dass diese Einstellungen mit der neuesten Version von MZ-Tools automatisch übernommen werden. Anderenfalls können Sie

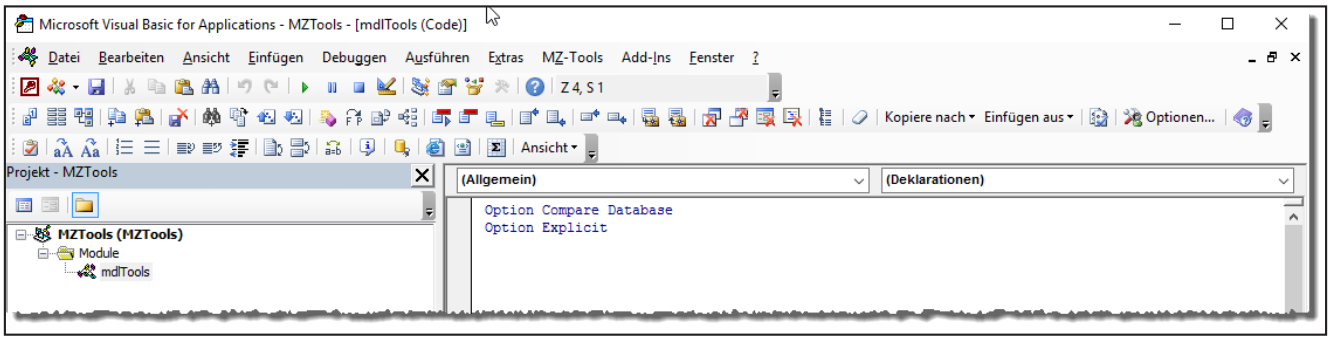

Bild 1: MZ-Tools 8.0 kommt mit zwei Symbolleisten sowie einem Menüleisten-Eintrag

die Einstellungen, die wir hier vorstellen, dem Download zu diesem Beitrag entnehmen und für Ihre eigenen Zwecke nutzen und anpassen. Dazu kopieren Sie einfach den Inhalt des Ordners aus dem Download in den Ordner, in dem sich die Konfigurationsdateien Ihrer MZ-Tools befinden – mehr dazu gleich.

Von Rechner zu Rechner

Wenn Sie bei der alten Version Ihre Konfigurationsdateien sichern oder auf einen anderen Rechner übertragen wollten, mussten Sie erst einmal herausfinden, wo diese gespeichert sind. Unter MZ-Tools 8.0 finden Sie diese Information gleich unten im **Optionen**-Dialog, den Sie direkt über die Symbolleiste öffnen können (s. Bild 4).

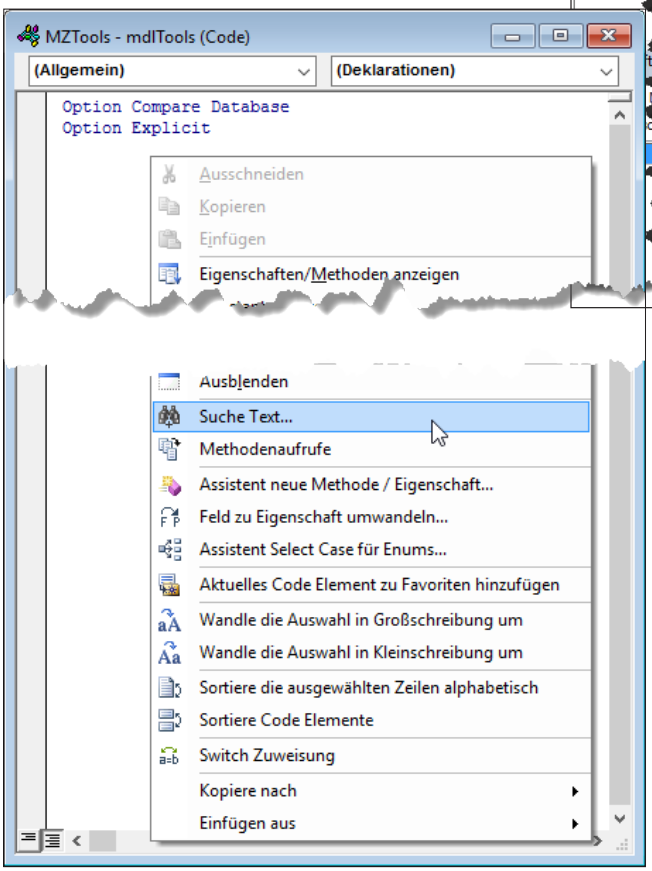

Bild 3: MZ-Tools per Kontextmenü

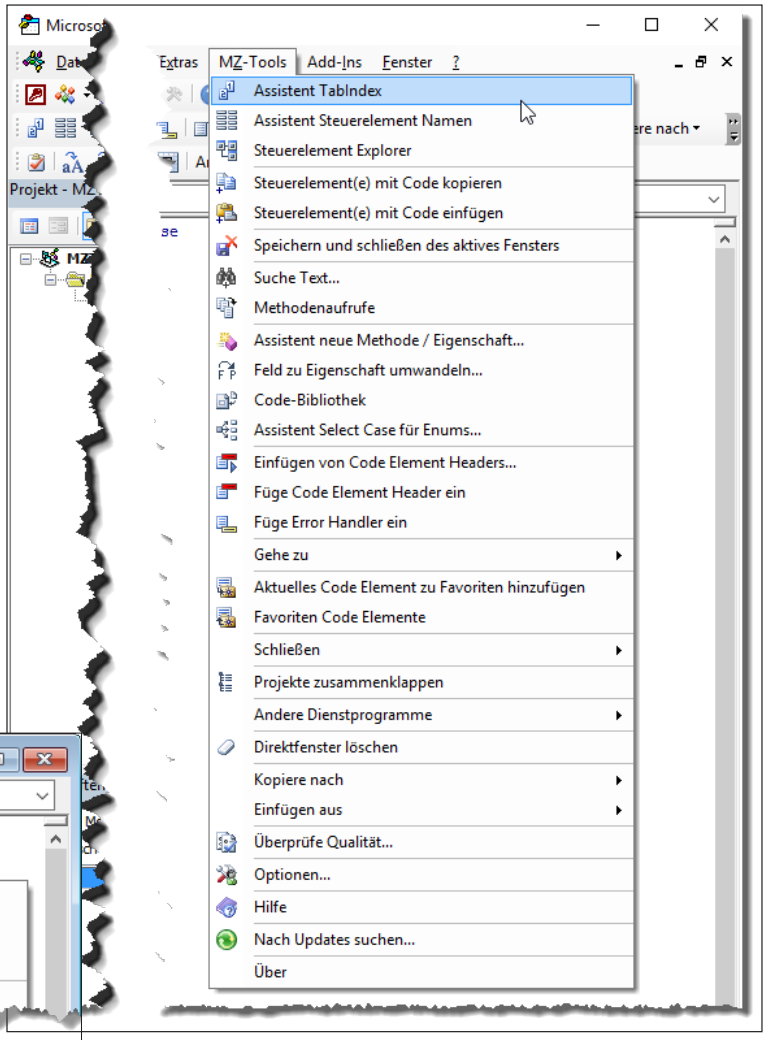

Bild 2: Befehle der Menüleiste von MZ-Tools

Von Text zu XML

Die Konfigurationsdaten werden nunmehr nicht mehr in einer Textdatei gespeichert, sondern im XML-Format.

Einstellungen in der Cloud

Wenn Sie mehrere Rechner zum Programmieren nutzen, möchten Sie vielleicht auf jedem Rechner die gleichen Einstellungen nutzen.

In diesem Fall brauchen Sie den Ordner, der die Einstellungsdateien speichert, einfach in einen Dropbox-, Google Drive- oder OneDrive-Ordner oder einen anderen Ordner, der mit der Cloud synchronisiert wird, zu speichern.

Frühere Lieblingsfeatures

Meine persönlichen Lieblingsfunktionen unter MZ-Tools 3.0 waren die folgenden (möglicherweise gibt es nützliche Features, die ich schlicht nicht erkannt habe – aber die nachfolgend aufgelisteten haben es mir angetan):

- Aufdecken nicht verwendeter Variablen und Parameter
- Einfügen von Code-Vorlagen wie einzelne Funktionen oder komplette Abschnitte mit API-Deklarationen, Konstanten, Routinen et cetera
- Festlegen von Tastenkombinationen. Meine Lieblingskombination: **Strg + Umschalt + A**, um die aktuell markierten Zeilen auszukommentieren, und **Strg + Umschalt + E**, um die Kommentarzeichen wieder zu entfernen
- Die Suchfunktion! Während man mit der eingebauten Suchfunktion des VBA-Editors die einzelnen Suchstellen nacheinander anspringt, erhält man hier eine Auflistung der Fundstellen mit Modul und betroffener Zeile in einem eigenen Fenster. Per Doppelklick springt man dann zu diesen Stellen. Einziger Haken: zum erneuten Suchen musste man die Funktion erneut starten. Wir werden sehen, ob MZ-Tools 8.0 hier nachgebessert hat.
- Suchen von Routinenaufrufen

Schauen wir also, ob diese Funktionen noch vorhanden sind – und wie wir unsere Konfigurationsdatei von der alten Version in die neue Version überführen können. Es wäre doch schade, wenn wir unsere Code-Vorlagen nicht komfortabel dorthin überführen könnten!

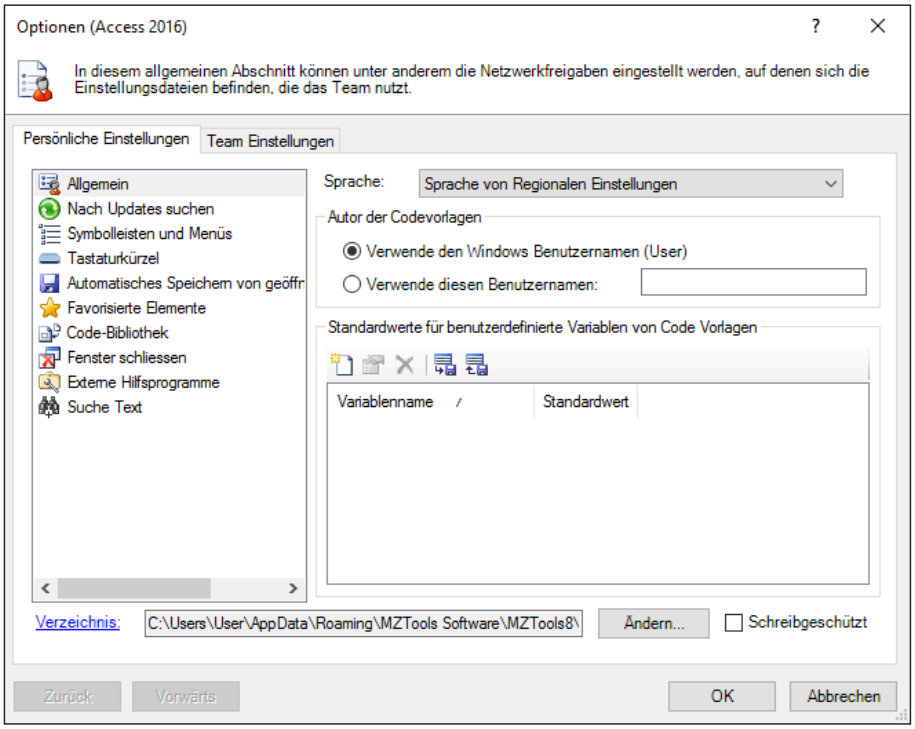

Bild 4: Optionen-Dialog der MZ-Tools 8.0

Ein Problem tritt mit der neuen Version jedenfalls nicht mehr auf: MZ-Tools ist jetzt umfassend für alle angegebenen Entwicklungsumgebungen. Früher ist es vielen Benutzern passiert, dass sie sich die VB6-Version heruntergeladen und installiert haben. Danach wunderten sie sich, warum die MZ-Tools nicht im VBA-Editor von Access erschienen. Die Lösung war einfach: Es gab für VBA eine eigene Version, die sie schlicht nicht installiert hatten.

Nicht verwendete Variablen und Parameter finden

Diese Funktion hilft, wenn Sie länger an Modulen und Routinen arbeiten und diese nach und nach immer wieder verändern. Dann kommt es leicht vor, dass die Parameter oder aber die Deklaration der Variablen innerhalb der Routine Elemente enthält, die in der gesamten Routine nicht verwendet werden. In der neuen Version von MZ-Tools finden Sie diese unter dem Menü-Eintrag **MZ-Tools|Qualität**. Damit öffnen Sie einen Dialog, der verschiedene Optionen bietet (s. Bild 5).

Wenn Sie hier die Option **Überprüfung von verwaistem Code** aktivieren und auf **OK** klicken, durchsucht MZ-Tools die Module nach nicht verwendeten Variablen und Parametern und gibt diese wie in Bild 6 aus.

Das Schöne ist, dass Sie per Doppelklick auf einen dieser Einträge direkt zur entsprechenden Code-Stelle springen und den überflüssigen Code entfernen können.

Einfügen von Code-Vorlagen

Ich verwende manche Code-Schnippsel immer wieder in meinen Anwendungen – zum Beispiel Funktionen wie zum Ermitteln des ISO-Datums aus einer Datumsangabe oder das Modul mit den Funktionen zum Öffnen von Dateidialogen.

Damit ich diese einfach und schnell zu einem VBA-Projekt hinzufügen kann, habe ich unter MZ-Tools 3.0 die dazu vorgesehene Funktion genutzt.

Unter der neuen Version gibt es diese Version ebenfalls, beim Update werden die Vorlagen von der alten Version automatisch übernommen.

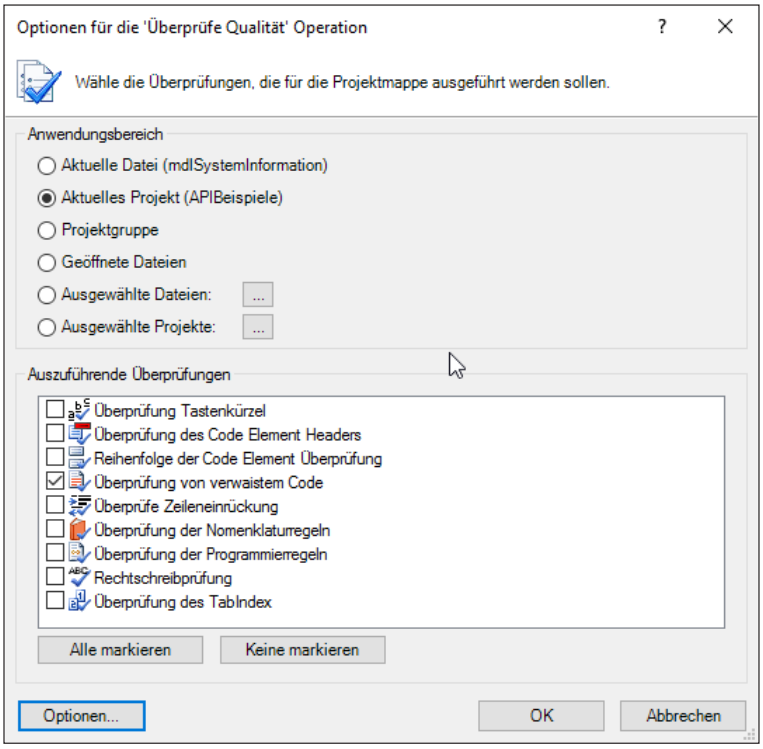

Bild 5: Optionen zum Überprüfen der Code-Qualität

Den Dialog zur Auswahl der Code-Vorlagen öffnen Sie mit dem Menü-Eintrag **MZ-Tools|Code-Bibliothek**. Er sieht wie in Bild 7 aus und liefert links eine Liste aller Vorlagen und rechts den Inhalt der aktuell ausgewählten Vorlage. Ein Klick auf die Schaltfläche **Einfügen** fügt den Inhalt der aktuellen Vorlage an der Position der Einfügemarke im aktuellen Code-Fenster ein. Klicken Sie in diesem Dialog auf die Schaltfläche **Optionen**, öffnet sich der Optionen-Dialog von MZ-Tools, diesmal mit der Anzeige der Codevorlagen.

Hier können Sie per Doppelklick auf einen der Einträge einen Dialog öffnen, mit dem Sie weitere Einstellungen zu jeder

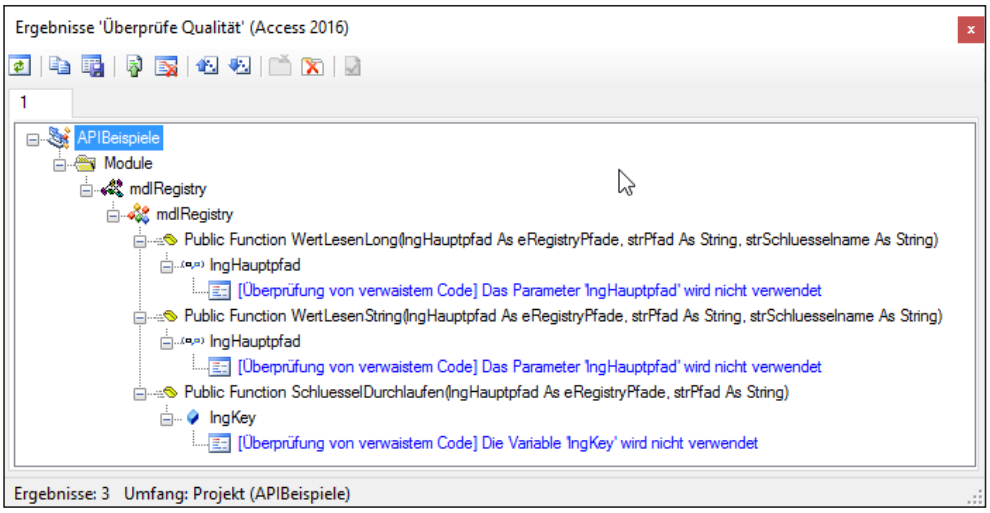

Bild 6: Liste der nicht verwendeten Parameter und Variablen

Berichtsereignisse – Grundlagen

Neben den verschiedenen Bereichen und Steuerelementen liefern Berichte auch eine Reihe von Ereigniseigenschaften, für die Sie wie bei Formularen Ereignisprozeduren hinterlegen können. Die Ereignisse werden zwar weniger durch Benutzerinteraktion ausgelöst als eher durch die Abfolge von Aktionen beim Zusammenstellen der benötigten Daten und beim Rendern des Berichts, aber dennoch ist es wichtig, die verschiedenen Ereignisse zu kennen.

Ereignisse in Berichten

Berichte bieten bereits seit Access 2007 erheblich mehr Ereignisse für Berichte als in den Vorgängerversionen – und fast so viele wie für Formulare. Die Möglichkeiten der Interaktion sind um einiges besser geworden; so können Sie etwa auf einen Datensatz in einem Bericht klicken, um ein Formular mit den Details zum Bearbeiten zu öffnen.

Diese Ereignisse beziehen sich aber alle auf eine mit Access 2007 eingeführte Darstellungsart, nämlich die **Berichtsansicht**. Mehr zu dieser neuen interaktiven Ansicht finden Sie in einem weiteren Beitrag.

Bis ein Bericht überhaupt einmal angezeigt ist, geschieht allerdings schon eine Menge und in vielen Fällen können Sie den Ablauf per Ereignisprozedur beeinflussen. Nachfolgend finden Sie zunächst eine Auflistung der Ereignisse eines Berichts und der Bereiche eines Berichts; im Anschluss lernen Sie einige Anwendungsfälle der Ereignisse kennen.

Zusammenfassung der Berichtsereignisse

Der Bericht löst folgende Ereignisse aus, die in den bisherigen Access-Versionen ebenfalls unterstützt wurden und die für die reine Anzeige eines Berichts wichtig sind:

- **Beim Öffnen**: Wird beim Öffnen, aber vor dem Erstellen des Berichts ausgelöst. Dient beispielsweise der Übergabe von Parametern oder zum Abbrechen der Ausgabe des Berichts.
- **Bei Aktivierung**: Bericht wird aktiviert oder gedruckt.
- **Bei Seite**: Wird nach Abschluss der **Bei Formatierung**- und der **Beim Drucken**-Ereignisse der Berichtsbereiche, aber vor dem Anzeigen der Seite ausgelöst.
- **Beim Schließen**: Wird beim Schließen des Berichts ausgelöst – etwa durch Betätigen der Schließen-Schaltfläche.
- **Bei Deaktivierung**: Bericht wird deaktiviert, da ein anderes Objekt den Fokus erhält oder der Bericht geschlossen wird.
- **Bei Ohne Daten**: Wird ausgelöst, wenn die Datensatzquelle des Berichts keine Daten enthält.
- **Bei Fehler**: Wird beim Auftreten eines Fehlers ausgelöst.

Zusammenfassung der Bereichsereignisse

Die Bereiche **Berichtskopf**, **Berichtsfuß**, **Seitenkopf**, **Seitenfuß**, **Kopf** und **Fuß** der einzelnen Gruppierungen und der Detailbereiche lösen ebenfalls Ereignisse aus.

Dabei werden die Ereignisse **Beim Formatieren** und **Beim Drucken** je Element eines jeden Bereichs mindestens einmal aufgerufen:

- **Beim Formatieren**: Wird ausgelöst, wenn die Daten ermittelt sind, bevor der Bereich formatiert wird. Dieses Ereignis wird gegebenenfalls mehrere Male ausgelöst. Wenn es sich beim aktuell formatierten Bereich um einen Gruppenkopf handelt, haben Sie Zugriff auf die

im Gruppenkopf angezeigten Daten und auf die Daten des ersten Datensatzes der Gruppierung im Detailbereich. Beim Gruppenfuß stehen die Daten des Gruppenfußbereichs und des letzten Datensatzes des Bereichs zur Verfügung. Im Detailbereich bietet dieses Ereignis Zugriff auf die Daten des aktuellen Datensatzes.

- **Beim Drucken**: Wird ausgelöst, wenn der Bereich formatiert, aber noch nicht gerendert ist. In diesem Ereignis können Sie je Bereich auf die gleichen Daten zugreifen wie im **Beim Formatieren**-Ereignis. Mit »Drucken« ist hier nicht die Ausgabe auf Papier gemeint, sondern die grafische Ausgabe in das »Dokument«, das dann auf dem Bildschirm oder auf Papier dargestellt werden kann.

- **Bei Rückname**: Wird jedes Mal ausgelöst, wenn Access einen Bereich infolge Positionierungsberechnungen neu formatieren muss (steht nicht für den Seitenkopf zur Verfügung).

Die genaue Abfolge der Ereignisse eines Berichts ist relativ komplex. Mehr Gruppierungen oder spezielle Bedingungen wie etwa das Zusammenhalten von Gruppierungen führen zu einer unüberschaubaren Menge von Ereignissen. Allein der Aufruf eines Berichts mit einer Gruppierung, der nur einen Datensatz anzeigt, löst die folgenden Ereignisse in der angegebenen Reihenfolge aus:

- Bericht **Beim Öffnen**
- Bericht **Bei Aktivierung**
- Seitenkopfbereich **Beim Formatieren**
- Gruppenfuß0 **Beim Formatieren**
- Detailbereich **Beim Formatieren**
- Gruppenfuß0 **Beim Formatieren**
- Seitenfußbereich **Beim Formatieren**
- Seitenkopfbereich **Beim Formatieren**
- Seitenkopfbereich **Beim Drucken**
- Gruppenfuß0 **Beim Formatieren**
- Gruppenfuß0 **Beim Drucken**
- Detailbereich **Beim Formatieren**
- Detailbereich **Beim Drucken**
- Gruppenfuß0 **Beim Formatieren**
- Gruppenfuß0 **Beim Drucken**
- Seitenfußbereich **Beim Formatieren**
- Seitenfußbereich **Beim Drucken**
- Bericht **Bei Seite**
- Bericht **Beim Schließen**
- Bericht **Bei Deaktivierung**

Zugriff auf die Berichtsbereiche

Der Zugriff auf die Steuerelemente eines Berichts erfolgt genau wie in Formularen. Interessanter sind da die Elemente, die in Formularen nicht vorhanden sind – die einzelnen Bereiche der Berichte.

Warum muss man eigentlich wissen, wie man per VBA auf diese Bereiche zugreift? Weil es Situationen gibt, in denen Sie beispielsweise einen Bereich ein- oder ausblenden oder die Eigenschaften eines Bereichs anpassen müssen.

Auf einen Berichtsbereich greifen Sie über die Auflistung **Section** zu. Als Argument geben Sie entweder den Namen,

Berichtsereignisse: Beim Öffnen

Neben den verschiedenen Bereichen und Steuerelementen liefern Berichte auch eine Reihe von Ereigniseigenschaften, für die Sie wie bei Formularen Ereignisprozeduren hinterlegen können. Eines dieser Ereignisse wird beim Öffnen des Berichts ausgelöst. Es ist eine gute Gelegenheit, verschiedene Dinge zu prüfen und darauf zu reagieren. Beispiele sind das Auswerten von Öffnungsargumenten oder das Abbrechen des Öffnungsvorgangs.

Beispiele für den Einsatz der Berichts- und Bereichsereignisse in der Seitenansicht

Die Berichtsereignisse lassen sich teilweise für recht spezielle Aktionen einsetzen. In diesem und weiteren Beiträgen finden Sie einige Beispiele, damit Sie ein Gefühl für das richtige Einsetzen der Berichtsereignisse erhalten.

Beim Öffnen: Auswertung von Öffnungsargumenten

Der richtige Zeitpunkt zum Auswerten eines Öffnungsarguments ist das Ereignis **Beim Öffnen**. Mit dem Öffnungsargument der **DoCmd.OpenReport**-Anweisung lässt sich beispielsweise ein Parameter zum Filtern der Datensatzquelle übergeben.

Da dies aber relativ langweilig ist, finden Sie nachfolgend ein Beispiel, wie Sie die Gruppierung eines Berichts beim Öffnen verändern können. Voraussetzung ist der Bericht **rptGruppierungenTauschen**, der wie in Bild 1 aufgebaut ist.

Der Clou an diesem Bericht ist, dass Sie mit wenigen Zeilen Code die Gruppierungsebenen vertauschen können. Dazu sind folgende, in der Abbildung nicht sichtbare Eigenschaften einzustellen:

- Name des Beschriftungsfeldes im Berichtskopf: **lblUeberschrift**
- Name des **LieferantID**-Kopfbereichs: **Gruppenkopf0**
- Name des Beschriftungsfeldes im **LieferantID**-Kopfbereich: **lblUeberschrift Gruppierung0**
- Name des Textfeldes im **LieferantID**-Kopfbereich: **txtUeberschriftGruppierung0**
- Name des **KategorieID**-Kopfbereichs: **Gruppenkopf1**

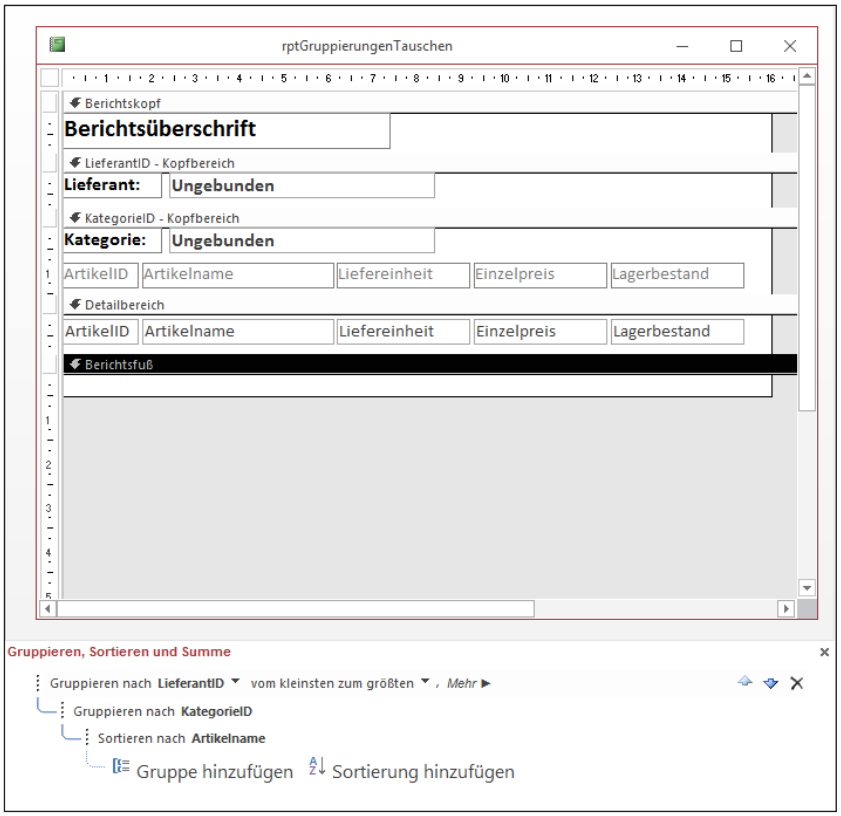

Bild 1: Bericht mit zwei Gruppierungen, die dynamisch getauscht werden sollen

- Name des Beschriftungsfeldes im **KategorieID**-Kopfbereich: **lblUeberschrift Gruppierung0**

- Name des Textfeldes im **KategorieID**-Kopfbereich: **txtUeberschriftGruppierung1**

Wie bringen Sie nun Dynamik ins Spiel? Betrachten Sie den reinen Ablauf, so rufen Sie den Bericht auf und übergeben mit dem Öffnungsargument Informationen über die im Bericht anzuzeigenden Daten. Das sieht etwa folgendermaßen aus (in einer Zeile):

```
DoCmd.OpenReport "rptGruppierungenTauschen", 
View:=acViewPreview, OpenArgs:="Artikel nach Kategorien 
und Lieferanten;KategorieID;LieferantID;Kategorie;
Lieferant"
```

Das Öffnungsargument enthält sogar mehrere Argumente, die durch Semikola voneinander getrennt sind. Das erste enthält den Text, der im Berichtskopf als Überschrift angezeigt werden soll, das zweite und dritte enthalten die Beschriftungen der Bezeichnungsfelder in den beiden Gruppenköpfen und das vierte und fünfte die Felder der Datensatzquelle, nach denen gruppiert werden soll.

Bericht mit zwei Gruppierungsebenen

Fehlt noch eine Routine, die diese Informationen auseinandernimmt und den entsprechenden Eigenschaften zuweist. Diese wird – wer hätte es gedacht – durch das **Beim Öffnen**-Ereignis des Berichts ausgelöst.

Diese Prozedur sortiert die Gruppierungen nach den Vorgaben im Öffnungsargument.

Die Prozedur **Report_Open** zerlegt zunächst die im Öffnungsargument übergebene Liste mit der **Split**-Funktion und speichert die einzelnen Elemente in einem Array namens **strGruppierungen** (s. Listing 1).

Warum so umständlich und gleich fünf Parameter mit **OpenArgs** übergeben? Die Reihenfolge der Gruppierungen wird doch in einem Formular festgelegt und da könnte man beim Öffnen des Berichts die Einstellungen des Formulars auslesen und entsprechend reagieren.

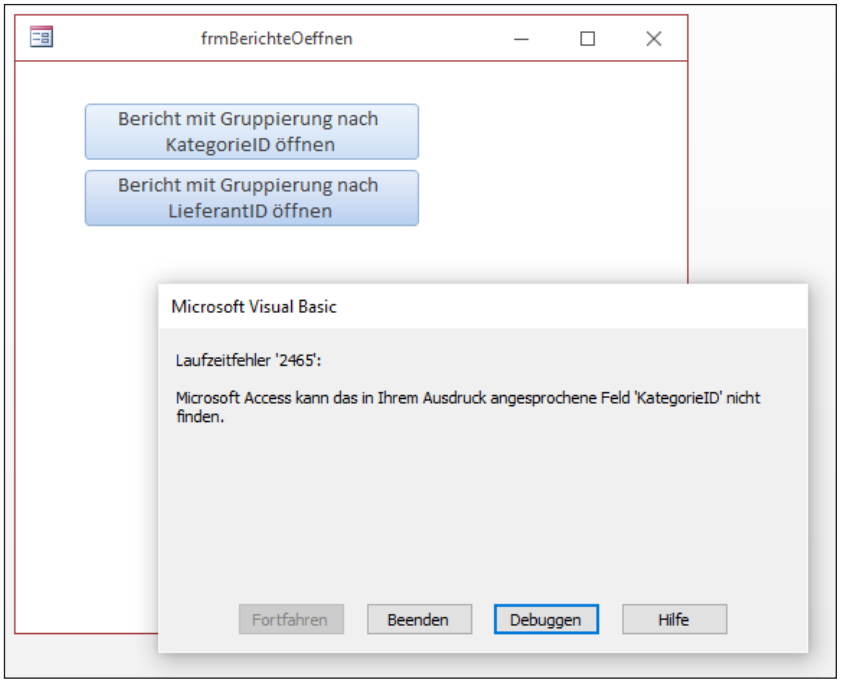

Bild 2: Formular zum Anzeigen von Berichten mit flexibler Gruppierungsreihenfolge

Dagegen spricht allerdings folgender Grundsatz: Je weniger Abhängigkeiten zwischen zwei Objekten bestehen, desto besser. Und weniger Abhängigkeit, als alle notwendigen Informationen beim Öffnen zu übergeben, ist fast nicht möglich.

Die folgende **For...Next**-Schleife wird genau zweimal durchlaufen – für jede Gruppierung einmal. Die erste Anweisung weist zunächst dem Beschriftungsfeld der äußeren Gruppierung die erste Gruppierungsbezeichnung zu – in diesem Fall **Kategorie**. Die zweite Anweisung legt das Feld fest, nach dem die äußere Gruppierung erfolgen soll – hier **KategorieID**.

Das gleiche Spiel wiederholt die Schleife noch für die innere Gruppierung – fertig! Mit dem obigen **DoCmd.OpenReport**-Befehl erhalten Sie eine nach Kategorien und Lieferanten gruppierte Artikelliste.

Lediglich die Angabe der aktuellen Kategorie beziehungsweise des aktuellen Lieferanten im Kopfbereich der Gruppierungen fehlt noch.

Dazu verwenden Sie die folgenden Prozeduren und erhalten gleichzeitig ein gutes Anwendungsbeispiel für den Einsatz des Ereignisses **Beim Formatieren**:

```
Private Sub Gruppenkopf1_Format(Cancel As Integer, _
        FormatCount As Integer)
    Me!txtUeberschriftGruppierung1 = _
        Me(Me.GroupLevel(1).ControlSource).Text
End Sub

Private Sub Gruppenkopf0_Format(Cancel As Integer, _
        FormatCount As Integer)
    Me!txtUeberschriftGruppierung0 = _
        Me(Me.GroupLevel(0).ControlSource).Text
End Sub
```

Nun erstellen wir ein kleines Formular mit zwei Schaltflächen, welche den Bericht einmal mit der Gruppierungsreihenfolge **LieferantID**-**KategorieID** und einmal mit der Reihenfolge **KategorieID**-**LieferantID** öffnet.

Das Formular sieht in Aktion so aus wie in Bild 2. Hier tritt leider ein häufiger Fehler auf: Wir haben dem Bericht nicht explizit die beiden Felder hinzugefügt, nach denen die angezeigten Datensätze gruppiert werden sollen.

Dies lässt sich leicht ändern: Fügen Sie die beiden Felder **KategorieID** und **LieferantID** aus der Feldliste zum Berichtsentwurf hinzu, und zwar an beliebiger Stelle im Detailbereich (die Position im Detailbereich ist entscheidend!). Sie können die Sichtbarkeit nämlich über die Eigenschaft **Sichtbar** auf **Nein** deaktivieren.

Einstellen der Überschriften in den Gruppierungsköpfen

Wie oben erwähnt, können Sie von der **Beim Formatieren**-Ereignisprozedur aus auf den Inhalt des ersten Detaildatensatzes der Gruppierung zugreifen, was hier sehr nützlich ist: Auf diese Weise lässt sich leicht der angezeigte Text der Felder **KategorieID** und **LieferantID** auslesen. Wenn Sie die Priorität der Gruppierung ändern möchten, verwenden Sie einfach die folgende Anweisung zum Anzeigen des Berichts:

```
DoCmd.OpenReport "rptGruppierungenTauschen", _
        View:=acViewPreview, OpenArgs:="Artikel nach _
    Lieferanten und Kategorien;LieferantID;KategorieID; _
                                Lieferant;Kategorie"
```

Wenn Sie den Bericht mit geänderter Gruppierung aufrufen möchten, müssen Sie ihn zuvor schließen. Ein Wechsel in die Entwurfsansicht reicht nicht aus.

```
Private Sub Report_Open(Cancel As Integer)
    Dim strOpenArgs As String
    Dim strGruppierungen() As String
    Dim i As Integer
    If IsNull(Me.OpenArgs) Then
        Exit Sub
    End If
    strGruppierungen() = Split(Me.OpenArgs, ";")
    Me!lblUeberschrift.Caption = strGruppierungen(0)
    For i = 1 To 2
        Me("lblUeberschriftGruppierung" & i - 1).Caption = strGruppierungen(i + 2)
        Me.GroupLevel(i - 1).ControlSource = strGruppierungen(i)
    Next i
End Sub
```

Listing 1: Auswertung der Öffnungsargumente zum Einstellen der Gruppierung

Berichtsereignisse: Bei Aktivierung/Deaktivierung

Neben den verschiedenen Bereichen und Steuerelementen liefern Berichte auch eine Reihe von Ereigniseigenschaften, für die Sie wie bei Formularen Ereignisprozeduren hinterlegen können. Zwei Ereignisse, die man üblicherweise nicht besonders oft benötigt, heißen »Bei Aktivierung« und »Bei Deaktivierung«. Wir zeigen, wie Sie diese einsetzen können: nämlich, um Elemente im Ribbon abhängig von diesen beiden Ereignissen zu aktivieren oder zu deaktivieren – zum Beispiel zum Ein- oder Ausschalten einer Drucken-Schaltfläche.

Bei Aktivierung und Bei Deaktivierung: Berichtsabhängige Funktionen ein- und ausschalten

Wenn Sie einen Bericht geöffnet haben und diesen dann schließen oder den Fokus auf ein anderes Objekt setzen, möchten Sie möglicherweise Elemente des Ribbons aktivieren oder deaktivieren. Eine **Drucken**-Schaltfläche macht beispielsweise am meisten Sinn, wenn gerade ein Bericht angezeigt wird. Was benötigen wir also, um das geplante Beispiel zum Aktivieren und Deaktivieren einer Ribbon-Schaltfläche synchron zum Auslösen der Ereignisse **Bei Aktivierung** und **Bei Deaktivierung** zu nutzen?

Richtig: Ein Ribbon, ein Objekt, welches die Ribbon-Definition referenziert, damit wir auf ihre Methoden zugreifen können, sowie ein paar Zeilen VBA-Code.

Das Ribbon fügen Sie hinzu, indem Sie die Tabelle **USysRibbons** erstellen und diese wie in Bild 1 mit der Definition des gewünschten Ribbons füllen. Damit Sie diese Tabelle nach der Erstellung sehen, müssen Sie in den Optionen des Navigationsbereichs die Option **Systemobjekte anzeigen** aktivieren.

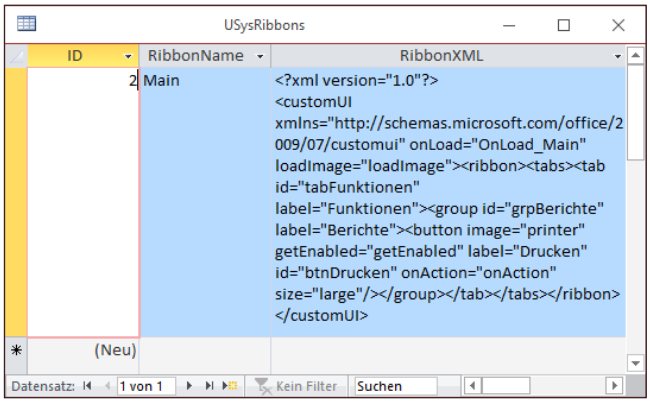

Bild 1: Tabelle mit der Ribbon-Definition

Die entsprechende Ribbon-Definition finden Sie in Listing 1. Die Elemente bilden die Struktur des Ribbons ab. Unter dem **customUI**-Element finden Sie das **tab**-Element, dann das **group**-Element und schließlich das **button**-Element als Schaltfläche. Wichtig sind in diesem Zusammenhang einige Attribute, die festlegen, welche VBA-Routinen beim Eintreten bestimmter Ereignisse auszulösen sind.

Das erste ist das Attribut **onLoad**, welches die folgende Prozedur auslöst, die Sie in ein Standardmodul namens **mdlRibbons** einfügen:

```
Sub onLoad_Main(ribbon As IRibbonUI)
    Set objRibbon_Main = ribbon
End Sub
```

Die Prozedur weist der Variablen **objRibbon_Main** einen Verweis auf diese Ribbon-Definition zu. Diese Variable deklarieren wir wie folgt im Kopf des gleichen Moduls:

```
Public objRibbon_Main As IRibbonUI
```

Die Klasse **IRibbonUI** ist in der Bibliothek **Microsoft Office x.0 Object Library** enthalten, weshalb Sie einen entsprechenden Verweis auf diese Klasse hinzufügen müssen (VBA-Editor, Menüpunkt **Extras|Verweise**, s. Bild 2).

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_Main" loadImage="loadImage">
    <ribbon>
        <tabs>
            <tab id="tabFunktionen" label="Funktionen">
                <group id="grpBerichte" label="Berichte">
                    <button image="printer" getEnabled="getEnabled" label="Drucken" id="btnDrucken"
                        onAction="onAction" size="large"/>
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

Listing 1: Ribbon-Definition für unsere Drucken-Schaltfläche

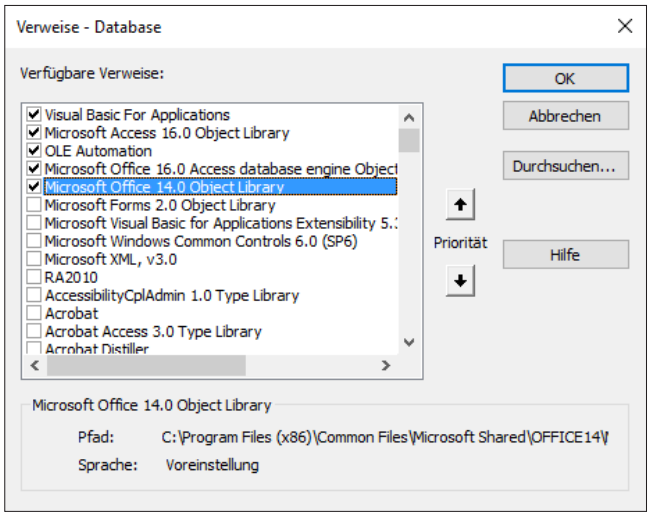

Bild 2: Verweis auf die Office-Bibliothek

Das zweite wichtige Attribut gehört zum **button**-Element und heißt **getEnabled**. Dieses Attribut wird immer ausgelöst, wenn das Ribbon eingeblendet oder mit einer speziellen Methode namens **Invalidate** aktualisiert wird. Für diese legen wir den Wert **getEnabled** an, also gleichlautend mit dem Attributnamen. So heißt dann auch die dadurch ausgelöste VBA-Funktion, die so aussieht:

```
Sub getEnabled(control As IRibbonControl, ByRef enabled)
    Dim obj As Object
    On Error Resume Next
    Set obj = Screen.ActiveReport
    On Error GoTo 0
    If Not obj Is Nothing Then
        enabled = True
    Else
        enabled = False
    End If
End Sub
```

Die Methode soll prüfen, ob es sich bei dem aktuell aktiven Objekt im Access-Fenster, also etwa einem Formular, einem Bericht oder einer Tabelle, um einen Bericht handelt. In diesem Fall soll das **button**-Element mit dem Namen **btnDrucken** aktiviert, anderenfalls deaktiviert werden. Dies prüfen wir, indem wir den Inhalt der Funktion **Screen.ActiveReport** in die Variable **obj** schreiben. Ist das aktuelle Objekt tatsächlich ein Bericht, erhält **obj** einen Verweis auf dieses Objekt. Anderenfalls löst dies einen Fehler aus, der aber nicht gemeldet wird, da wir die Fehlerbehandlung zuvor mit **On Error Resume Next** deaktiviert haben. Hat **obj** nach dieser Prüfung einen Wert, liegt ein Bericht als aktives Objekt vor und wir stellen den Wert des Rückgabeparameters **enabled** auf **true** ein. Anderenfalls erhält dieser Parameter den Wert **false**.

Auslösen der getEnabled-Funktion

Diese Funktion wird also erstmalig beim Einblenden dieses Teils der Ribbon-Definition ausgelöst, also wenn Sie auf das **tab**-Element namens **Funktionen** drücken.

Berichtsereignisse: Bei ohne Daten

Neben den verschiedenen Bereichen und Steuerelementen liefern Berichte auch eine Reihe von Ereigniseigenschaften, für die Sie wie bei Formularen Ereignisprozeduren hinterlegen können. Ein interessantes Ereignis heißt »Bei ohne Daten«. Es wird ausgelöst, wenn die Datenherkunft des Berichts keinerlei Datensätze beinhaltet. Ist dies der Fall, wird üblicherweise der Bericht ohne Daten geöffnet. Das möchten Sie aber vielleicht verhindern – und zwar mit Hilfe der hier vorgestellten Ereignisprozedur.

Bei Ohne Daten: Öffnen leerer Berichte vermeiden

Das Ereignis **Bei Ohne Daten** wird ausgelöst, wenn die Anzahl Datensätze in der Datensatzquelle des Berichts **0** ist. Damit ist die entsprechende Ereignisprozedur prädestiniert, Aufrufe der Seiten- und der Berichtsansicht zu unterbinden, wenn die Datenherkunft gar keine Daten bereitstellt.

Die Benutzer werden es Ihnen danken, wenn nicht hin und wieder Ausdrucke von eigentlich leeren Berichten vorkommen. Zum Simulieren eines leeren Berichts setzen Sie einfach eine **DoCmd.OpenReport**-Anweisung mit einem entsprechenden Kriterium ein:

```
Private Sub cmdBerichtOhneDatenOeffnen_Click()
    DoCmd.OpenReport "rptBerichtOhneDaten", _
        View:=acViewPreview, WhereCondition:="1=2"
End Sub
```

Der Aufruf führt zur Anzeige eines bis auf die Bezeichnungsfelder leeren Berichts (s. Bild 1). Das ist noch zu verschmerzen. Unangenehmer wird es, wenn diese »Blanko«-Variante des Berichts unnötig ausgedruckt wird.

Leeren Bericht verhindern

Den Druck eines leeren Berichts und die Anzeige einer leeren Vorschau können Sie abfangen, indem Sie eine Prozedur für das **Bei Ohne Daten**-Ereignis des Berichts anlegen.

Diese soll eine Meldung anzeigen und die Ausgabe abbrechen:

```
Private Sub Report_NoData(Cancel As Integer)
    MsgBox "Der Bericht enthält keine Daten", _
        vbExclamation Or vbOKOnly, "Keine Daten"
    Cancel = True
End Sub
```

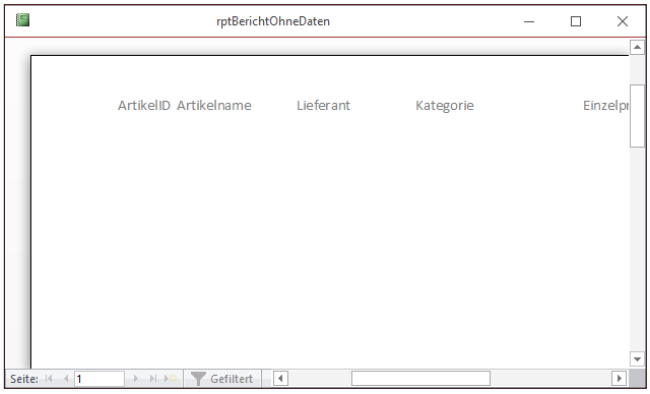

Bild 1: Ein Bericht ohne Daten zeigt nur die Spaltenüberschriften an.

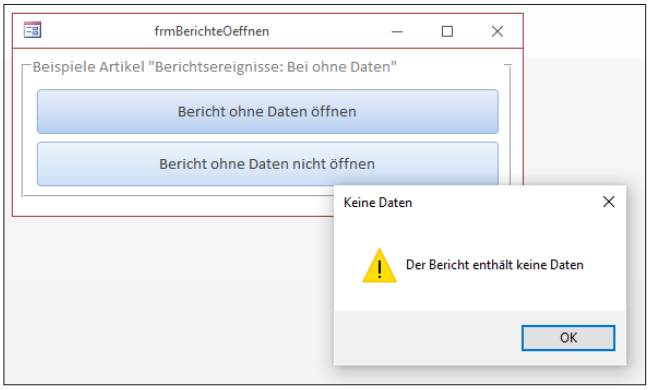

Bild 2: Ist der Bericht leer, erscheint diese Meldung.

```
Private Sub cmdBerichtOhneDatenNichtOeffnen_Click()
    Dim lngError As Long
    On Error Resume Next
    DoCmd.OpenReport "rptBerichtNichtOhneDaten", View:=acViewPreview, WhereCondition:="1=2"
    lngError = Err.Number
    On Error GoTo 0
    Select Case lngError
        Case 2501, 0
        Case Else
            MsgBox "Fehler in 'cmdBerichtOhneDatenNichtOeffnen_Click': " & lngError
    End Select
End Sub
```

Listing 1: Diese Prozedur liefert keinen Fehler, wenn das Öffnen eines Berichtes abgebrochen wird, weil keine Daten vorhanden sind.

Der erste Teil des Plans funktioniert sogar – der Bericht wird gar nicht erst geöffnet, stattdessen erscheint die gewünschte Meldung (s. Bild 2).

Unterbinden der Ausgabe eines leeren Berichts

Leider liefert ein abgebrochener Aufruf der **DoCmd.OpenReport**-Methode einen Laufzeitfehler (s. Bild 3). Diesen müssen Sie auch noch behandeln, indem Sie den Aufruf in eine geeignete Fehlerbehandlung einbetten.

Dazu fügen Sie der Prozedur, die den Bericht aufruft, eine Fehlerbehandlung wie in Listing 1 hinzu. Die Fehlerbehandlung wird vor dem Aufruf der Methode **DoCmd.OpenReport** deaktiviert. Dadurch wird kein Laufzeitfehler ausgelöst, wenn der Bericht wegen einer leeren Datenherkunft wieder geschlossen wird. Wir speichern die Fehlernummer in einer **Long**-Variablen namens **lngError** und analysieren diese anschließend. Im Falle der Werte **0** (kein Fehler) oder **2501** (Öffnen des Berichts abgebrochen) geschieht nichts, anderenfalls soll die Fehlermeldung für den aufgetretenen Fehler angezeigt werden.

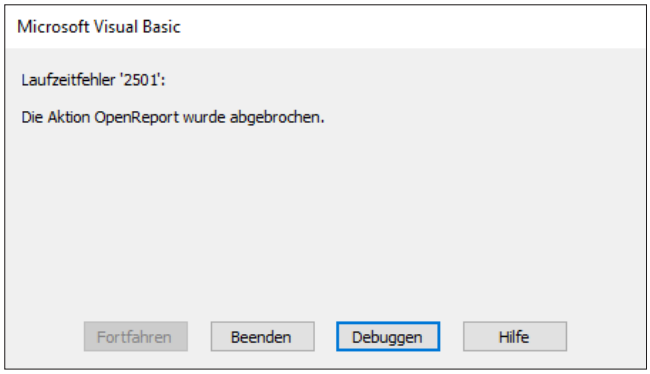

Bild 3: Fehlermeldung beim Abbruch des Öffnungsvorgangs eines Berichts

Besonderheiten

Das Ereignis wird übrigens nicht ausgelöst, wenn der Bericht gar nicht an eine Datenherkunft gebunden ist.

Außerdem ist wichtig, dass es nach den **Beim Formatieren**-Ereignissen des Berichts und seiner Bereiche auftritt, aber vor den **Beim Drucken**-Ereignissen.

Alternativen

Natürlich können Sie auch vor dem Öffnen des Berichts mit der **DoCmd.OpenReport**-Methode prüfen, ob die Datenherkunft überhaupt Daten enthält.

Bericht per PDF und Mail verschicken

Es fragen immer wieder Leser an, ob wir nicht einmal beschreiben können, wie man einen Bericht als PDF-Dokument speichert und diesen dann per E-Mail verschickt. Kein Problem: Wir schauen uns erst die einzelnen Grundtechniken an und gehen dann dazu über, Lösungen für Spezialfälle zu entwickeln.

Bericht als PDF speichern

Während wir in früheren Access-Versionen noch eine externe Software benötigten, um PDF-Dokumente auf Basis von Access-Berichten zu erstellen, liefert Access nun zum Glück ein Export-Format für diese Anforderung mit. Also ist ja alles in Butter – wir müssen nur noch herausfinden, wie wir den Bericht per Knopfdruck im PDF-Format speichern.

Beispielbericht

Dazu benötigen wir erstmal einen passenden Bericht. Wir wollen allerdings nicht mit einem einfachen Bericht beginnen, der einfach per Doppelklick auf den Berichtsnamen im Navigationsbereich aufgerufen werden kann, sondern mit etwas Anspruchsvollerem. Was könnte das sein? Natürlich ein Bericht, dem wir beim Öffnen per **WhereCondition** ein Kriterium übergeben, nach dem die anzuzeigenden Daten ausgegeben werden. Dies ist nämlich üblicherweise der Fall – es gibt wohl nur selten Berichte, die alle Datensätze der zugeordneten Datenherkunft ausgeben. Die meisten Berichte liefern gefilterte Daten, beispielsweise in einer Rechnung, die nur die Daten zu einer bestimmten Bestellung enthalten soll.

Unser Beispielbericht ist so aufgebaut, dass Sie mit ihm alle Datensätze der Tabelle **tblBestellungen** ausgeben können oder auch nur eine einzige Rechnung. Er ist wie in Bild 1 aufgebaut und enthält beispielsweise einen Seitenkopf, der nur für die Folgeseiten der ersten Seite der jeweiligen Rechnung ausgegeben werden soll. Dazu enthält der Bericht ein paar Ereignisprozeduren, auf deren Bedeutung wir weiter unten zu sprechen kommen (den vollständigen Bericht stellen wir im Beitrag **Rechnungsbericht** vor – siehe **www.access-im-unternehmen.de/1044**).

Bericht öffnen

Wenn Sie diesen Bericht einfach über den Navigationsbereich öffnen, zeigt er alle Rechnungen zu den Datensätzen der Tabelle **tblBestellungen** an (immerhin einige hundert). Das ist natürlich nicht das Ziel – wir wollen am Ende ge-

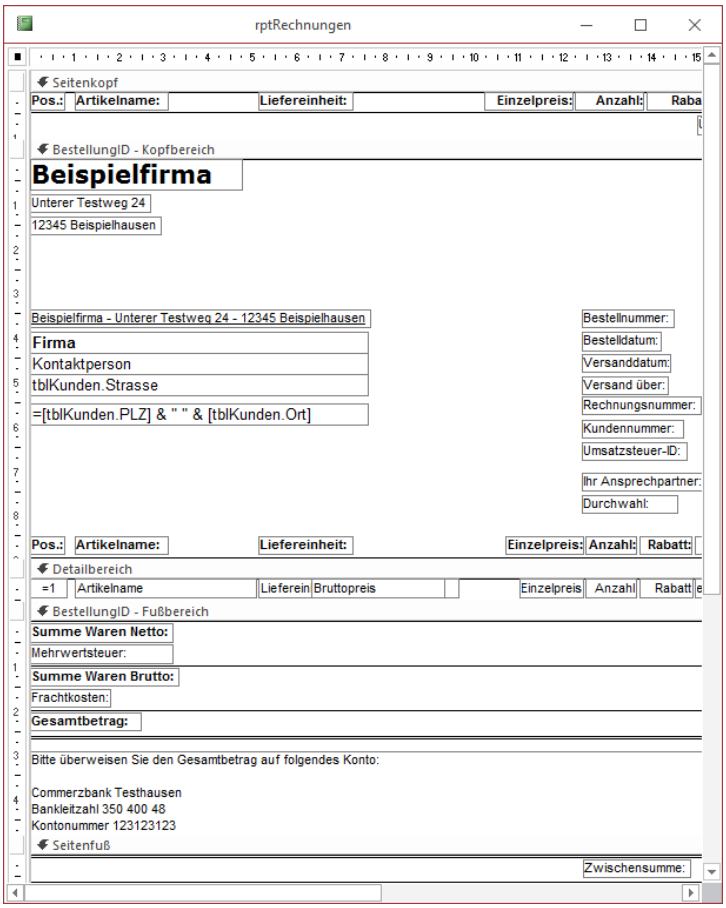

Bild 1: Dialog zum Festlegen von Sortierungen und Gruppierungen

nau einen Bericht öffnen, in ein PDF-Dokument speichern und dann per E-Mail verschicken.

Dazu bauen wir ein kleines Formular auf, das in der Entwurfsansicht wie in Bild 2 aussieht. Wenn wir nun auf eine der Schaltflächen klicken, soll immer genau die Rechnung für die aktuell angezeigte Bestellung ausgegeben werden.

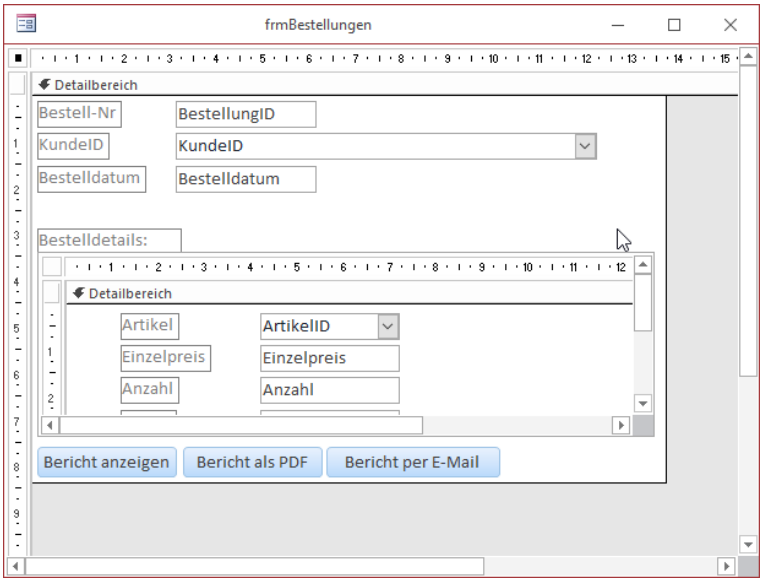

Bild 2: Formular zum Anzeigen einer Bestellung und zum Ausgeben von Rechnungen

Wenn wir die Schaltfläche **cmdBerichtAnzeigen** anklicken, soll der Bericht einfach nur in der Seitenansicht geöffnet werden. Dazu verwenden wir eine Ereignisprozedur, die den Bericht mit der Methode **DoCmd.OpenReport** öffnet und dabei mit dem Parameter **WhereCondition** den Ausdruck **"BestellungID = " & Me!BestellungID** übergibt. Dies gelingt auch, wie Bild 3 zeigt: Der Bericht enthält lediglich eine Seite mit den im Formular zu dieser Bestellung angezeigten Rechnungspositionen. Die Ereignisprozedur sieht wie folgt aus:

```
Private Sub cmdBerichtAnzeigen_Click()
    DoCmd.OpenReport "rptRechnungen", View:=acViewPreview, ↵
        WhereCondition:="BestellungID = " & Me!BestellungID
End Sub
```

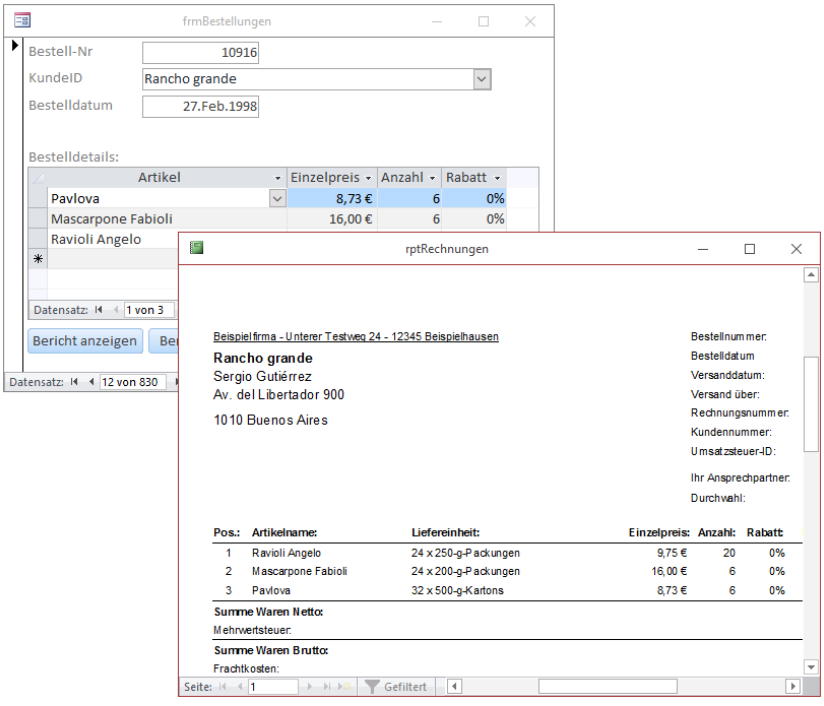

Bild 3: Formularansicht zum Anzeigen einer Bestellung und zum Ausgeben von Rechnungen

Soweit, so gut: Aber wir wollen ja nicht die Berichtsvorschau, sondern ein PDF-Dokument mit dem gleichen Inhalt. Bei geöffnetem Bericht kommen wir relativ einfach dorthin: Wir müssen nur die **Drucken**-Schaltfläche des Ribbons betätigen und im nun erscheinenden **Drucken**-Dialog den Eintrag **Microsoft Print To PDF** auswählen (s. Bild 4). Nach einem Klick auf **OK** erscheint ein **Dateiauswahl**-Dialog, mit dem Sie festlegen, unter welchem Dateinamen die PDF-Datei gespeichert werden soll. Danach können Sie die PDF-Datei öffnen, die wie in Bild 5 aussieht.

PDF per VBA erstellen

Das ist allerdings nicht unser Ziel, wenn wir wollen ja nicht noch auf weitere

Schaltflächen klicken geschweige denn Dateinamen für die zu erstellende PDF-Datei vergeben – und per Mail verschickt ist das PDF-Dokument nun noch längst nicht. Also wenden wir uns der folgenden Stufe zu – der Schaltfläche **cmdBerichtAlsPDF**. Die dadurch ausgelöste Prozedur soll den Bericht möglichst automatisch in ein PDF-Dokument schreiben. Als Erstes versuchen wir, bei geöffnetem Bericht die folgende Methode aufzurufen:

```
RunCommand acCmdPrint
```

Dies hilft leider nicht weiter, denn damit öffnen Sie nur den bereits bekannten **Drucken**-Dialog. Hilfe erhalten wir erst durch die Methode **DoCmd.OutputTo**. Dieser können wir folgende Parameter übergeben:

- **ObjectType**: **acOutputReport** für einen Bericht
- **ObjectName**: Name des Berichts in Anführungszeichen, hier **"rptRechnungen"**

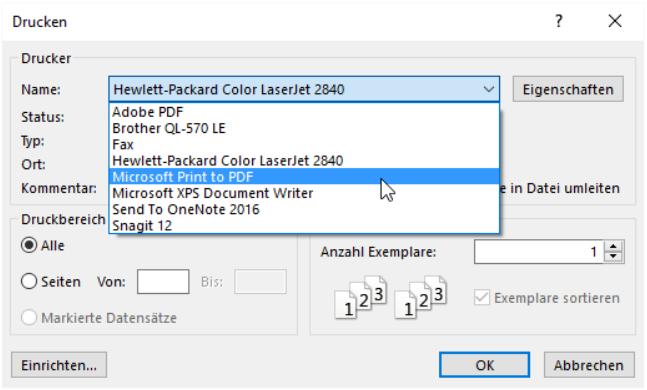

Bild 4: Drucken-Dialog mit dem Microsoft-eigenen PDF-Treiber

- **OutputFormat**: **acFormatPDF** für PDF
- **OutputFile**: Dateiname der zu erstellenden Datei inklusive Verzeichnis

Die übrigen Parameter spielen für unseren Anwendungsfall keine Rolle. Die Methode liefert allerdings keine Möglichkeit, ein Kriterium für die Festlegung des zu druckenden Rechnungsdatensatzes zu definieren. Wie also wollen wir dafür sorgen, dass nur der gewünschte Datensatz berücksichtigt wird? Die einzige Möglichkeit ist, den Bericht tatsächlich zuvor in der Seitenansicht zu öffnen, dann die **OutputTo**-Methode aufzurufen und den Bericht dann wieder zu schließen (s. Listing 1).

Hier öffnen wir den Bericht mit der bereits zuvor verwendeten Methode. Dann speichern wir den Dateinamen in der Variablen **strDateiname** und rufen die **DoCmd.OutputTo**-Methode auf – unter anderem mit dem Dateinamen als Parameterwert.

Nach dem Schließen des Berichts verwenden wir die API-Funktion **ShellExecute**, um den Bericht direkt im PDF-Reader anzuzeigen.

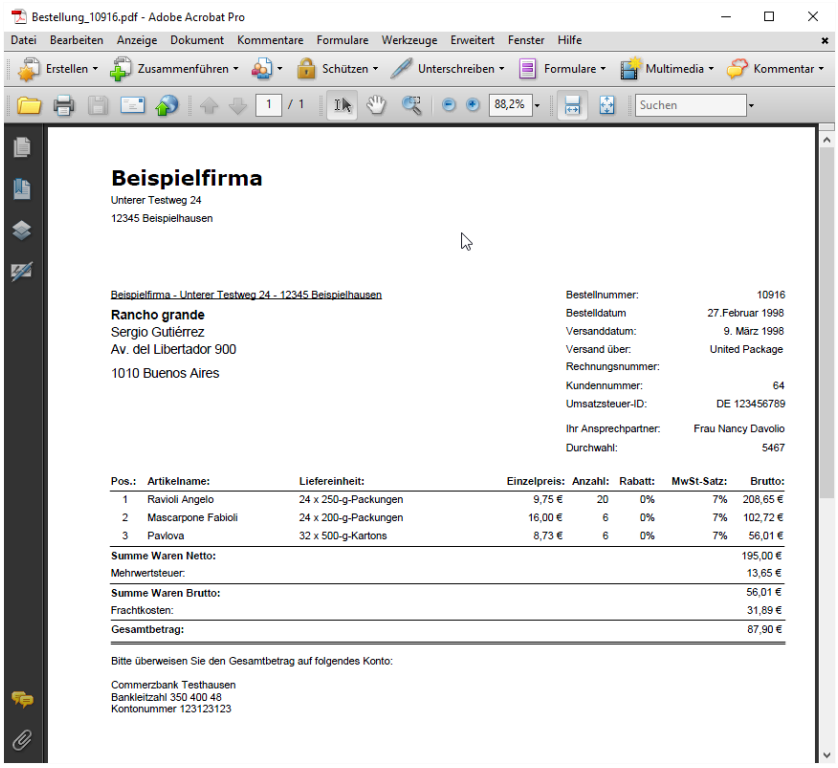

Bild 5: Ein Bericht im PDF-Format

API-Praxis: Zugriff auf die Registry

Windows stellt einige API-Prozeduren zum Zugriff auf die Registry zur Verfügung. Die Registry verwaltet viele der für das System wichtigen Informationen. Hierzu gehören Informationen über das System, über die verschiedenen Anwendungen und Dateitypen und wie sie verknüpft werden und vieles mehr. Dieser Beitrag zeigt, wie Sie per VBA über die Funktionen der Windows-API auf die Registry zugreifen.

Neben den Informationen über das System finden Sie auch Informationen zu den Anwendungen selbst in der Registry. Wenn Sie zum Beispiel in den Optionen von Access den Standarddatenbankpfad ändern, wird der neue Pfad in die Registry eingetragen.

Auch wenn Sie der Datenbank einen neuen Assistenten oder ein Add-In hinzufügen, schreibt der Add-In-Manager die benötigten Informationen in die Registry. Für den Entwickler wird die Registry interessant, wenn er bestimmte Daten speichern will, die nicht in einer Tabelle untergebracht werden können oder sollen.

Die Registry ist im Prinzip wie das Dateisystem von Windows aufgebaut. Die Verzeichnisstruktur ist nahezu identisch. Statt der Dateien enthält die Registry Variablen. Wie die Dateien haben auch die Variablen einen bestimmten Inhalt.

Die Windows-API stellt einige Befehle zur Bearbeitung der Registry zur Verfügung. Im Folgenden lernen Sie Befehle, die Sie zum Hinzufügen, Bearbeiten und Entfernen von Schlüsseln benötigen, anhand eines Beispiels kennen.

Aufbau der Registry-Einträge

Die Einträge in die Registry, die Sie über den Befehl **RegEdit** öffnen (beispielsweise über das Suchen-Feld von Windows 10), sind nach einem bestimmten Schema aufgebaut. Auf der obersten Ebene befinden sich die Hauptpfade. Es gibt insgesamt fünf Hauptpfade, die Sie auch in Bild 1 sehen können:

- **HKEY_CLASSES_ROOT**: Speichert Informationen über die unterschiedlichen Dateitypen und die dazugehörenden Anwendungen.
- **HKEY_CURRENT_USER**: Speichert die Einstellungen des aktuellen Benutzers.
- **HKEY_LOCAL_MACHINE**: Speichert Informationen über die Hard- und Software des Systems.
- **HKEY_USERS**: Speichert die voreingestellten Einstellungen bezüglich Hard- und Software sowie benutzerdefinierte Änderungen.

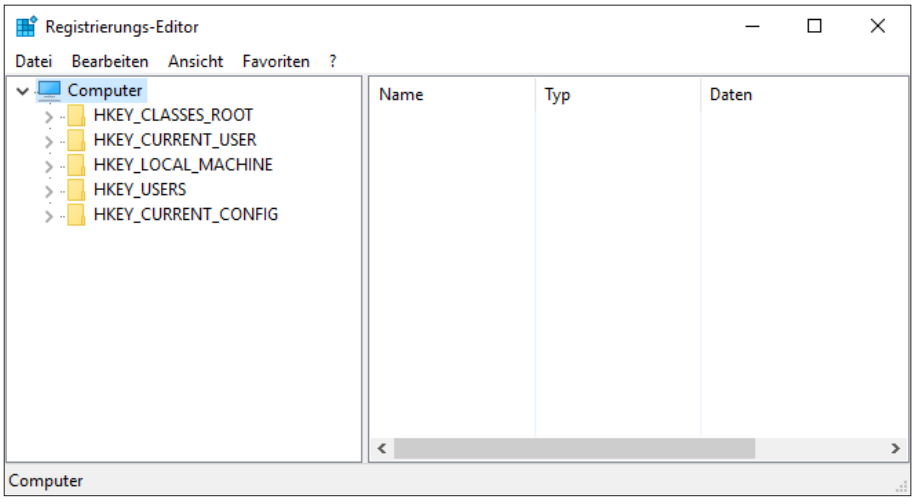

Bild 1: Die Registry mit den fünf Hauptzweigen

- **HKEY_CURRENT_CONFIG**: Speichert allgemeine Informationen über die Systemkonfiguration.

Jeder dieser Hauptpfade enthält weitere, aus unterschiedlich vielen Ebenen bestehende Unterpfade. Der unterste Pfad jeder der Ebenen enthält schließlich die eigentlichen Schlüssel. Die Schlüssel bestehen aus einem Namen und einem Wert. Mit dem Registrierungseditor, mit dem Sie die Einträge der Registry bequem einsehen, editieren und löschen sowie neue Einträge hinzufügen können, greifen Sie über die Benutzeroberfläche auf diese Schlüssel zu.

Der in Bild 2 markierte Pfad lautet beispielsweise **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\Access\Menu Add-Ins\RibbonAdmin2016**.

In dem Pfad finden Sie vier Schlüssel, die einige Informationen zu einem Access-Add-In beinhalten – unter anderem die Funktion, die beim Aufruf des Add-Ins gestartet werden soll, sowie der Ort der Bibliothek, in der die benötigte Funktion enthalten ist.

Praxisbeispiel: Shareware mit zeitlicher Begrenzung

Wenn Sie eine Datenbankanwendung als Shareware vertreiben, möchten Sie die Benutzung der Anwendung möglicherweise auf eine Testphase von beispielsweise 30 Tagen beschränken. Die Anwendung muss also bei jedem Start zunächst überprüfen, ob der angegebene Zeitraum schon beendet ist oder nicht. Dazu müssen Sie zunächst eine Information über das Installationsdatum oder das Datum der ersten Anwendung speichern.

Das Speichern der Daten innerhalb der Datenbank ist deshalb nicht zu empfehlen, weil der Anwender die Datenbank nach Ablauf der 30 Tage einfach neu installieren könnte. Eine weitere Möglichkeit ist das Speichern des Datums in der Registry. Wenn der Benutzer die Anwendung startet, vergleicht die Anwendung das in die Registry

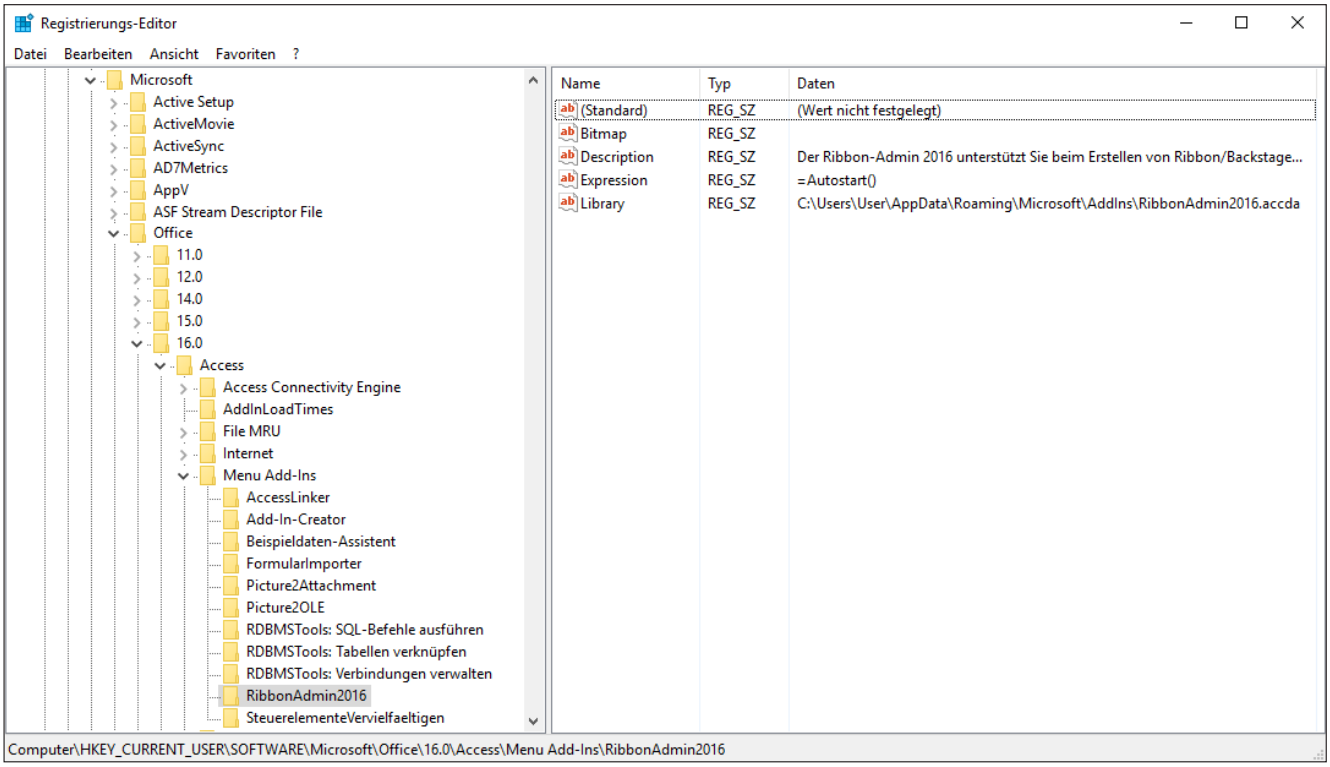

Bild 2: Die Access-Add-Ins und ihre Registrierung in der Windows-Registry

eingetragene Datum mit dem aktuellen Datum und soll den Benutzer gegebenenfalls auf das Ablaufen der Testphase hinweisen. Wenn der Benutzer dann die Datenbank neu installiert, findet die Datenbank den noch vorhandenen Registry-Eintrag und erkennt, dass die Testphase bereits einmal abgelaufen ist.

Zunächst einmal überlegt man sich, wo in der Registry ein solcher Wert am besten abgelegt wird. Da die Anwendung vermutlich unabhängig vom Benutzer auf dem Rechner installiert, wählt man den Hauptpfad **HKEY_CURRENT_USER** aus. Der Name des Pfades soll **\AMVShop\Beispiele\Registry** lauten. Den Schlüssel nennen Sie Installationsdatum und der Wert soll das aktuelle Datum sein.

Selbstverständlich sollten Sie die Informationen etwas besser verstecken und gegebenenfalls auch verschlüsseln, wenn Sie eine Anwendung wirksam schützen wollen. Da der vorliegende Anwendungsfall jedoch nur als Beispiel dienen soll, wird hier mit offenen Karten gespielt.

Konstanten

Im Rahmen der nachfolgend beschriebenen API-Funktionen definieren Sie einige Konstanten. Die Konstanten benötigen Sie, um innerhalb des VBA-Code statt nichtssagender Werte für den Entwickler verständliche Begriffe verwenden zu können. So definiert man beispielsweise die folgende Konstante **HKEY_CLASSES_ROOT**, um sie bei der Angabe eines Registryschlüssels statt des Ausdrucks **&H80000003** zu verwenden:

```
Public Const HKEY_CLASSES_ROOT As Long = &H80000000
```

Damit Sie nicht zwischendurch immer wieder neue Konstanten definieren müssen, geben Sie direkt alle in Zusammenhang mit den Registryzugriffen benötigten Konstanten ein:

```
Public Const HKEY_CURRENT_USER    As Long = &H80000001
Public Const HKEY_LOCAL_MACHINE   As Long = &H80000002
Public Const HKEY_USERS           As Long = &H80000003
Public Const HKEY_CURRENT_CONFIG  As Long = &H80000005
Public Const REG_STR              As Long = 1
Public Const REG_OPTION_NON_VOLATILE  As Long = 0
Public Const REG_SZ               As Long = 1
Public Const KEY_ALL_ACCESS       As Long = &H3F
Public Const StringlängeMax       As Long = 200
Public Const ERROR_SUCCESS        As Long = 0&
Public Const ERROR_NO_MORE_ITEMS  As Long = 259&
```

Noch praktischer ist es, wenn Sie für die fünf Konstanten für die Hauptpfade der Registry gleich eine Enumeration anlegen. Dann können Sie später, wenn Sie in Funktionen eine dieser Konstanten als Parameter übergeben wollen, ganz einfach per Intellisense auf diese Konstanten zugreifen. Dies sieht dann so aus:

```
Public Enum eRegistryPfade
    HKEY_CLASSES_ROOT = &H80000000
    HKEY_CURRENT_USER = &H80000001
    HKEY_LOCAL_MACHINE = &H80000002
    HKEY_USERS = &H80000003
    HKEY_CURRENT_CONFIG = &H80000005
End Enum
```

Sie müssen die entsprechenden Konstanten dann allerdings löschen oder umbenennen, da sonst mehrere Elemente gleichen Namens im Code vorkommen, was zu Kompilierfehlern führt.

Achtung: Sie werden im Folgenden neben den Konstanten auch noch API-Funktionen deklarieren sowie VBA-Funktionen und -Prozeduren eingeben. Am besten legen Sie für die Konstanten und die API-Funktionen ein eigenes Modul an (zum Beispiel **mdlAPIRegistry**). Die VBA-Funktionen und -Prozeduren zum Thema Registry sollten Sie ebenso in einem eigenen Modul zusammenfassen – beispielsweise **mdlRegistry**.

Anlegen eines Schlüssels

Die API-Funktion **RegCreateKeyEx** dient zum Anlegen eines Registrierungsschlüssels. Falls der anzulegende

```
Public Function SchluesselErstellen(strHauptpfad As eRegistryPfade, strPfad As String) As Long
    Dim lngHandle As Long
    Dim secAttrib As SECURITY_ATTRIBUTES
    RegCreateKeyEx strHauptpfad, strPfad, 0&, "", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, secAttrib, lngHandle, 0&
    SchluesselErstellen = lngHandle
End Function
```

Listing 1: Hilfsfunktion zum Erstellen eines Schlüssels in der Registry

Schlüssel bereits vorhanden ist, öffnet die Funktion den Schlüssel. Die Deklaration der Funktion lautet:

```
Declare Function RegCreateKeyEx _
    Lib "advapi32.dll" _
    Alias "RegCreateKeyExA" (_
    ByVal hKey As Long, _
    ByVal lpSubKey As String, _
    ByVal Reserved As Long, _
    ByVal lpClass As String, _
    ByVal dwOptions As Long, _
    ByVal samDesired As Long, _
    lpSecurityAttributes As SECURITY_ATTRIBUTES, _
    phkResult As Long, _
    lpdwDisposition As Long
) As Long
```

Interessant sind vor allem die Parameter **hKey** und **lpSubKey**. Für den Parameter **hKey** geben Sie entweder das Handle an, das auf den entsprechenden Eintrag in der Registry zeigt, oder Sie geben einen der Hauptpfade aus der obigen Liste ein. Für den Parameter **lpSubKey** geben Sie den Pfad des Registry-Wertes an, den Sie öffnen möchten.

Für den Parameter **lpSecurityAttributes** müssen Sie noch einen **Type** definieren, und zwar im Modul **mdlAPIRegistry**:

```
Private Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Boolean
End Type
```

Um den Aufruf der API-Funktion ein wenig komfortabler zu gestalten, erstellen Sie eine Funktion, an die Sie einfach nur den Hauptpfad und den Pfad des Registry-Eintrages übergeben müssen:

```
Public Function SchluesselErstellen(lngHauptpfad As Long, _
        Pfad As String) As Long
...
```

Als Hauptpfad würden Sie nun eine der Konstanten wie etwa **HKEY_LOCAL_MACHINE** eingeben. Da wir dafür aber eine Enumeration angelegt haben, können Sie diese auch als Datentyp für **lngHauptpfad** angeben, sodass die Funktion nun wie in Listing 1 aussieht.

Sie können die Funktion zum Beispiel über das Testfenster aufrufen.

Geben Sie dort den folgenden Befehl ein, erhalten Sie Unterstützung bei der Auswahl des Hauptpfades (s. Bild 3):

```
MsgBox SchlüsselErstellen(HKEY_CURRENT_USER, "SOFTWARE\
Haufe\Beispiele\Test")
```

Das Meldungsfenster zeigt daraufhin die Nummer des Handles an, mit dem Sie auf den Eintrag in die Registry

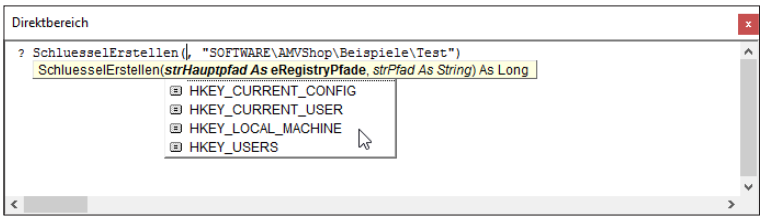

Bild 3: Aufruf der Funktion **SchluesselErstellen** mit IntelliSense-Unterstützung

zugreifen können, ohne den kompletten Pfad eingeben zu müssen – zum Beispiel **4576**.

Außerdem erstellt die Funktion den angegebenen Pfad in der Registry, wenn er nicht bereits vorhanden ist. Wenn der Pfad bereits erstellt wurde, bereitet die Funktion den Pfad auf den Zugriff vor. Mit dem Registrierungseditor können Sie schnell überprüfen, ob die Funktion den Registrierungsschlüssel wie gewünscht angelegt hat. Öffnen Sie den Editor und navigieren Sie zu dem entsprechenden Pfad (s. Bild 4).

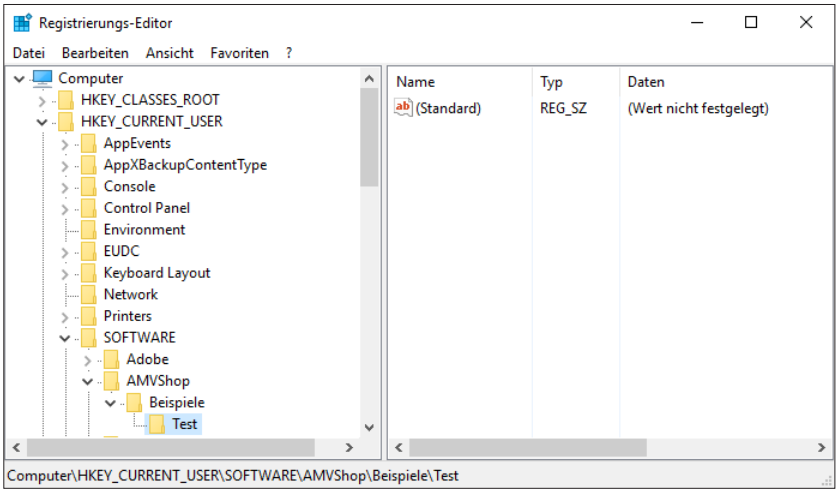

Bild 4: Der neue Eintrag in der Windows-Registry

Öffnen eines Schlüssels

Wenn Sie nur auf einen Schlüssel zugreifen möchten, verwenden Sie den API-Befehl **RegOpenKey**. Die Funktion hat die folgende Syntax:

```
Declare Function RegOpenKeyEx _
    Lib "advapi32.dll" _
    Alias "RegOpenKeyExA" ( _
    ByVal hKey As Long, _
    ByVal lpSubKey As String, _
    ByVal ulOptions As Long, _
    ByVal samDesired As Long, _
    phkResult As Long) _
As Long
```

Auch den Aufruf dieser API-Funktion können Sie ein wenig komfortabler gestalten, indem Sie eine Funktion erstellen, die nur die wichtigsten Werte als Parameter erfordert:

Die Funktion **SchluesselOeffnen** aus Listing 2 hat nur einen Unterschied im Vergleich zu der vorherigen Funktion: Sie erstellt keinen neuen Schlüssel, falls der angegebene Schlüssel nicht vorhanden ist. Beachten Sie, dass die API-Funktion keinen Fehler auslöst, wenn der zu öffnende Schlüssel nicht vorhanden ist. Sie gibt lediglich für das Handle den Wert **0** zurück.

Ein Beispielaufruf sieht wie folgt aus:

```
? SchluesselOeffnen(HKEY_CURRENT_USER, "SOFTWARE\AMVShop\
Beispiele\Test")
 3816
```

Entfernen eines Schlüssels

Bevor Sie beginnen, den neuen Schlüssel mit Werten zu füllen, erfahren Sie noch, wie Sie einen Schlüssel per API-Aufruf wieder entfernen können. Auch das ist wichtig, denn wenn Sie mit der Installation der Software in die Registry eingreifen, sollten Sie die Änderung bei einer

```
Public Function SchluesselOeffnen(lngHauptpfad As eRegistryPfade, strPfad As String) As Long
    Dim lngHandle As Long
    RegOpenKeyEx lngHauptpfad, strPfad, 0, KEY_ALL_ACCESS, lngHandle
    SchluesselOeffnen = lngHandle
End Function
```

Listing 2: Hilfsfunktion zum Ermitteln des Handles eines Registry-Eintrags

```
Public Function SchluesselLoeschen(lngHauptpfad As eRegistryPfade, strPfad As String) As Long
    Dim lngHandle As Long
    lngHandle = RegDeleteKey(lngHauptpfad, strPfad)
    SchluesselLoeschen = lngHandle
End Function
```

Listing 3: Hilfsfunktion zum Löschen eines Eintrags

Deinstallation auch wieder rückgängig machen. Zum Löschen eines Pfades in der Registry verwendet man den API-Befehl **RegDeleteKey**, den Sie wie folgt deklarieren:

```
Declare Function RegDeleteKey _
    Lib "advapi32.dll" _
    Alias "RegDeleteKeyA" ( _
    ByVal hKey As Long, _
    ByVal lpSubKey As String _
) As Long
```

Die Funktion erwartet nur den Hauptpfad und den Pfad des zu löschenden Schlüssels. Die Funktion gibt den Wert **0** zurück – es sei denn, die Operation schlägt fehl, zum Beispiel, weil der angegebene Pfad nicht vorhanden ist. Wichtig ist, dass Sie den kompletten zu entfernenden Pfad bis in die unterste Ebene angeben.

Auch für diese API-Funktion haben wir wieder eine Wrapper-Funktion programmiert, die Sie in Listing 3 finden.

Wenn Sie nun den gleichen Pfad wieder löschen wollen, den Sie soeben angelegt haben, gelingt dies nicht mit dem folgenden Aufruf der Funktion:

```
? SchluesselLoeschen(HKEY_CURRENT_USER, "SOFTWARE\AMVShop\
Beispiele\Test")
0
```

Dies liefert mit dem Wert **0** zwar eine Erfolgsmeldung, allerdings wird hier logischerweise nur genau das angegebene Element gelöscht, nämlich das Verzeichnis **Test**.

Wenn Sie alle Verzeichnisse löschen wollen, die Sie zuvor angelegt haben, sind noch zwei weitere Aufrufe nötig – Sie müssen nämlich jedes Verzeichnis vom hintersten an einzeln löschen:

```
? SchluesselLoeschen(HKEY_CURRENT_USER, "SOFTWARE\AMVShop\
Beispiele")
0
? SchluesselLoeschen(HKEY_CURRENT_USER, "SOFTWARE\AMV-
Shop")
0
```

Wenn Sie direkt versuchen, den Pfad **SOFTWARE\AMV-Shop** zu löschen, lösen Sie einen Fehler aus beziehungsweise erhalten einen anderen Wert als **0** zurück, in diesem Fall den Wert **5**. Wenn Sie einen Pfad löschen wollen, der nicht mehr vorhanden ist, liefert die Funktion den Wert **2**.

Schließen eines Schlüssels

Mit der API-Anweisung **RegCloseKey** schließen Sie einen geöffneten Schlüssel wieder und geben die dabei gebundenen Ressourcen wieder frei.

Der einzige Parameter der Funktion ist das Handle des zu schließenden Schlüssels, das Sie beim Öffnen mit **RegOpenKey** ermittelt haben. Die Deklaration lautet folgendermaßen:

```
Declare Function RegCloseKey _
    Lib "advapi32.dll" ( _
    ByVal hKey As Long _
) As Long
```

Wenn auch eigentlich nicht nötig, da ohnehin nur ein Parameter übergeben wird, haben wir dennoch auch hier eine Wrapper-Funktion erstellt. Sie liefert den Wert **0** zurück, wenn das Schließen gelungen ist, sonst den Wert **6**:

```
Public Function SchluesselSchliessen(lngHandle As Long) As Long
    SchluesselSchliessen = RegCloseKey(lngHandle)
End Function
```

Setzen eines Schlüsselwertes

Nachdem Sie nun wissen, wie Sie Registrypfade anlegen, öffnen, schließen und wieder entfernen, lernen Sie jetzt, wie Sie mit der Funktion **RegSetValueEx** einen Schlüssel hinzufügen und einen Wert dafür angeben. Deklarieren Sie die Funktion **RegSetValueEx** folgendermaßen:

```
Declare Function RegSetValueEx _
    Lib "advapi32.dll" _
    Alias "RegSetValueExA" ( _
    ByVal hKey As Long, _
    ByVal lpValueName As String, _
    ByVal Reserved As Long, _
    ByVal dwType As Long, _
    lpData As Any, _
    ByVal cbData As Long _
) As Long
```

Wenn Sie die Deklaration aufmerksam betrachten, dass der Parameter **lpData** als einziger nicht **ByVal**, sondern **ByRef** übergeben wird und dass als Datentyp **Any** angegeben ist. Dahinter verbergen sich einige Besonderheiten.

Wie bereits weiter oben erwähnt, gibt es in C im Gegensatz zu Visual Basic Parameter, die unterschiedlichen Datentyps sein können. Hier tritt dieser Fall zu Tage: Auch die Schlüsselwerte in der Registry können unterschiedlichen Datentyps sein, zum Beispiel **String** oder **Long**. Unmittelbar vor dem Parameter **lpData** finden Sie aber den Parameter **dwType**. Mit ihm können Sie der API-Funktion mitteilen, welchen Datentyps der Parameter ist. Im Falle der Registry können Sie beispielsweise Zeichenketten (**String**) oder Variablen des Typs **Long** verwenden.

Falls Sie eine Zeichenkette verwenden, heißt die entsprechende Konstante **REG_SZ**. Für Variablen des Datentyps **Long** verwenden Sie die Konstante **REG_DWORD**. Die Konstanten dienen der besseren Handhabung für den menschlichen Benutzer, der mit ihnen sicher mehr anfangen kann als mit den Zahlenwerten, die sich hinter den Konstanten verbergen. Natürlich müssen Sie diese und andere Konstanten deklarieren:

```
Public Const REG_OPTION_NON_VOLATILE As Long = 0
Public Const REG_SZ As Long = 1
Public Const
Public Const KEY_ALL_ACCESS As Long = &H3F
Public Const StringlaengeMax As Long = 200
Public Const ERROR_SUCCESS As Long = 0&
Public Const ERROR_NO_MORE_ITEMS As Long = 259&
```

Die API-Funktion verwendet einige Parameter, die von vornherein festgelegt beziehungsweise bei der Ausführung ermittelt werden. Die Funktion aus Listing 4 vereinfacht den Aufruf der Funktion **RegSetValueEx**.

Leider gelingt das Eintragen des Wertes noch nicht wie gewünscht, denn dieser erscheint in einem falschen Zei-

```
Public Function WertSetzenString(lngHauptpfad As eRegistryPfade, strPfad As String, strSchluesselname As String, _
        strSchluesselwert As String) As Long
    Dim lngHandle As Long
    Dim lngLaenge As Long
    lngLaenge = Len(strSchluesselwert)
    strSchluesselwert = strSchluesselwert & Chr$(0)
    lngHandle = SchluesselOeffnen(lngHauptpfad, strPfad)
    WertSetzenString = RegSetValueEx(lngHandle, strSchluesselname, 0&, REG_SZ, strSchluesselwert, lngLaenge)
End Function
```

Listing 4: Hilfsfunktion zum Eintragen eines Wertes in die Registry

Ticketsystem mit Access, Teil 2

Im ersten Teil der Beitragsreihe haben wir uns darum gekümmert, die E-Mails mit Kundenanfragen zu erfassen und diese in einem Formular in neue Tickets umzuwandeln. Außerdem haben wir in diesem Zuge auch gleich die Kundendaten erfasst. Nun schauen wir uns an, wie wir die Tickets bearbeiten, Antworten an die Kunden senden und deren Antworten automatisch zum Ticket hinzufügen. Außerdem stehen noch einige Feinheiten an, die uns die Arbeit mit den Tickets erleichtern sollen. Schließlich sollen die Tickets auf einfachste Weise von Outlook in unsere Anwendung gelangen.

Bin ich im richtigen Frontend?

Bevor wir zu den eigentlichen Funktionen der Ticketverwaltung übergehen, wollen wir noch eine Sache sicherstellen, die mir beim Vorbereiten der Anwendung für den zweiten Teil dieser Beitragsreihe kurz Kopfzerbrechen bereitet hat. In der vorherigen Ausgabe von Access im Unternehmen haben wir im Beitrag **Benutzerdefinierte Outlook-Eigenschaften** (**www.access-im-unternehmen.de/1030**) erfahren, wie wir in Outlook benutzerdefinierte Eigenschaften speichern können.

Genau genommen war das eine Vorarbeit für die Ticketverwaltung, denn wir wollen dort in Outlook speichern, in welche Datenbank die Kundenanfragen, die dort als E-Mail eingehen, gespeichert werden sollen. Dabei handelt es sich natürlich um unser Ticketsystem.

Wenn wir dort den Pfad zur entsprechenden Datenbankdatei speichern, soll Outlook eingehende Kundenanfragen als E-Mails in dieser Datenbank speichern. Das gelingt genau so lange, bis die Datenbank nicht mehr am angegebenen Speicherort vorgebunden werden kann.

Es kann aber auch der Fall eintreten, dass Sie – aus welchen Gründen auch immer – die Datenbank mit dem Ticketsystem nicht von einem Verzeichnis in ein anders verschieben, sondern es lediglich kopieren. Bei mir war dies der Fall, weil ich die Datenbank von dem Ordner, in dem die Daten für den ersten Teil der Beitragsreihe lagen, in den Ordner mit den Daten für den zweiten Teil kopiert habe. Dadurch lag die Zieldatenbank für die neu eingehenden Kundenanfragen zwar nun in einem neuen Verzeichnis, gleichzeitig aber auch noch im alten Verzeichnis!

Hätte ich die Datenbank einfach verschoben und somit aus dem alten Ordner entfernt, wäre dies beim Öffnen von Outlook durch den dann ausgelösten VBA-Code aufgefallen und ich hätte einfach den Pfad zum neuen Speicherort eingeben müssen.

Nun habe ich aber fleißig neue Kundenanfragen in den dafür angelegten Outlook-Ordner kopiert, in der Hoffnung, diese würden so auch automatisch in der Access-Datenbank im Ordner für den zweiten der Beitragsreihe gespeichert. Dort geschah aber gar nichts – keine neuen Einträge!

Damit Sie eine Möglichkeit haben, zu erkennen, dass etwas falsch läuft, fügen wir also nun zunächst etwas Code zum Access-Ticketsystem hinzu, der prüft, ob die aktuell geöffnete Access-Datenbankdatei auch diejenige ist, die in Outlook für die entsprechende Eigenschaft hinterlegt ist.

Dazu können wir auch von Access aus die im Beitrag **Benutzerdefinierte Outlook-Eigenschaften** formulierten VBA-Routinen nutzen – mit Ausnahme einer einzigen Routine, die wir noch anpassen müssen.

Die Routinen **EigenschaftEinlesen**, **EigenschaftSetzen** und **OpenFilename** können Sie aus der Beispieldatenbank zu **Benutzerdefinierte Outlook-Eigenschaften** entnehmen und in das Ticketsystem einpflegen.

```
Public Function DatenbankpfadInOutlookAktualisieren()
    Dim strDatenbankpfad As String
    Dim bolVorhanden As Boolean
    Dim strElement As String
    Dim strName As String
    strElement = "Ticketsystem"
    strName = "Datenbankpfad"
    strDatenbankpfad = EigenschaftEinlesen(strElement, strName)
    If Not Len(strDatenbankpfad) = 0 Then
        If Not Len(Dir(strDatenbankpfad)) = 0 Then
            bolVorhanden = True
        End If
    End If
    If Not bolVorhanden Then
        strDatenbankpfad = CurrentDb.Name
        MsgBox "Der Datenbankpfad für die Ticketverwaltung in Outlook ist noch nicht gesetzt. Dies wird nun erledigt.", _
            vbOKOnly + vbExclamation
        EigenschaftSetzen strElement, strName, strDatenbankpfad
    Else
        If Not (strDatenbankpfad = CurrentDb.Name) Then
            MsgBox "Der Datenbankpfad in Outlook ist nicht auf die aktuelle Datenbankdatei gesetzt, sondern auf:" _
                & vbCrLf & vbCrLf & strDatenbankpfad & vbCrLf & vbCrLf _
                & "Er wird nun auf die aktuelle Datenbank eingestellt."
            strDatenbankpfad = CurrentDb.Name
            EigenschaftSetzen strElement, strName, strDatenbankpfad
        End If
    End If
End Function
```

Listing 1: Aktualisieren des Datenbankpfades in Outlook

Die weitere Routine **DatenbankpfadHolen** überführen wir in eine parameterlose Routine namens **DatenbankpfadInOutlookAktualisieren** (s. Listing 1). Diese deklariert zunächst einige Variablen und legt deren Werte fest, zum Beispiel für die Variablen **strElement** (Wert: **Ticketsystem**) und **strName** (**Datenbankpfad**).

Außerdem haben wir den Code dieser Methode etwas angepasst. Sie soll nun nicht mehr prüfen, ob die benutzerdefinierte Eigenschaft **Datenbankpfad** einen Wert enthält und ob dieser dem Pfad einer vorhandenen Datenbankdatei entspricht, um gegebenenfalls den Benutzer aufzufordern, den passenden Pfad per **Datei öffnen**-Dialog anzugeben.

Das ist nicht nötig, denn wir befinden uns ja gerade in der gewünschten Datenbank-Datei. Also müssen wir nur noch prüfen: Ist in der Eigenschaft **Datenbankpfad** überhaupt ein Pfad gespeichert und entspricht dieser dem Datenbankpfad der aktuell geöffneten Datei? Falls nicht, soll die Eigenschaft einfach auf die aktuelle Datenbank eingestellt werden.

Die Routine liest über die Funktion **EigenschaftEinlesen** den aktuell in der Eigenschaft **Datenbankpfad** des **StorageItem**-Elements namens **Ticketsystem** gespeicherten Wert in die Variable **strDatenbankpfad** ein. Sie prüft, ob **strDatenbankpfad** überhaupt einen Wert enthält. Falls ja, prüft sie, ob der Wert einem gültigen Dateipfad

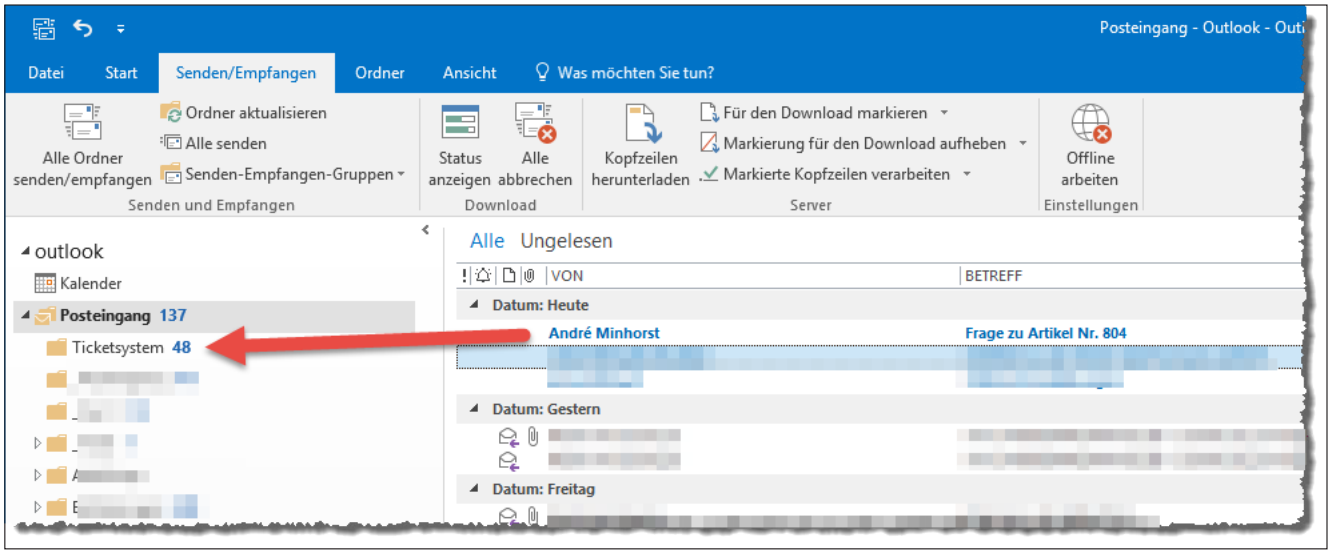

Bild 1: Verschieben einer Anfrage in den Ordner **Ticketsystem**

entspricht. Falls ja, wird die Variable **bolVorhanden** auf **True** eingestellt. Ist **bolVorhanden** danach **False**, stellt die Prozedur den Wert der Variablen **strDatenbankpfad** auf den Pfad zur aktuellen Datenbankdatei ein und ruft dann die Funktion **EigenschaftSetzen** auf, um den Wert der Eigenschaft **Datenbankpfad** auf den Namen der aktuellen Datenbankdatei einzustellen – nicht ohne den Benutzer zuvor über diese Aktion zu informieren.

War bereits ein Datenbankpfad gespeichert, vergleicht die Prozedur diesen mit dem Pfad zur aktuellen Datenbankdatei. Sind die beiden Zeichenketten nicht gleich, erscheint ebenfalls eine Meldung und der Pfad der aktuellen Datenbankdatei landet in der benutzerdefinierten Eigenschaft **Datenbankpfad** von Outlook.

Mails von Outlook halbautomatisch in die Datenbank einlesen

Bevor wir uns nun weiter um die Verarbeitung der Tickets in Access kümmern, wollen wir noch eine Möglichkeit schaffen, neue Kundenanfragen, die im Ticketsystem verarbeitet werden sollen, dort hinein zu kopieren.

Dies soll auf der Outlook-Seite ganz einfach geschehen: Der Benutzer soll die infrage kommenden Mails von Kunden einfach per Drag and Drop in einen Ordner namens **Ticketsystem** ziehen (s. Bild 1).

Wenn die E-Mails dann im Zielordner gelandet sind, sollen sie vor allem mit dem Zusatz **[Ticket xxx]** ausgestattet sein. Damit können wir zwei Dinge erkennen: Erstens, dass der Mechanismus, der durch das Hineinziehen einer Mail in diesen Ordner ausgelöst werden soll, funktioniert hat.

Zweitens soll der Platzhalter **xxx** den Wert des Primärschlüsselfeldes enthalten, unter dem die Mail als neuer Datensatz in der Tabelle **tblMailItems** angelegt wurde (s. Bild 2).

Mechanismus zum Übertragen der Kundenanfragen in Outlook

Damit kommen wir zu dem Mechanismus selbst. Dieser soll ausgelöst werden, wenn wir eine E-Mail in den Ordner **Ticketsystem** von Outlook ziehen.

Diesen initialisieren wir in einer Prozedur namens **Application_Startup_Ticketverwaltung** (s. Listing 2). Diese wird nicht automatisch mit Outlook gestartet. Allerdings löst Outlook beim Start automatisch eine Prozedur aus,

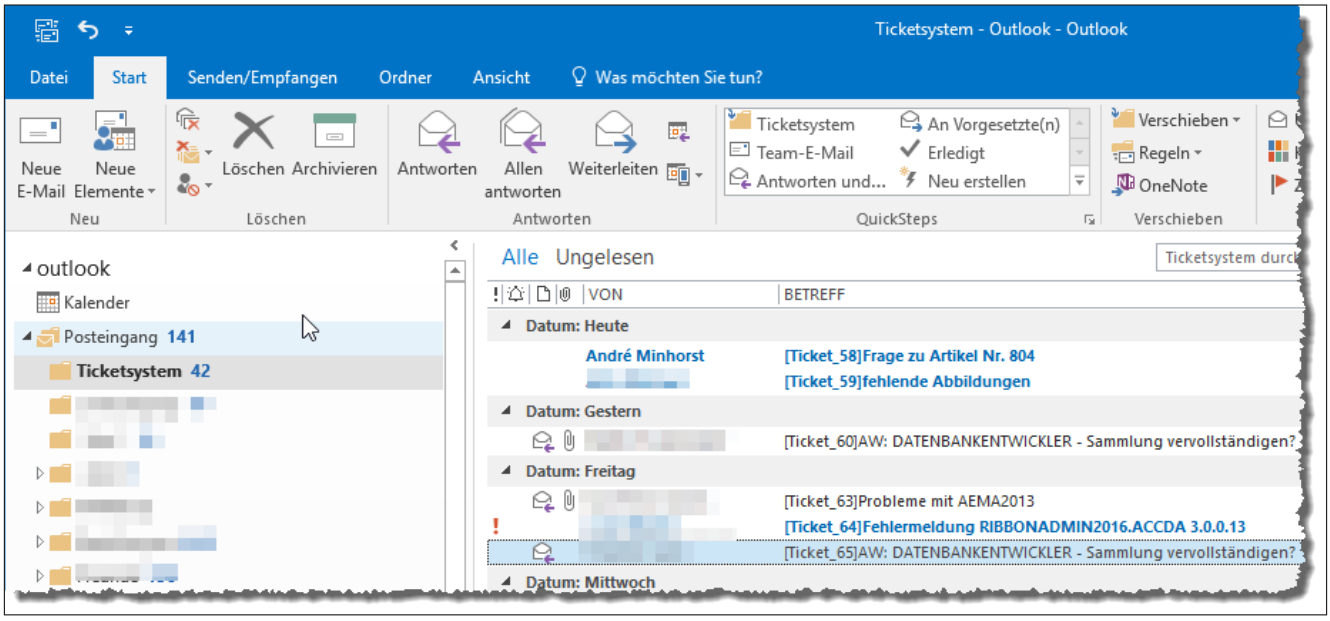

Bild 2: In den Ordner **Ticketsystem** verschobene Anfragen

die wir **Application_Startup** nennen und die sich in der Klasse **ThisOutlookSession** des VBA-Projekts von Outlook befindet.

Sollten Sie diese noch nicht verwenden, tragen Sie nur den Aufruf der Prozedur **Application_Startup_Ticketverwaltung** dort ein. Anderenfalls fügen Sie diesen Aufruf einfach an die bereits vorhandenen Anweisungen an:

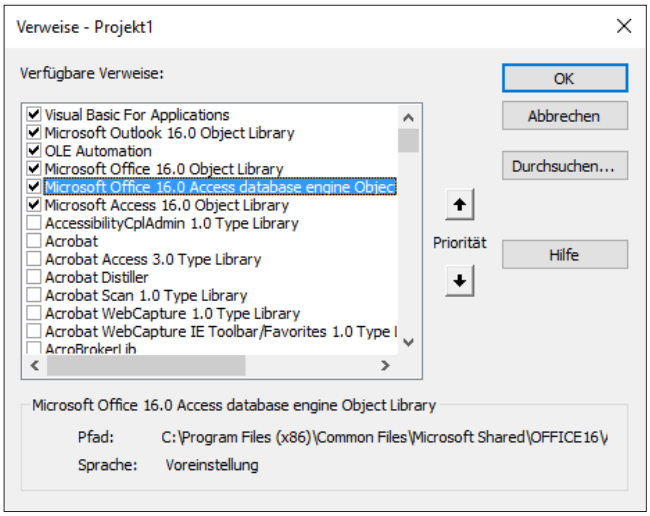

Bild 3: Verweis auf die DAO-Bibliothek, zusätzlich auch noch auf die Access-Bibliothek

```
Public Sub Application_Startup()
    ...
    Application_Startup_Ticketverwaltung
End Sub
```

Die Prozedur **Application_Startup_Ticketverwaltung** deklariert Variablen zum Speichern von Verweisen auf die Ticketsystem-Datenbank und ein Recordset für den Zugriff auf die Optionentabelle, ein **Folder**-Objekt für den Zugriff auf den Ziel-Ordner in Outlook sowie eine Variable zum Speichern von Objekten des Typs **clsFolderArchiv**.

Da wir hier innerhalb des Outlook-VBA-Projekts auf Elemente der DAO-Bibliothek zugreifen, fügen wir dem VBA-Projekt noch einen Verweis auf diese Objektbibliothek hinzu – plus einen Verweis auf die Access-Bibliothek, falls wir mal eine Funktion wie **Nz()** oder **DLookup()** benötigen (s. Bild 3).

Die Prozedur verwendet dann zunächst die Funktion **DatenbankpfadHolen**, die wir bereits im Beitrag **Benutzerdefinierte Outlook-Eigenschaften** beschrieben haben. Diese Funktion liefert den Pfad zur Datenbank **Ticketsyste.accdb** zurück, der in der dafür angelegten

```
Public Sub Application_Startup_Ticketverwaltung()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim objFolder As Outlook.Folder
    Dim objFolderArchiv As clsFolderArchiv
    Dim strTicketsystemDatenbank As String
    On Error GoTo Application_Startup_Err
    strTicketsystemDatenbank = DatenbankpfadHolen("Ticketsystem", "Datenbankpfad")
    Set db = DBEngine.OpenDatabase(strTicketsystemDatenbank, , True)
    Set rst = db.OpenRecordset("SELECT * FROM tblOptionen", dbOpenDynaset)
    Set colFolders = New Collection
    Do While Not rst.EOF
        Set objFolderArchiv = New clsFolderArchiv
        With objFolderArchiv
            Set objFolder = GetFolderByPath(rst!Verzeichnis)
            If objFolder Is Nothing Then
                MsgBox "Der in der Export-Datenbank '" & strTicketsystemDatenbank & "' angegebene Outlook-Ordner '" _
                    & rst!Verzeichnis & "' ist nicht in Outlook vorhanden. Wählen Sie diesen nun erneut aus."
                Set objFolder = Outlook.GetNamespace("MAPI").PickFolder
                db.Execute "UPDATE tblOptionen SET Verzeichnis = '" & objFolder.FolderPath & "'", dbFailOnError
            End If
            Set .Folder = objFolder
            .AnlagenSpeichern = rst!AnlagenSpeichern
            Set .Database = db
            .NeuEinlesen = rst!NeuEinlesen
            .Groesse = Nz(rst!Groesse)
        End With
        colFolders.Add objFolderArchiv
        If rst!Rekursiv Then
            UnterordnerInstanzieren objFolder, db, Nz(rst!Groesse), rst!NeuEinlesen, rst!AnlagenSpeichern, colFolders
        End If
        rst.MoveNext
    Loop
End Sub
```

Listing 2: Starten des Mechanismus zum automatischen Übertragen von E-Mails, die in einen bestimmten Ordner verschoben werden

benutzerdefinierten Eigenschaft gespeichert ist. Diese Vorgehensweise verwenden wir, damit wir diesen Pfad nicht fest im Code verdrahten müssen – etwa in Form einer Konstanten –, sondern ihn auch einmal dynamisch ändern können, wenn sich der Speicherort des Ticketsystems ändert. Dann öffnen wir mit der **OpenDatabase**-Methode die soeben ermittelte Datenbankdatei und speichern einen Verweis darauf in der **Database**-Variablen **db**.

Als Nächstes greifen wir über die **OpenRecordset**-Methode auf die Tabelle **tblOptionen** dieser Datenbank zu und speichern den Verweis darauf in der **Recordset**-Variablen **rst**.

Die Prozedur könnte auch mehrere Ordner als Zielordner definieren, daher verwenden wir eine Collection, um für jeden Zielordner ein Objekt der Klasse **clsFolderArchiv** zu speichern. Diese deklarieren wir wir folgt im Kopf des Moduls **ThisOutlookSession**:

```
Dim colFolders As Collection
```

Da wir wie gesagt auch mehrere Ordner nutzen könnten, durchlaufen wir die Datensätze der Tabelle **tblOptionen** in einer **Do While**-Schleife. Im ersten Schritt erstellen wir eine neue Instanz der Klasse **clsFolderArchiv**.

Dieser weisen wir einige Informationen zu, die wir dem aktuellen Datensatz der Tabelle **tblOptionen** entnehmen. Der erste ist der Zielordner. Den Pfad zu diesem Ordner, in unserer Beispielkonstellation **\\Outlook\Posteingang\Ticketsystem**, lesen wir aus dem Feld **Verzeichnis** ein.

Dann verwenden wir die Hilfsfunktion **GetFolderByPath**, um ein **Folder**-Objekt zu diesem Verzeichnis zu ermitteln und das Ergebnis in der Variablen **objFolder** zu speichern.

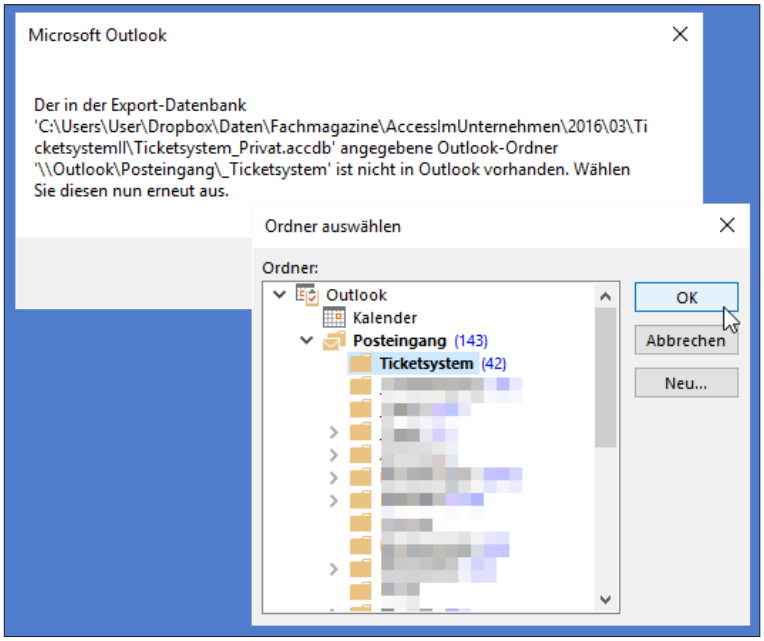

Bild 4: Auswählen eines neuen Zielordners für die Kundenanfragen

Die Funktion **GetFolderByPath** erwartet den Pfad als String. Sie durchläuft alle Ordner, auch rekursiv, bis sie den entsprechenden Outlook-Folder gefunden hat. Dieser landet dann in der Variablen **objFolder**.

Diese Funktion haben wir, neben einigen anderen Funktionen für den Zugriff auf die Outlook-Objekte, schon in mehreren Ausgaben genutzt. In der nächsten Ausgabe finden Sie im Beitrag **Outlook-Ordner im Griff** (**www.access-im-unternehmen.de/1043**) eine Erläuterung dieser Funktion sowie einige weiterführende Informationen zu Outlook-Ordnern.

Es kann sein, dass die Funktion **GetFolderByPath** kein Ergebnis zurückliefert. Das ist genau dann der Fall, wenn es kein **Folder**-Objekt mit dem angegebenen Pfad gibt.

Deshalb prüfen wir mit der folgenden **If...Then**-Bedingung, ob **objFolder** den Wert **Nothing** liefert, also dass es leer ist. Ist das der Fall, folgt eine entsprechende Meldung und die **PickFolder**-Methode zeigt einen Dialog an, mit dem der Benutzer den gewünschten Zielordner auswählen kann (s. Bild 4).

Der auf diese Weise der Variablen **objFolder** zugewiesene Outlook-Folder liefert mit der Eigenschaft **FolderPath** seine Pfadangabe, die wir mit der nächsten Anweisung per **UPDATE**-Abfrage in der Tabelle **tblOptionen** speichern.

Nun können wir die Eigenschaft **Folder** der Klasse **clsFolderArchiv** auf dieses Objekt einstellen. Auf wesentlich einfachere Weise füllen wir nun die Eigenschaften **AnlagenSpeichern**, **NeuEinlesen** und **Groesse**, denen wir einfach die Werte der entsprechenden Felder des Recordsets **rst** zuweisen – und die Eigenschaft **Database** erhält einen Verweis auf das Objekt **db**, also die aktuelle referenzierte Datenbank.

Das so gefüllte Objekt **objFolderArchiv** fügen wir dann per **Add**-Methode zum **Collection**-Objekt **colFolders** hinzu, damit es nach Beenden der Prozedur und somit dem Verfallen der Gültigkeit der Objektvariablen nicht in den ewigen Jagdgründen landet. Auf die gleiche Weise werden, sollten auch Unterordner dieses Ordners als Ziel

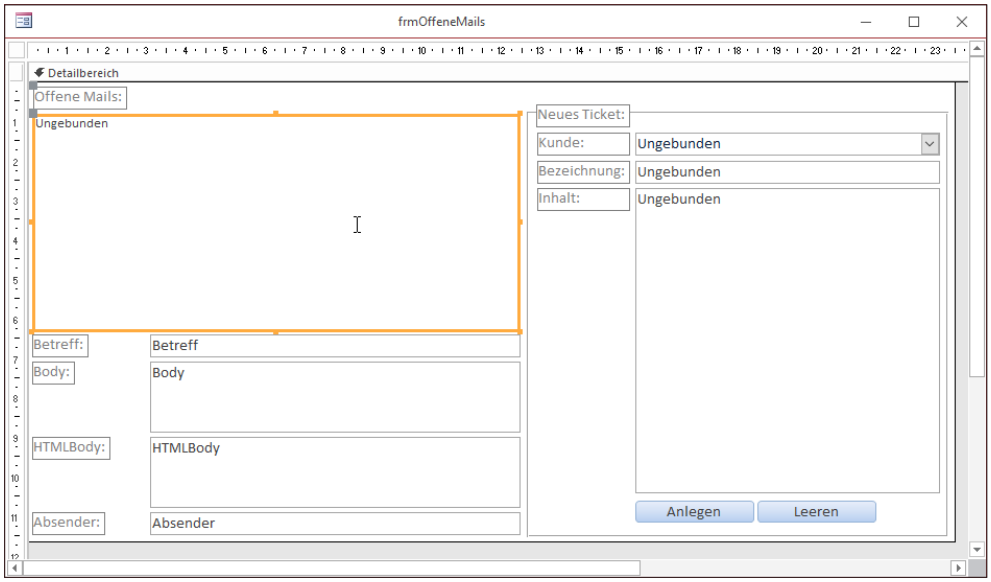

Bild 5: Das Formular **frmOffeneMails** in der Entwurfsansicht

für das Verschieben von Kundenanfragen infrage kommen, auch noch alle Ordner unterhalb des angegebenen Ordners referenziert und in der Collection **colFolders** gespeichert – und natürlich deren Unterordner und so weiter. Dafür muss jedoch das Feld **Rekursiv** der Tabelle **tblOptionen** den Wert **True** aufweisen.

Die Klasse **clsFolderArchiv** haben wir weitgehend aus der gleichnamigen Klasse des Beitrags **Outlook-Mails nach Empfang archivieren** (**www.access-im-unternehmen.de/1007**) übernommen. Es gibt ein paar kleinere Unterschiede.

So verfügt unsere Tabelle **tblMailItems** in der Ticketsystem-Datenbank über ein eindeutiges Feld namens **EntryID**, sodass eine E-Mail nur einmalig in der Tabelle gespeichert werden kann und eine Fehlermeldung angezeigt wird, wenn der Benutzer die gleiche Mail zwei Mal in einen der Outlook-Ordner der Ticketverwaltung zieht.

Die übrigen Vorgänge sind mit den in diesem Beitrag beschriebenen Abläufen identisch: Beim Hineinziehen einer E-Mail in einen Outlook-Ordner, den wir mit einem Objekt der Klasse **clsFolderArchiv** referenziert haben, löst dies ein Ereignis aus, das den Inhalt der E-Mail in die Tabelle **tblMailItems** der Zieldatenbank schreibt.

Funktionsweise des Formulars frmOffeneMails

Das Formular **frmOffeneMails** soll alle Mails anzeigen, die in den entsprechenden Outlook-Ordner gezogen und so automatisch in der Tabelle **tblMailItems** gespeichert wurden (s. Bild 5). Dazu nutzt das Formular das Listenfeld **lstOffeneMails** links oben.

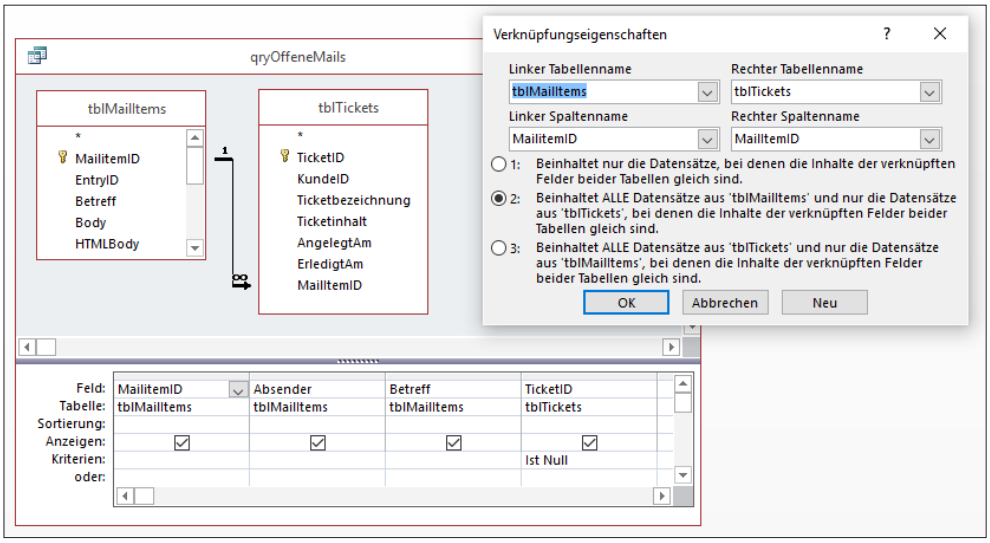

Bild 6: Datensatzherkunft für das Listenfeld **lstOffeneMails**

ebay-Zugriff mit Access

Bereits im Jahr 2009 haben wir gezeigt, wie Sie von Access aus Daten von ebay auslesen und selbst Angebote einstellen. Über die Jahre ändern sich jedoch die Mechanismen einer solchen Online-Schnittstelle, also schauen wir uns in diesem Beitrag an, wie der aktuelle Zugriff auf ebay aussieht und wie Sie diesen von einer Access-Anwendung aus realisieren. Dazu legen wir einen neuen Benutzer- und Entwickler-Account an und erstellen dann die notwendigen Objekte, um von Access aus auf die Daten zuzugreifen.

Für das Verständnis der folgenden Abhandlung ist es wichtig zu wissen, dass Sie sowohl einen normalen eBay-Account als auch einen Entwickler-Account benötigen.

Neuen ebay-Account anlegen

Falls Sie noch kein ebay-Konto haben, öffnen Sie einfach die ebay-Seite unter **www.ebay.de**. Dort finden Sie ganz unten unter **KaufenlNeu anmelden bei ebay** alle notwendigen Informationen. Kurz gefasst klicken Sie dort auf **Jetzt bei eBay anmelden** und landen auf der Seite **signin.ebay.de**. Hier geben Sie Ihre Daten auf der Registerseite **Neu anmelden** ein (s. Bild 1).

Sie erhalten dann einen eindeutigen, automatisch generierten Benutzernamen wie **andminhors_0**, den Sie aber auch durch einen selbst gewählten Benutzernamen überschreiben können. Damit sind die Arbeiten auf dieser Seite bereits erledigt. Die meisten allgemeinen Informationen finden Sie nun im Bereich **Mein eBay**.

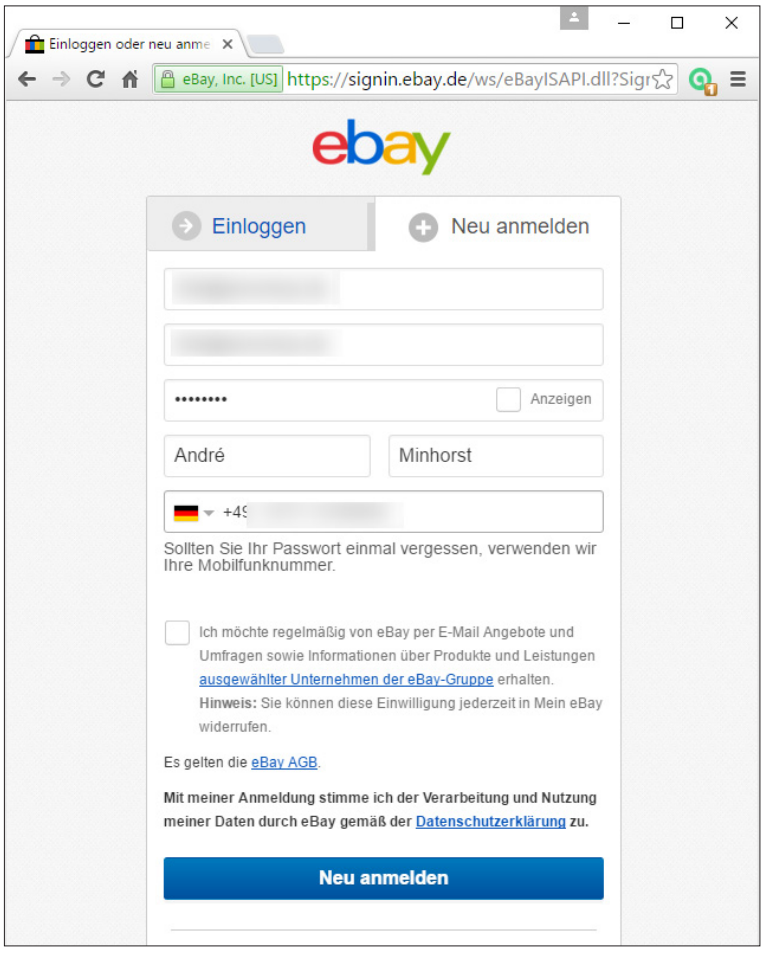

Bild 1: Neu registrieren bei **ebay**

Neuen Entwickler-Account anlegen

Nun öffnen Sie die Webseite **developer.ebay.com** und damit das ebay-Entwicklerprogramm. Auch hier registrieren wir uns zunächst, und zwar durch einen Klick auf **Sign In/Join**. Auch dort nutzen wir ein Formular, um unsere Daten einzugeben (s. Bild 2). Damit wir die beiden Benutzer nachher unterscheiden können, habe ich dem Benutzernamen beim Developer-Programm die Zeichenfolge **_api** angehängt.

Application Keys holen

Im folgenden Dialog geben Sie für die Vergabe der sogenannten **Application Keys**, die Sie später benötigen, den Namen der geplanten Anwendung ein – hier kurz und bündig **accessBay** (s. Bild 3). Nun erscheinen unten Links zum Erstellen von Keysets für Sandbox- und Production-

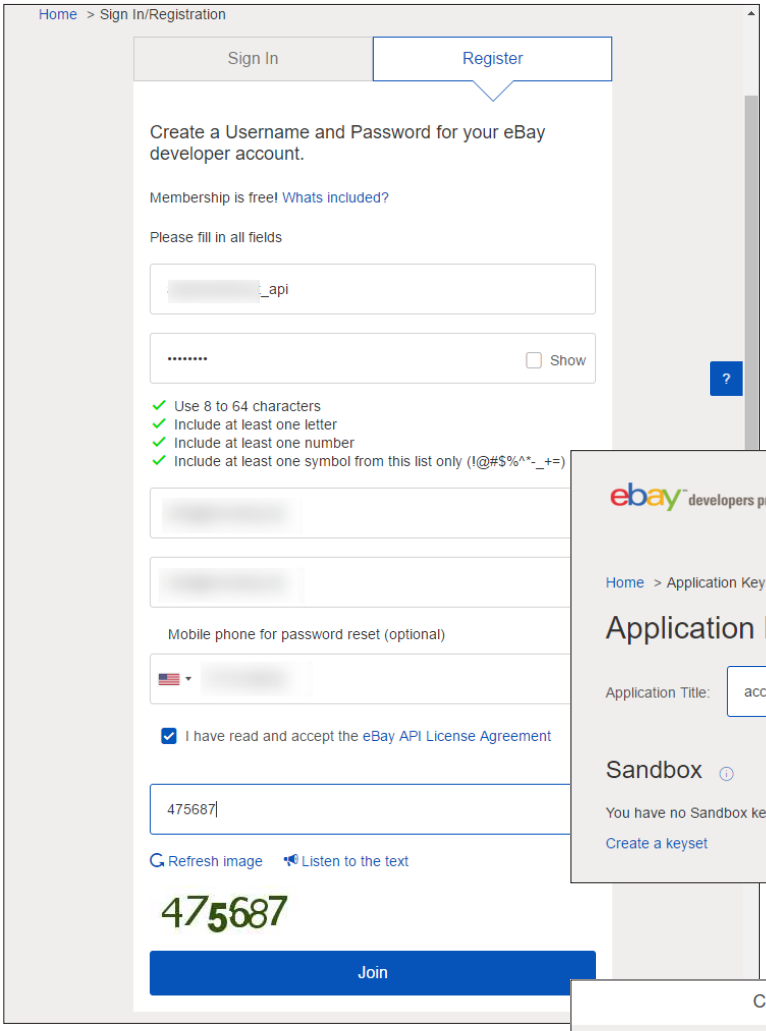

Bild 2: Neu registrieren als Developer bei **ebay**

Modus. Sie entscheiden selbst, ob Sie zunächst in der Sandbox üben wollen oder direkt auf echte ebay-Daten zugreifen wollen.

Nach der Auswahl des Eintrags Production fragt ebay einige weitere Daten zu Ihrer Person ab, um Sie kontaktieren zu können, falls es im Rahmen der Benutzung der Keys zu Problemen kommt (s. Bild 4).

Danach erhalten Sie eine Übersicht, über die Sie wahlweise auch noch ein Keyset für den Bereich erstellen können, den Sie noch nicht berücksichtigt haben – also wahlweise **Sandbox** oder **Production** (s. Bild 5). Gleichzeitig weist ebay im Falle des Production-KeySet darauf hin, dass Sie 5.000 Aufrufe pro Tag an die ebay-Api absetzen können. Für mehr ist eine Ansicht der Anwendung durch das ebay-Team erforderlich.

User Token erstellen

Nun benötigen Sie noch einen User Token für das Keyset, das Sie verwenden möchten.

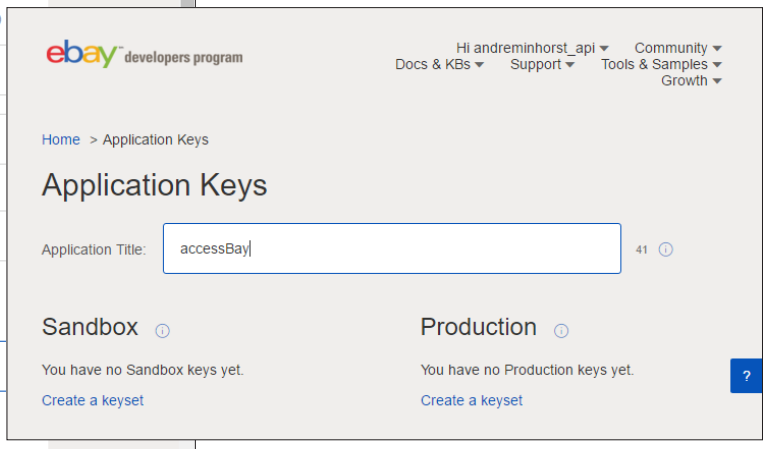

Bild 3: Anwendungsname angeben

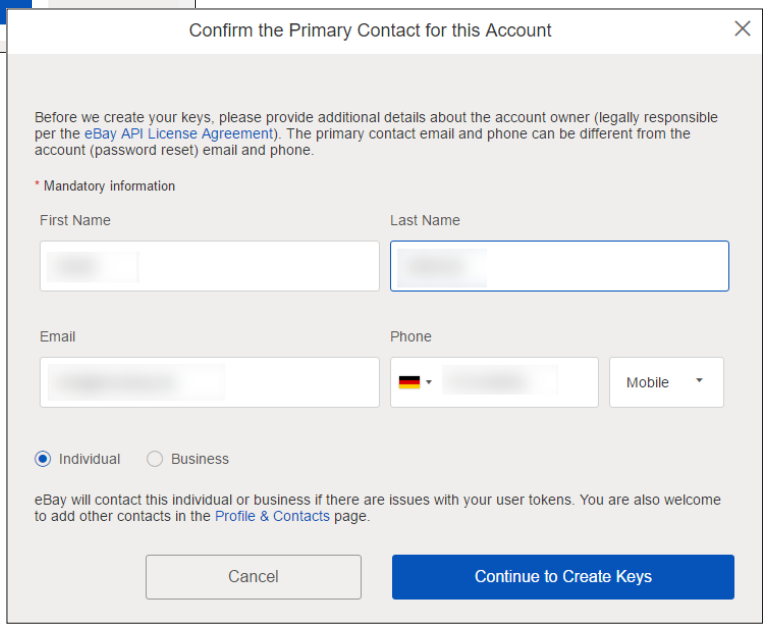

Bild 4: Angabe weiterer Informationen

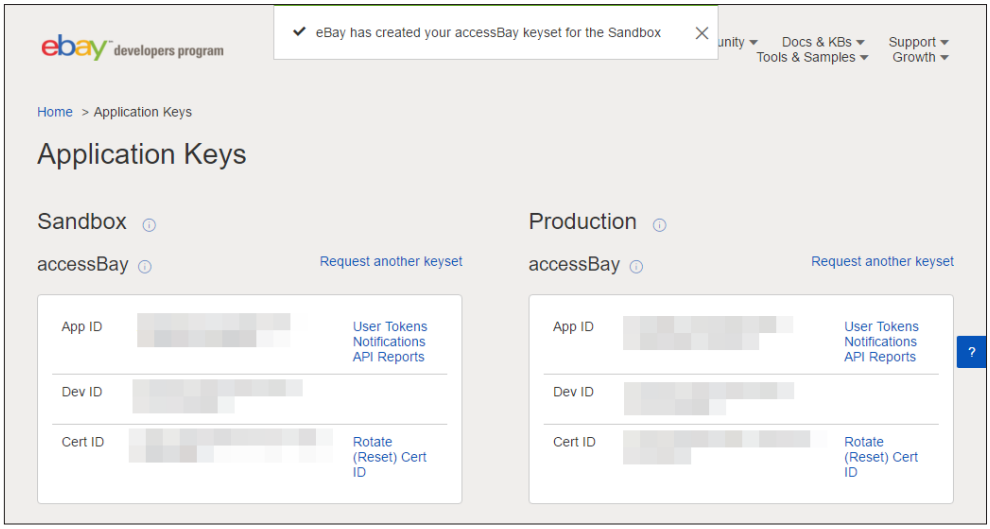

Bild 5: Übersicht mit den Application Keys für den Sandbox- und für den Production-Modus

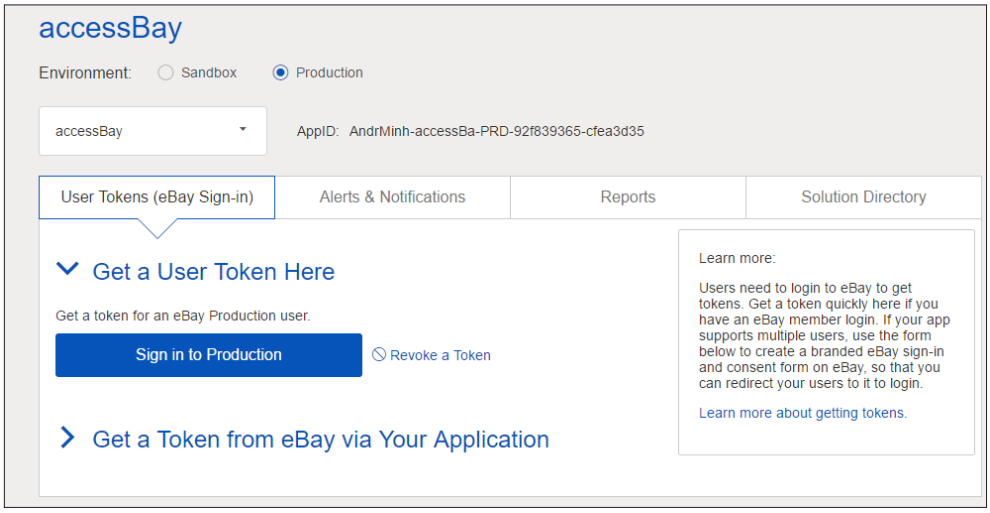

Bild 6: Erstellen eines User Tokens

Dazu klicken Sie rechts neben **App ID** auf den Link **User Tokens**.

Der Clou des User Tokens ist, dass es sich dabei um einen Schlüssel handelt, der im Kontext eines ebay-Benutzerkontos erstellt wird und der sich auf die zu programmierende Anwendung bezieht. Da wir eine Anwendung nur für uns selbst programmieren möchten, ist dies die richtige Option für uns. Die Alternative wäre, einen Token anzufragen, der eine Anwendung ermöglicht, die beliebige ebay-Nutzer verwenden können, ohne dass jeder einzelne sich einen User Token holen muss – die Benutzer müssen sich dann lediglich selbst anmelden.

Den Token erhalten wir über den Dialog aus Bild 6. Hier klicken Sie auf **Sign in to Production** (sofern Sie einen Token für den Production-Modus anfragen – sonst würde hier stehen **Sign in to Sandbox**. Dazu müssten Sie allerdings auch noch einen neuen Sandbox-User anlegen). Die gewünschte Umgebung können Sie über die Optionen oben auf der Seite auswählen.

Im folgenden Dialog fragt ebay nach weiteren Daten zu Ihrer Person, also zu der Person, die das Entwickler-Konto nutzt. In diesem Fall werden die bestehenden Daten um die Adressdaten ergänzt.

Anschließend erscheint der Anmeldedialog der ebay-Seite selbst, also nicht der der Entwicklerplattform. Hier melden Sie sich nun mit den Zugangsdaten des eingangs erstellten (oder bereits vorhandenen) ebay-Benutzers an, in dessen Namen die App auf ebay zugreifen soll.

Im folgenden Dialog klicken Sie noch auf **I agree** und erhalten dann eine Seite später Zugriff auf Ihren User Token, mit dem Sie im Namen des angegebenen ebay-Benutzerkontos über Ihre App auf ebay zugreifen (s. Bild

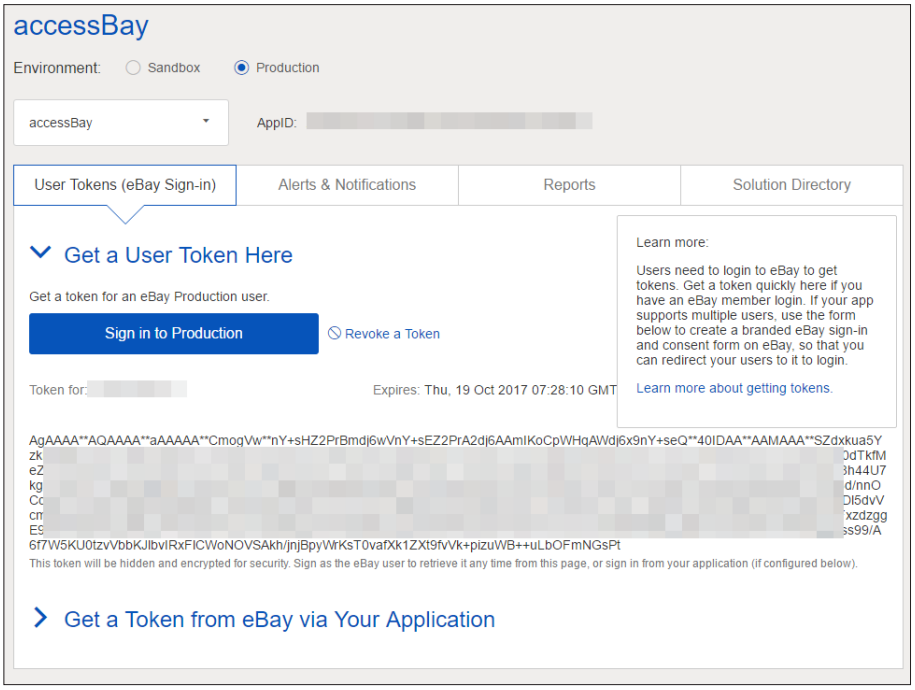

Bild 7: Dialog mit dem fertigen User Token

8). Der Token ist die recht lange, kryptische Zeichenfolge, die Sie für folgende Anwendungszwecke immer komplett benötigen.

Damit haben Sie alle Daten zusammen, die Sie für den Zugriff von einer Access-Datenbankanwendung auf ebay benötigen. Nun erstellen wir die Access-Anwendung selbst und programmieren eine kleine Benutzeroberfläche sowie den Code für den Zugriff auf die ebay-Schnittstelle per XML.

Sandbox-User anlegen

Das Anlegen eines Sandbox-Users erledigen Sie ebenfalls vom obigen Dialog aus. Dort wählen Sie unter **Environment** die Option **Sandbox** und finden unter der Schaltfläche **Sign in to Sandbox** den Link **Register a new Sandbox user** vor. Der Dialog hier sieht natürlich anders aus als der zum Registrieren eines »echten« ebay-Benutzers (s. Bild 7). Hier sind nur wenige Informationen nötig, dafür aber auch einige, die Sie bei der herkömmlichen Registrierung nicht erhalten. Wichtig für Entwickler, die gegebenenfalls eine App für den Zugriff auf andere als die deutsche ebay-Seite bauen wollen, ist die Option **Registration Site**. Hier können Sie auch **United States (US)**, **Motors** oder andere Länder auswählen. Diese Option werden Sie später in der Tabelle **tblOptionen** im Feld **SiteID** vorfinden.

Merken Sie sich, dass der Benutzername des Sandbox-Users mit **TESTUSER_** beginnt.

Tabelle zum Speichern der Accountdaten

Nun haben wir bereits einige Daten, die wir der Developer-Seite von ebay entnehmen und in unserer Anwendung

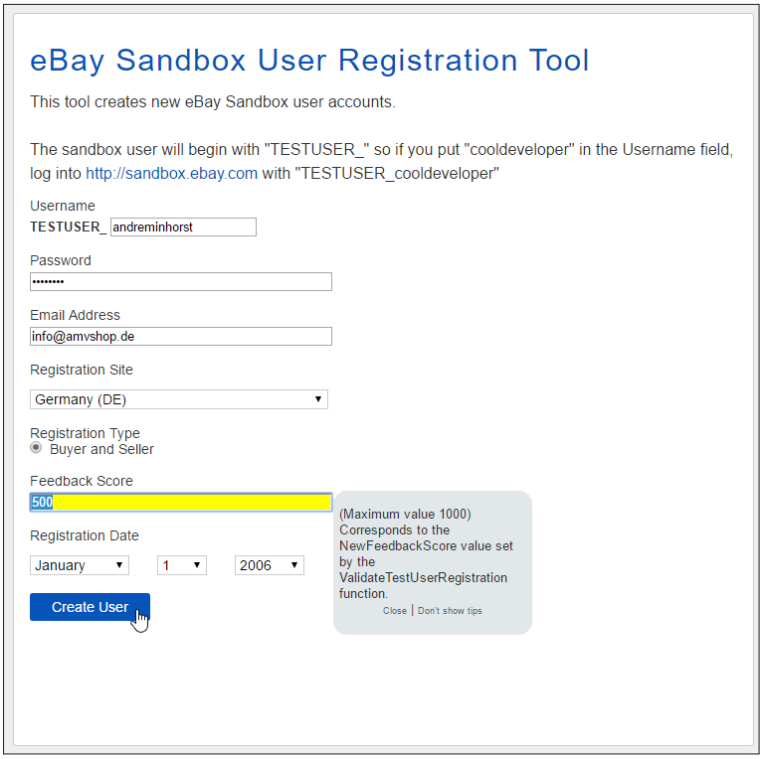

Bild 8: Registrierung eines Sandbox-Users

speichern wollen – die **DevID**, **AppID**, **CertID** und den **Token** jeweils für einen Produktiv- und einen Sandbox-Account. Dazu legen wir eine Tabelle namens **tblOptions** an, die im Entwurf wie in Bild 9 aussieht.

Neben den Account-Daten nimmt die Tabelle noch einige weitere Informationen auf. Dazu gehört etwa ein **Ja/Nein**-Feld namens **Sandbox**, das festlegt, ob Sie gerade den Sandbox- oder den Produktiv-Account für den Zugriff über die Anwendung nutzen wollen.

tblOptionen

Feldname	Felddatentyp	Beschreibung (optional)
DevID_Production	Kurzer Text	DevID für Produktivumgebung
AppID_Production	Kurzer Text	AppID für Produktivumgebung
CertID_Production	Kurzer Text	CertID für Produktivumgebung
UserToken_Production	Langer Text	UserToken für Produktivumgebung
DevID_Sandbox	Kurzer Text	DevID für die Sandbox
AppID_Sandbox	Kurzer Text	AppID für die Sandbox
CertID_Sandbox	Kurzer Text	CertID für die Sandbox
UserToken_Sandbox	Langer Text	UserToken für die Sandbox
Sandbox	Ja/Nein	Sandbox oder produktiv?
WarningLevel	Kurzer Text	Empfindlichkeit für die Ausgabe von Warn-/Fehlerhinweisen
ErrorLanguage	Kurzer Text	Sprache für die Ausgabe von Fehlermeldungen
Item_Country	Kurzer Text	Code für das Land
Item_Currency	Kurzer Text	Code für die Währung
SiteID	Zahl	ID der zu verwendenden Zielseite, zum Beispiel
Version	Zahl	Gibt die zu verwendende Version der ebay-API an.
RequestVersion	Zahl	Gibt die zu verwendende Version des Requests an.
RequestURLTrading_Sandbox	Kurzer Text	URL für Trading-API/Sandbox
RequestURLShopping_Sandbo:	Kurzer Text	URL für Shopping-API/Sandbox
RequestURLTrading_Productio	Kurzer Text	URL für Trading-API/Production
RequestURLShopping_Product	Kurzer Text	URL für Shopping-API/Production
Version_Kategorien	Zahl	Version der Kategorien
RequestURLFinding_Sandbox	Kurzer Text	URL für Finding-API/Sandbox
RequestURLFinding_Productio	Kurzer Text	URL für Finding-API/Production
AnzahlSeite	Zahl	Anzahl der Suchergebnisse je Seite
Sortierung	Kurzer Text	Sortierung der Suchergebnisse
AnzahlTreffer	Zahl	Zuletzt angeforderte maximale Anzahl von Suchergebnissen
Suchbegriff	Kurzer Text	Zuletzt verwendeter Suchbegriff

Bild 9: Tabellen zum Speichern der Optionen

Weitere Felder legen allgemeine Eigenschaften für den Zugriff auf die ebay-Plattform fest, zum Beispiel um zu definieren, wie diese auf die Übergabe fehlerhafter XML-Elemente in Anfragen reagieren soll, in welcher Sprache Fehlermeldungen auszugeben sind, in welcher Version die XML-Anfragen formuliert sind, auf welche ebay-Seite zugegriffen werden soll und welche URLs dazu verwendet werden – Letzteres jeweils wieder für den Sandbox- und den Production-Betrieb.

frmOptionen

Sandbox: ☐

Sandbox-Account:
DevID: fdec0ec5-faa5-496c-8dd8-f4c617709d0d
AppID: AndrMinh-accessBa-SBX-12f8dabd4-f78da3fb
CertID: SBX-2f8dabd49640-feb7-4b82-b623-a55b
UserToken: AgAAAA**AQAAAA**aAAAAA**XoIgVw**nY+sHZ2PrBmdj6wVnY+sEZ2PrA2dj6wFk4GgCpiKpwWdj6x9nY+seQ**JcoDAA**AAMAA
URL Trading: https://api.sandbox.ebay.com/ws/api.dll
URL Shopping: http://open.api.sandbox.ebay.com/shopping
URL Finding: http://svcs.sandbox.ebay.com/services/search

Production-Account:
DevID: fdec0ec5-faa5-496c-8dd8-f4c617709d0d
AppID: AndrMinh-accessBa-PRD-92f839365-cfea3d35
CertID: PRD-2f8393657d2f-116f-494e-bf76-fa2d
UserToken: AgAAAA**AQAAAA**aAAAAA**tH4gVw**nY+sHZ2PrBmdj6wVnY+sEZ2PrA2dj6AAmIKoCpWHqAWdj6x9nY+seQ**40IDAA**AAMA
URL Trading: https://api.ebay.com/ws/api.dll
URL Shopping: http://open.api.ebay.com/shopping
URL Finding: http://svcs.ebay.com/services/search/Finding

Weitere Einstellungen:
WarningLevel: High
ErrorLanguage: en_US
Item_Country: DE
Item_Currency: EUR
Anzahl Seite: 10
SiteID: 77
Version: 535
RequestVersion: 459
Vers.Categories: 87

OK

Bild 10: Formular für die Eingabe der Optionen

Und auch das Land, in dessen Kontext die Abfrage abgesetzt wird, und die Währung benötigen wir später zum Zusammenstellen der Anfragen.

Formular zum Bearbeiten der Einstellungen

Damit Sie die Einstellungen komfortabel bearbeiten können, legen wir ein passendes Formular namens **frmOptionen** an (s. Bild 10). Es enthält die Tabelle **tblOptionen** als Datenherkunft und zeigt alle Felder dieser Tabelle an – aufgeteilt in verschiedene Bereiche.

Mit dem Kontrollkästchen, das an das Feld Sandbox gebunden ist, legen Sie fest, ob Sie gerade mit den Sandbox- oder den Produktiv-Daten arbeiten möchten.

Je nach Auswahl werden zur besseren Kennzeichnung die nicht verwendeten Elemente in den Bereichen **Sandbox-Account** beziehungsweise **Production-Account** deaktiviert. Wenn Sie nun die Informationen von der ebay-Developer-Webseite in die entsprechenden Felder des Formulars übertragen, sind wichtige Voraussetzungen für den Zugriff schon einmal gegeben.

Anschließend schauen wir und die übrigen benötigten Werte an:

- **WarningLevel**: Stellen wir auf **High** ein, damit wir auch Meldungen erhalten, wenn wir veraltete oder ungültige Elemente in unseren XML-Anfragen verwenden. Dies sollten Sie für den Produktivzugriff mit vielen Anfragen auf **Low** einstellen.

- **ErrorLanguage**: Sprache, in der Fehlermeldungen ausgegeben werden. **de_DE** wäre sicher komfortabel, um schnell alles zu verstehen, aber meist hilft das nicht weiter und Sie müssen nach Möglichkeiten für das Beheben von Fehlern googeln. Da es wesentlich mehr englischsprachige Quellen zu diesem Thema gibt, ist also **en_US** die bessere Wahl.

- **Item_Country**: Code aus zwei Buchstaben, der angibt, in welchem Land der Benutzer registriert ist – in unserem Fall also **DE**, **AT** oder **CH** (weitere siehe **http://developer.ebay.com/devzone/xml/docs/reference/ebay/types/CountryCodeType.html**).

- **Item_Currency**: Code aus drei Buchstaben, der die Angebotswährung angibt, also etwa **EUR** (weitere Möglichkeiten hier: **http://developer.ebay.com/devzone/xml/docs/reference/ebay/types/CurrencyCodeType.html**).

- **SiteID**: ebay-Seite, auf die zugegriffen werden soll. Hier gibt es die verschiedenen nationalen Seiten, aber auch spezielle Seiten wie **Motors**. Für Deutschland **77**, Österreich **16**, Schweiz **193** – weitere siehe hier: **http://developer.ebay.com/devzone/xml/docs/reference/ebay/extra/AddItm.Rqst.Itm.St.html**.

- **Version**: ebay wird ständig weiterentwickelt, und somit auch die Programmierschnittstelle. Mit der Versionsnummer geben Sie an, gegen welche Schnittstelle Sie programmieren. Die Schnittstelle darf nicht älter als 18 Monate sein. Den Plan für die niedrigste aktuell unterstützte Version finden Sie hier: **http://developer.ebay.com/devzone/xml/docs/HowTo/eBayWS/eBaySchemaVersioning.html#VersionSupportSchedule**

- **Vers.Categories**: Version der Kategorie-Daten

Verschiedene APIs

ebay bietet verschiedene Programmierschnittstellen an, die Sie mit entsprechenden XML-Anfragen aufrufen können. In diesem Artikel schauen wir uns zwei davon an: die Shopping-API, die den Zugriff auf angebotene Artikel erlaubt, sowie die Trading-API, mit der Sie selbst Artikel einstellen und alle weiteren Aufgaben erledigen können, die mit dem Handeln auf der ebay-Plattform zu tun haben.

Shopping-API: Artikel suchen

Die Shopping-API erlaubt beispielsweise das Suchen nach Artikeln. Um dies zu testen, wollen wir die gefundenen Artikel in einer Tabelle speichern, die wie in Bild 11 aussieht.

Das Formular zum Eingeben des Suchbegriffes und zur Ausgabe der Suchergebnisse besteht aus Haupt- und Unterformular. Das Unterformular verwendet die Tabelle

tblSuchergebnisse als Datenherkunft und zeigt die drei Felder **Artikel** und **Preis** an (s. Bild 12). Außerdem stellen wir die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein.

Das Hauptformular enthält logischerweise das Unterformular. Außerdem soll es drei Steuerelemente zur Festlegung der Suchkriterien bieten:

- **txtSuchbegriff**: Dient zur Eingabe des Suchbegriffs.
- **cboSortierung**: Erwartet die Angabe einer Sortierung. Enthält eine Wertliste mit folgender Datensatzherkunft (**Herkunftstyp** auf **Wertliste** einstellen): **"BestMatch";"Beste Treffer";"CurrentBid";"Aktuelles Gebot";"EndTime";"Angebotsende"**
- **txtAnzahlTreffer**: Angabe der Anzahl der Suchergebnisse, welche die Anfrage zurückliefern soll.

Da die Parameter möglicherweise beim nächsten Aufruf nochmals verwendet werden sollen, wollen wir diese auch in der Tabelle **tblOptionen** speichern.

Dazu fügen Sie dieser die drei Felder **Sortierung**, **AnzahlTreffer** und **Suchbegriff** hinzu und binden erst das Formular **frmArtikelsuche** über die Eigenschaft **Datenherkunft** an die Tabelle **tblOptionen** und dann die drei Steuerelemente des Formulars an die neu hinzugefügten Felder.

Die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** stellen Sie für das Hauptformular auf den Wert **Nein**, die Eigenschaft **Automatisch zentrieren** auf **Ja** ein.

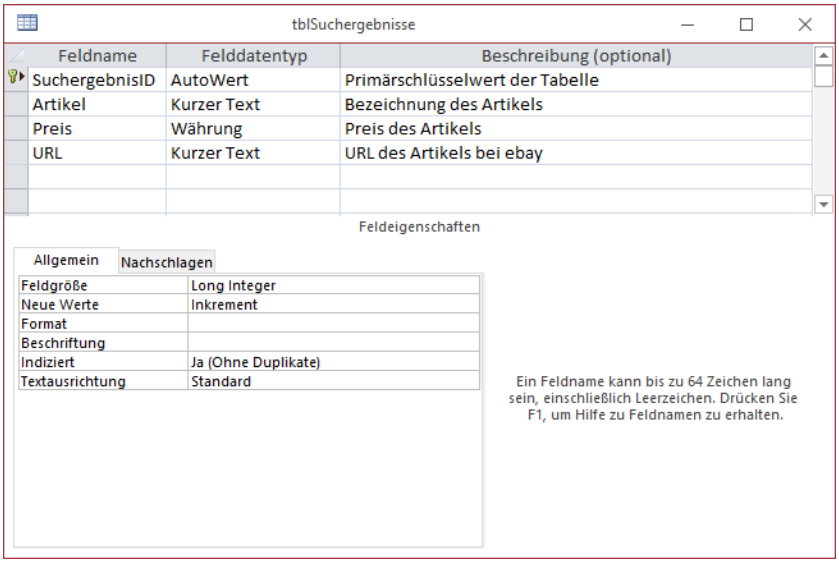

Bild 11: Tabelle zum Speichern von Suchergebnissen

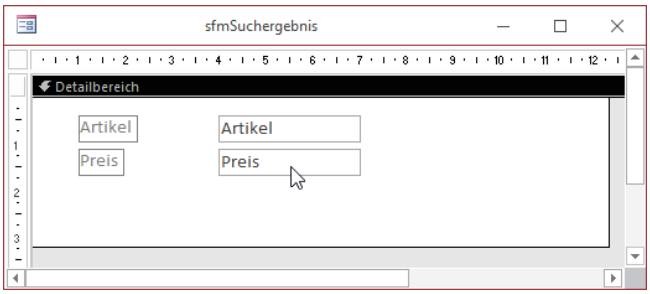

Bild 12: Unterformular zur Anzeige der Suchergebnisse

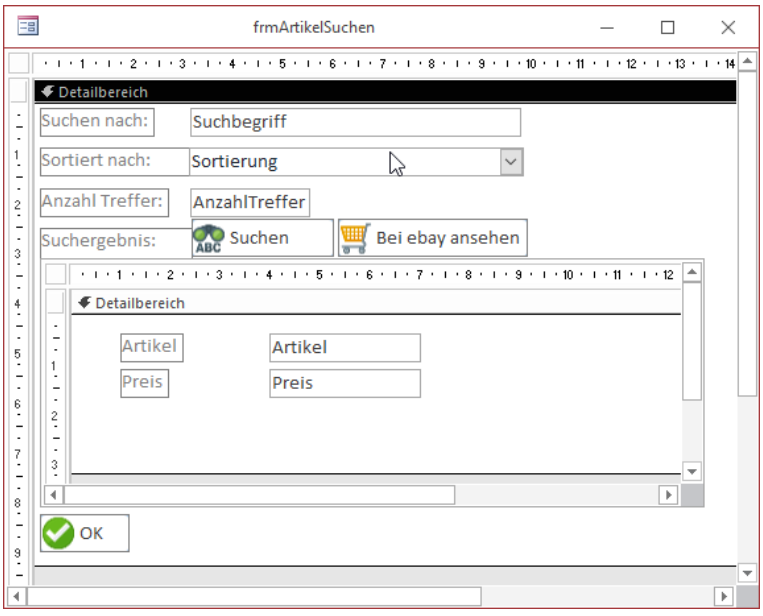

Bild 13: Hauptformular für die Suche nach ebay-Artikeln

Wir fügen dem Formular außerdem drei Schaltflächen hinzu: Die erste startet die Suche und heißt **cmdSuchen**, die zweite soll den aktuell markierten Eintrag im Unterformular bei ebay öffnen und heißt **cmdBeiEbayAnsehen** und die dritte schließt das Formular (**cmdOK**). Das Formular sieht im Entwurf wie in Bild 13 aus.

Suche programmieren

Nun brauchen wir nur noch ein wenig VBA-Code, der uns aus den Angaben in den drei Steuerelementen eine XML-Anfrage zusammenstellt, diese an ebay schickt und das Ergebnis in die Tabelle **tblSuchergebnisse** schreibt.

Den Start macht die Schaltfläche **cmdSuche**, über die wir die Suche aufrufen. Die für das Ereignis **Beim Klicken** hinterlegte Ereignisprozedur finden Sie in Listing 1.

Diese Methode löscht zunächst alle in der Tabelle **tblSuchergebnisse** gespeicherten Einträge. Dann speichert sie den Suchbegriff auf dem Textfeld **txtSuchbegriff** in der Variablen **strSuchbegriffe**. Ist das Feld leer beziehungsweise enthält es den Wert **Null**, wird dank der Funktion **Nz** eine leere Zeichenkette gespeichert.

Ist dies der Fall, meldet die Prozedur, dass noch kein Suchbegriff eingegeben wurde, und bricht die Prozedur ab. Anderenfalls ruft sie die Funktion **FindItemsAdvanced** auf.

```
Private Sub cmdSuchen_Click()
    Dim db As DAO.Database
    Dim strSuchbegriffe As String
    Set db = CurrentDb
    db.Execute "DELETE FROM tblSuchergebnisse"
    strSuchbegriffe = Nz(Me!txtSuchbegriff)
    If Len(strSuchbegriffe) = 0 Then
        MsgBox "Bitte geben Sie einen Suchbegriff ein."
        Exit Sub
    End If
    FindItemsAdvanced
    Me!sfmSuchergebnis.Form.Requery
End Sub
```

Listing 1: Starten der Suche über die Schaltfläche **cmdSuchen**

Diese soll die Suche ausführen und die Ergebnisse in der Tabelle **tblSuchergebnisse** speichern, damit der anschließende Aufruf der **Requery**-Methode des Unterformulars **sfmSuchergebnis.Form** diese im Unterformular anzeigt.

Die Funktion FindItemsAdvanced

Diese Funktion ist für das Zusammenstellen einer XML-Anfrage verantwortlich, die per HTTP-Request an die ebay-API übergeben wird und ein Ergebnis in Form eines XML-Dokuments zurückliefern soll. Dieses soll die Funktion dann mit einer weiteren Hilfsprozedur auswerten und die Ergebnisse in der Tabelle **tblSuchergebnisse** speichern.

Bei der vorherigen Version unserer Artikelabfrage von 2009 konnte man noch per einfachem REST-Aufruf, also einer einfachen URL mit ein paar Parametern, auf die Suche nach Artikeln zugreifen. Diese ist nun seit einigen Jahren veraltet und die dazu verwendeten Funktionen wurden aus der Shopping-Api in die Finding-Api übertragen.

Informationen über den grundlegenden Aufbau eines Requests an die Finding-API erhalten Sie unter folgender Adresse:

`http://developer.ebay.com/devzone/finding/Concepts/MakingACall.html`

Das heißt für uns, dass wir nicht mehr mit einer einfachen URL weiterkommen, die wir an ebay schicken, sondern wir müssen ein XML-Dokument mit den erforderlichen Informationen zusammenstellen.

Die speziellen Informationen über den Aufruf der API-Funktion **findItemsAdvanced** finden Sie hier:

`http://developer.ebay.com/devzone/finding/CallRef/findItemsAdvanced.html`