HAUFE.
Ausgabe 01/2018

ACCESS
IM UNTERNEHMEN

GRAFIK MIT HTML5

Statten Sie Ihre Access-Lösung mit Diagrammen zu Ihren Daten aus (ab S. 22).

In diesem Heft:

KOMPRIMIEREN

Verringern Sie die Menge der in Ihren Tabellen gespeicherten Daten.

SEITE 52

MSFORMS-TEXTBOX

Lernen Sie eine Textfeld-Alternative kennen, die einiges mehr bietet als das eingebaute TextBox-Element.

SEITE 4

VOLLTEXTSUCHE

Durchsuchen Sie lange Texte und zeigen Sie die Suchergebnisse übersichtlich an.

SEITE 62

Malen mit HTML5

In dieser Ausgabe bieten wir einen Exkurs in die Welt von HTML5 an: Dabei zeigen wir Ihnen, wie Sie die von Microsoft vernachlässigten Funktionen zur Ausgabe von Diagrammen mithilfe von HTML5 nachrüsten. Alles, was Sie dazu brauchen, sind die Kenntnisse, die wir Ihnen mit zwei Beiträgen vermitteln sowie ein Webbrowser-Steuerelement.

Mit Möglichkeiten zur Ausgabe von Diagrammen ist der Access-Entwickler nicht gerade gesegnet. Also schaut sich unser Autor Sascha Trowitzsch einmal an, welche Möglichkeiten die HTML5-Bibliothek in Zusammenarbeit mit dem Webbrowser-Steuerelement bietet. Der erste Beitrag zu diesem Thema heißt **HTML5 als Grafikkomponente** (ab S. 22) und zeigt, wie Sie grundsätzlich grafische Elemente erzeugen können. Darauf aufbauend zeigt der Beitrag, wie Sie beispielsweise Balkendiagramme auf Basis der Daten aus den Tabellen einer Datenbank erstellen können.

Der zweite Beitrag zu diesem Thema namens **Mit HTML5 zeichnen und malen** zeigt ab S. 42 etwa, wie Sie in einem Formular mit der Maus Bilder malen können. Der Clou dabei: Sie können nicht nur einfach Bilder malen, sondern diese auch noch zusammen mit einem Datensatz speichern und die Bilder wieder anzeigen!

Ein weiterer Beitrag mit dem Titel **Datenkompression leicht gemacht** zeigt ab S. 52, wie Sie längere Texte oder umfangreiche Binärdaten in Tabellen in komprimierter Form ablegen. So benötigen große Datenmengen je nach Art und Menge der Daten mitunter viel weniger Speicher, als es ohne weiteres Zutun der Fall gewesen wäre. Das ist gerade in der heutigen Zeit wichtig: Denn obwohl Speichern kein Problem mehr darstellt, stellen Mehrbenutzerumgebungen und vernetzte Rechner ganz eigene Herausforderungen.

Wenn Sie beim Freischalten einer Software einen Registrierungsschlüssel eingeben, kennen Sie das automatische Springen von Textfeld zu Textfeld, wenn Sie einen Fünferblock vollständig eingegeben haben. Wie Sie dies mit herkömmlichen Access-Textfeldern realisieren, lesen Sie im Beitrag **Autotab bei der Eingabe** ab S. 2.

Als Alternative zum herkömmlichen Textfeld greifen wir bereits in diesem Beitrag auf das TextBox-Element der MSForms-Bibliothek zurück. Dieses bietet einige Vorteile gegenüber dem eingebauten Pendant: Beispielsweise lassen sich damit auch Zeichenketten in sehr großen Texten markieren und diese Markierungen können einfach in den sichtbaren Bereich des Textfeldes geholt werden. Alles, was Sie zu dieser Textfeld-Alternative wissen müssen, erfahren Sie im Beitrag **Die MSForms-Textbox** ab S. 4.

Dieses Textfeld nutzen wir auch im Beitrag **Selektieren in langen Texten** (ab S. 16), welcher auch der Aufhänger für die Untersuchung des MSForms-Textfeldes ist: Dort ergab sich nämlich das Problem, dass wir keine Textpassagen markieren konnten, die sich hinter einer bestimmten Anzahl von Zeichen befinden. Dieses Problem konnten wir dann mithilfe des MSForms-Textfeldes lösen.

Angewendet haben wir die Markierung in großen Texten dann in der Lösung **Volltextsuche mit Fundstellen** (ab S. 62). Dort zeigen wir, wie Sie Suchergebnisse aus einem Memofeld komfortabel anzeigen und auswählen können, und zwar am Beispiel meiner Artikeldatenbank.

Und nun: Viel Spaß beim Lesen!

Ihr André Minhorst

Autotab bei der Eingabe

Sie kennen das von den Dialogen zur Eingabe von Registrierungsschlüsseln: Sie geben den ersten Fünferpack des Schlüssels ein und der Fokus springt dann automatisch zum folgenden Feld, wo Sie den nächsten Fünferpack eingeben können. Dieses Verhalten wollen wir einmal nachbilden. Dieser Beitrag zeigt, wie das durch den geschickten Einsatz von Ereignisprozeduren leicht gelingt.

Formular erstellen

Als Erstes erstellen wir ein Formular namens **frmKeys**, dem wir die fünf Textfelder **txtKey1**, **txtKey2**, **txtKey3**, **txtKey4** und **txtKey5** hinzufügen. Dazwischen platzieren wir jeweils ein Minuszeichen (s. Bild 1).

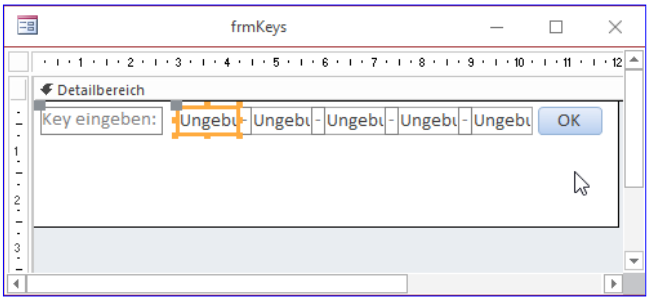


Bild 1: Das Formular **frmKeys** in der Entwurfsansicht

Vorgaben

Was geschieht nun? Optimalerweise gibt der Benutzer nacheinander die 25 Zeichen des Keys ein. Dabei wollen wir dafür sorgen, dass immer nach Eingabe von fünf Zeichen ein automatischer Sprung zum nächsten Textfeld erfolgt. Das erledigen wir etwa für das erste Textfeld mit der Prozedur, die durch das Ereignis **Bei Taste auf** ausgelöst wird und die wir wie folgt füllen:

```
Private Sub txtKey1_KeyUp(KeyCode As Integer, _
        Shift As Integer)
    If Len(Me!txtKey1.Text) = 5 And _
            Me!txtKey1.SelStart = 5 Then
        Me!txtKey2.SetFocus
    End If
End Sub
```

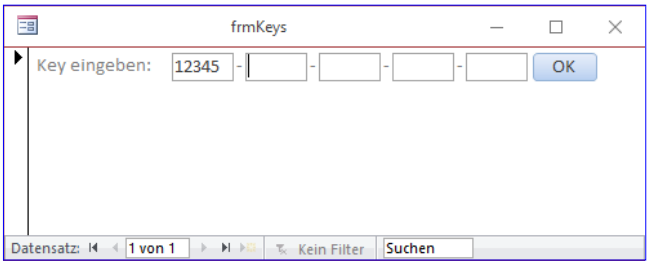


Bild 2: Verschieben des Fokus in Aktion

Wenn Sie nun fünf Zeichen in das erste Textfeld eingeben, landet die Einfügemarke wie in Bild 2 im zweiten Textfeld. Die **If...Then**-Bedingung in der Prozedur prüft, ob die Anzahl der Zeichen im Textfeld **txtKey1** dem Wert **5** entspricht und ob sich die Einfügemarke an der letzten Position befindet. In diesem Fall wird der Fokus auf das zweite Textfeld **txtKey2** verschoben.

Diese Prozedur könnten wir nun für die ersten vier Textfelder duplizieren. Allerdings wollen wir dies gleich ein wenig refaktorieren, indem wir die Anweisungen innerhalb der Prozedur in eine eigene Prozedur auslagern, der wir nur noch die Nummer aus der Bezeichnung des Textfeldes übergeben. Die Prozedur **txtKey1_Up** sieht danach etwa so aus:

```
Private Sub txtKey1_KeyUp(KeyCode As Integer, _
        Shift As Integer)
    KeyUp KeyCode, Shift, 1
End Sub
```

Hier wird also die Prozedur **KeyUp** aufgerufen, der wir neben den Parametern der aufrufenden Ereignisprozedur noch die Nummer der Textbox übergeben, welche das

Die MSForms-Textbox

Die MSForms-Bibliothek beherbergt einige Steuerelemente, die unter Access ein stiefmütterliches Dasein erleben. Es gibt jedoch Einsatzzwecke, welche die dort enthaltenen Steuerelemente ans Tageslicht kommen lassen. Eines dieser Steuerelemente ist die MSForms-Textbox. Sie bietet gegenüber dem eingebauten TextBox-Steuerelement unter anderem den Vorteil, dass auch Texte mit mehr als 64.000 Zeichen angezeigt werden können. Die übrigen Vor- und Nachteile liefert der vorliegende Beitrag.

Problem Bearbeitung langer Texte

Ein Problem herkömmlicher Textfelder ist, dass Sie damit keine Texte bearbeiten können, die länger als 64.000 Zeichen sind. Dies können Sie reproduzieren, indem Sie eine Tabelle namens **tblTexte** mit den beiden Feldern **TextID** und **Inhalt** (Datentyp: **Langer Text/Memo**) anlegen.

Erstellen Sie dann ein Formular mit der Tabelle **tblTexte** als Datenherkunft und fügen Sie die beiden Felder aus der Feldliste zum Formularentwurf hinzu. Nun legen wir einen Text von knapp 64.000 Zeichen im ersten Datensatz der Tabelle an:

```
Public Sub EinLangerText()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strText As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblTexte", _
        dbOpenDynaset)
    rst.AddNew
    strText = String(63995, "A")
    rst!Inhalt = strText
    rst.Update
End Sub
```

Wechseln Sie dann zum Formular in der Formularansicht und fügen Sie noch ein paar Zeichen zum Text hinzu. Nach dem Hinzufügen von fünf Zeichen werden Sie keine neuen Zeichen mehr hinzufügen können – das Limit von 64.000 ist dann erreicht. Sie können auch nicht mehr als 64.000 Zeichen über eine **INSERT INTO**-Anweisung hinzufügen. Die einzige Möglichkeit ist der Einsatz der oben verwendeten Prozedur mit den DAO-Methoden **AddNew** und **Update** des **Recordset**-Objekts.

Das Ändern des Textes ist weder in der Datenblattansicht der Tabelle noch im Formular möglich.

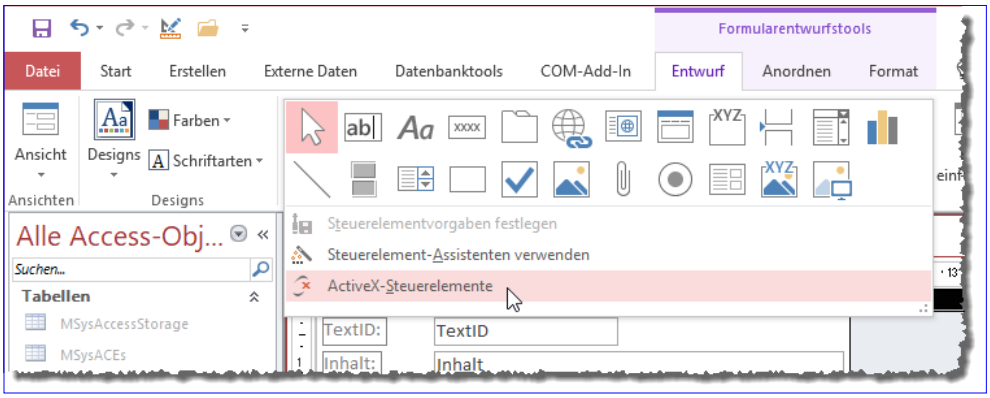


Bild 1: Öffnen des Dialogs zum Hinzufügen von ActiveX-Steuerelementen

Lange Texte in der MSForms-TextBox

Damit kommt der Einsatz der MSForms-TextBox. Diese fügen Sie hinzu, indem Sie im Ribbon zum Tab **Entwurf** wechseln und das Menü unter Steuerelemente aufklappen und den Eintrag ActiveX-Steuerelemente auswählen (s. Bild 1).

Daraufhin erscheint der Dialog aus Bild 2, mit dem Sie den Eintrag **Microsoft Forms 2.0 TextBox** selektieren und dieses per Mausklick auf die **OK**-Schaltfläche zum Formularentwurf hinzufügen können. Das neue Steuerelement wird dann relativ klein in der linken, oberen Ecke des Formulars angelegt – Sie dürften das vom **TreeView**-Steuerelement her kennen.

Die weiteren offensichtlichen Unterschiede zwischen diesem Steuerelement und dem eingebauten **TextBox**-Element sind das fehlende Bezeichnungsfeld sowie der 3D-Effekt für den Rahmen (s. Bild 3).

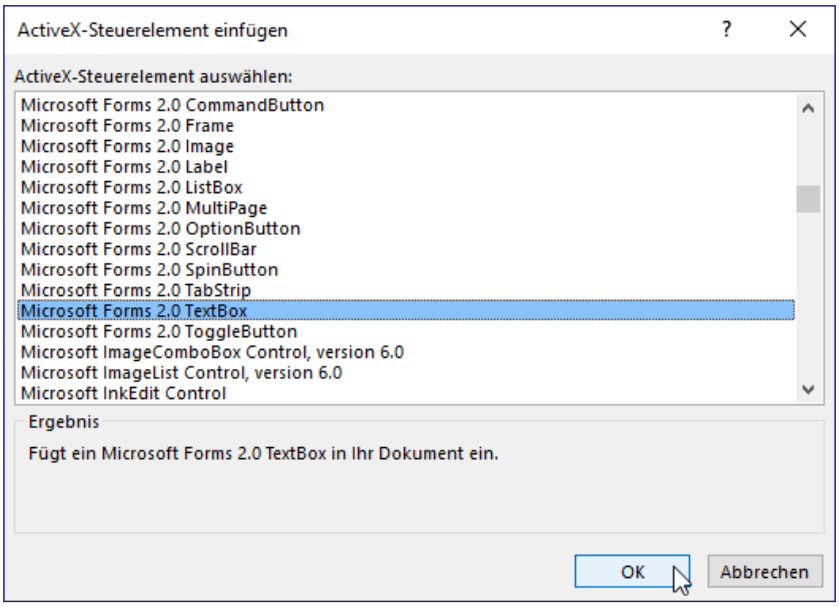


Bild 2: Der Dialog zur Auswahl von ActiveX-Steuerelementen

Wenn wir uns die Eigenschaften dieses Steuerelements ansehen, finden wir auf den meisten Registerseiten eine vergleichsweise geringe Menge an Eigenschaften vor. Die spezifischen Eigenschaften des Steuerelements finden wir in englischer Sprache auf der Registerseite **Andere** (s. Bild 4). Bevor wir uns die Eigenschaften unter VBA ansehen, benennen wir das Steuerelement noch schnell in **txtMSForms** um.

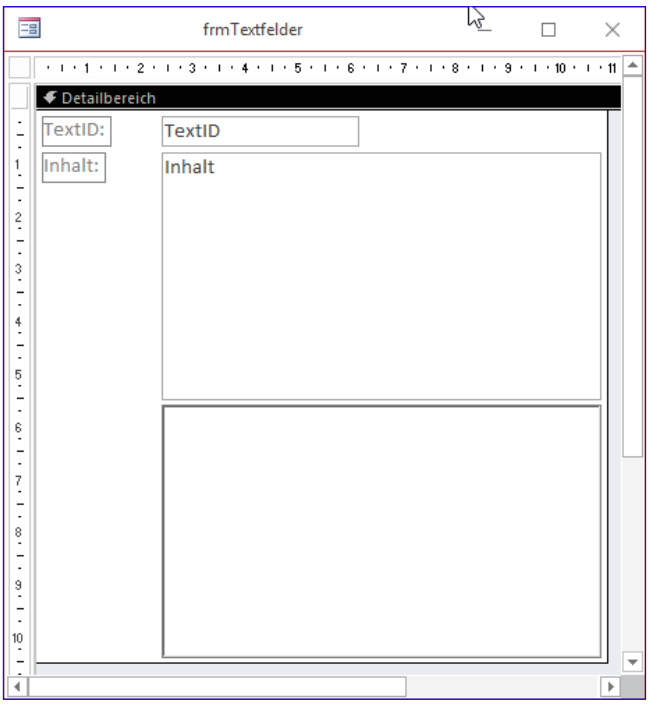


Bild 3: Das herkömmliche Textfeld und sein MSForms-Bruder

Danach können wir zum VBA-Editor wechseln, mit dem Menübefehl **AnsichtlObjektkatalog** (oder **F2**) den Objektkatalog öffnen und uns die Eigenschaften des **TextBox**-Objekts der Bibliothek **MSForms** ansehen (s. Bild 5). Hier finden Sie noch einige weitere Einträge gegenüber denen im Eigenschaftsfenster.

Aus eins mach zwei

Das ActiveX-Steuerelement besteht aus einem Container des Typs **CustomControl** sowie dem darin enthaltenen Steuerelement **TextBox**. Den Container referenzieren wir direkt über den Namen des Elements, das enthaltene Steuerelement über die Eigenschaft **Object**.

Dies verdeutlichen die folgenden beiden Codezeilen, mit denen wir den Typ der beiden Elemente im Direktbereich ausgeben:

```
Debug.Print TypeName(Me!txtMSForms)
```

```
Debug.Print
TypeName(Me!txtMSForms.Object)
```

Diese liefern nämlich:

```
CustomControl
TextBox
```

Eigenschaften per IntelliSense

Wenn Sie einfach **Me!txtMSForms.Object** und den Punkt im VBA-Editor eingeben, würden Sie vermutlich gern eine Liste der Eigenschaften und Methoden präsentiert bekommen.

Das ist aber nicht der Fall. Eine Alternative wäre es, eine Objektvariable für das TextBox-Element zu deklarieren und diese im **Form_Load**-Ereignis des Formulars zu füllen. Hier ist die Deklaration:

Bild 4: Eigenschaften der MSForms-TextBox

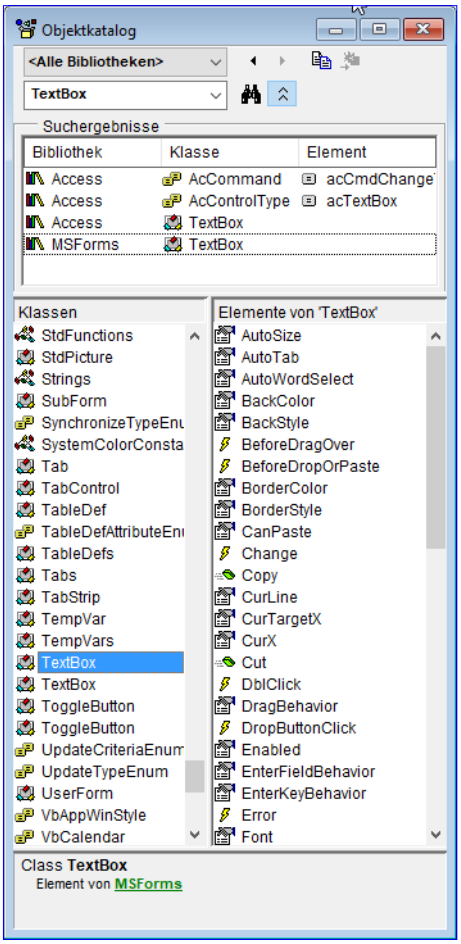


Bild 5: Eigenschaften unter VBA

```
Dim objMSForms As MSForms.TextBox
```

Und so stellen Sie die Objektvariable ein:

```
Private Sub Form_Load()
    Set objMSForms = Me!txtMSForms.Object
    With objMSForms
    '...
    End With
End Sub
```

Ereignisse

Die Menge der über das Eigenschaftsfenster verfügbaren Ereigniseigenschaften ist mit **Bei OLE Aktualisierung**, **Beim Hingehen**, **Beim Verlassen**, **Bei Fokuserhalt** und **Bei Fokusverlust** überschaubar. Der Grund für diese wenigen Ereignisse ist, dass dies die Ereignisse des **CustomControl**-Elements sind, das als Container für das eigentliche Steuerelement dient – diese Ereignisse finden Sie also auch bei den übrigen Steuerelementen der MSForms-Sammlung.

Wenn Sie die Ereignisse des eigentlichen Steuerelements nutzen wollen, müssen Sie den VBA-Editor zum Anlegen bemühen. Hier wählen Sie aus dem linken Kombinationsfeld den Namen des Steuerelements aus, hier **txtMSForms**. Im rechten Kombinationsfeld erscheinen dann die verfügbaren Ereignisse (s. Bild 6). Das Ereignis **txtMSForms_Change** wird hier automatisch bei Auswahl des Eintrags im linken Kombinationsfeld angelegt.

Selektieren in langen Texten

Wenn Sie einen Text in einem Textfeld selektieren wollen, erledigen Sie das in der Regel per Maus oder Tastatur. Das geht jedoch auch per Code – in der Regel durch das Setzen der Eigenschaften SelStart und SelLength. Damit lässt sich auch die Position der Markierung auslesen, wenn diese von Hand gesetzt wurde. Die genannten Eigenschaften haben jedoch einen Haken: Sie können damit nur in Texten mit bis zu 32.767 Zeichen arbeiten, da die Eigenschaften als Integer definiert sind. Abhilfe schafft die Windows API – alle notwendigen Informationen dazu finden Sie in diesem Beitrag.

Anlass für die Untersuchung dieses Problems ist die Lösung aus dem Beitrag **Volltextsuche mit Fundstellen** (**www.access-im-unternehmen.de/1119**), wo wir einen Suchbegriff an der entsprechenden Stelle im Text durch eine Markierung hervorheben wollen.

Die dort verwendeten Texte sind jedoch mitunter länger als 32.767 Zeichen, enthalten also mehr Zeichen, als durch den Datentyp **Integer** erfasst werden können.

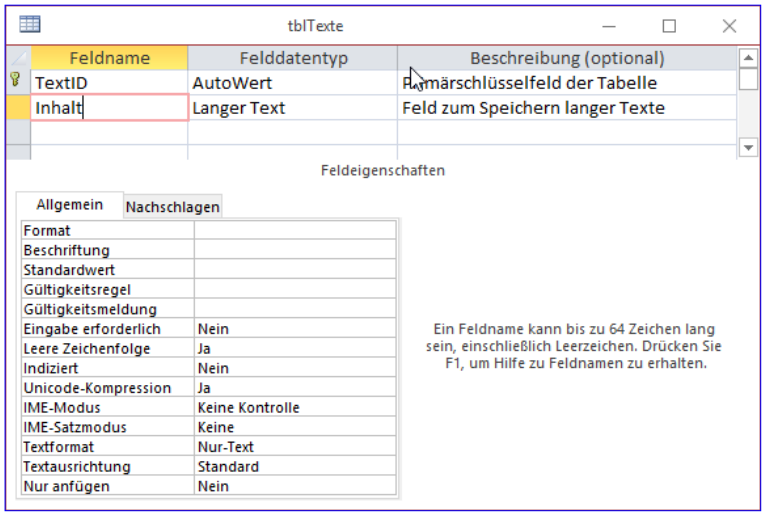


Bild 1: Tabelle zum Speichern langer Texte

Zum Setzen der Markierung wollten wir aber gern die beiden Eigenschaften **SelStart** und **SelLength** verwenden. Wenn wir in einem Textfeld mit einem langen Text eine Markierung setzen wollen, die hinter dem 32.767ten Zeichen liegt, lösen wir damit einen Überlauf-Fehler aus.

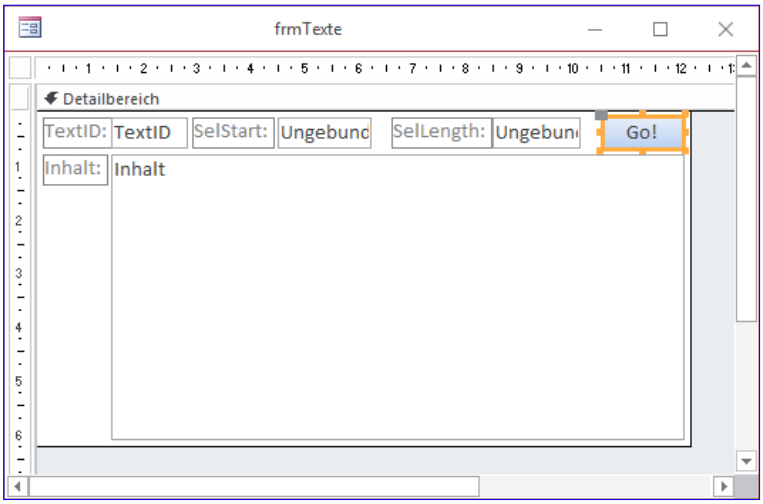


Bild 2: Entwurf des Testformulars

Als Spielwiese für die nachfolgend entwickelte Lösung erstellen wir eine Tabelle namens **tblTexte** mit dem Primärschlüsselfeld **TextID** und dem Feld **Inhalt** mit dem Datentyp **Langer Text** beziehungsweise **Memo** (s. Bild 1).

Das Formular, mit dem wir das Selektieren ausprobieren wollen, sieht im Entwurf wie in Bild 2 aus. Es ist an die Tabelle **tblTexte** gebunden. Damit wir möglichst viel Text sehen, haben wir die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** für das Textfeld, das an das

Feld **Inhalt** gebunden ist, auf **Beide** eingestellt. Das Textfeld haben wir außerdem **txtInhalte** benannt.

Oben finden Sie noch zwei Textfelder, in die wir die gewünschten Werte für **SelStart** und **SelLength** eintragen können und eine Schaltfläche namens **cmdGo**, welche die Markierung anwenden soll. Die Schaltfläche **cmdGo** soll die folgende Ereignisprozedur auslösen:

```
Private Sub cmdGo_Click()
    With Me!txtInhalt
        .SetFocus
        .SelStart = Me!txtSelStart
        .SelLength = Me!txtSelLength
    End With
End Sub
```

Wenn wir nun etwa die Werte **10** für **SelStart** und **20** für **SelLength** eingeben, erhalten wir im Textfeld eine Markierung wie in Bild 3.

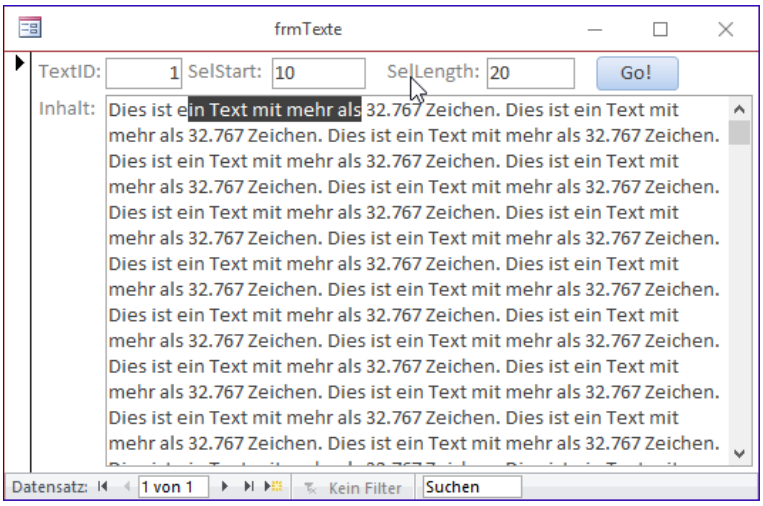


Bild 3: Setzen einer Markierung am Anfang des Textes

Nun geben wir einmal einen Wert größer als 32.767 für die Eigenschaft **SelStart** ein und klicken auf die Schaltfläche **cmdGo**. Dies löst nun den Fehler aus Bild 4 aus. Die Ursache für das Problem können wir uns gleich im Direktbereich nochmal bestätigen lassen.

Dort geben wir eine **Debug.Print**-Anweisung ein, die mit der Funktion **TypeName** den Datentyp der **SelStart**-Eigenschaft liefert:

```
Debug.Print TypeName(
            Me!txtInhalt.SelStart)
    Integer
```

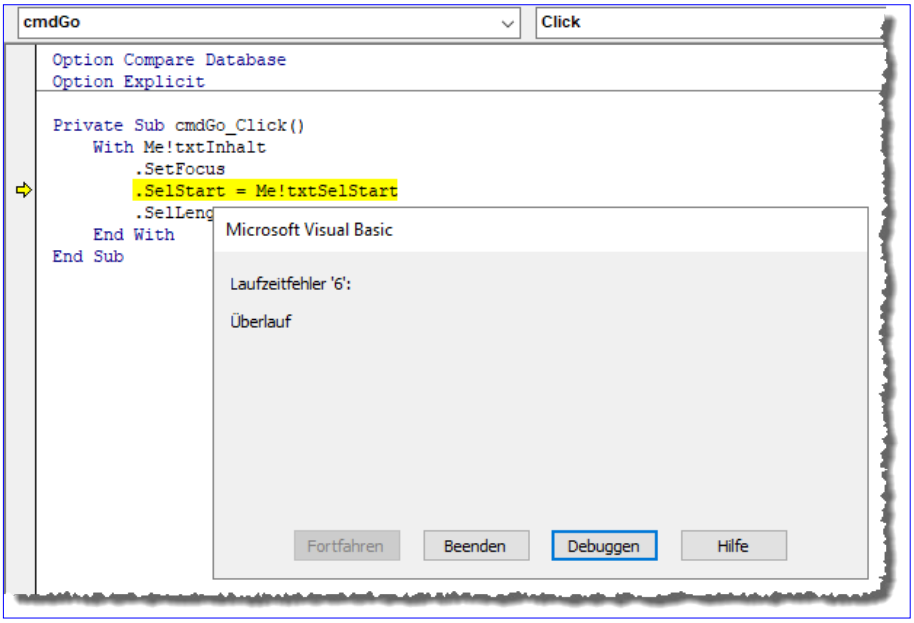


Bild 4: Fehler beim Einsatz von Werten größer als 32.767

Alternative: API-Funktionen

An dieser Stelle können wir also wohl nicht ändern und müssen uns eine Alternative überlegen.

Diese finden wir in den API-Funktionen von Windows.

Damit lassen sich die meisten Dinge bezüglich Benutzeroberfläche und Steuerelemente erledigen, die Sie auch über den Befehlssatz von Access/VBA einstellen können

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

Private Const EM_SETSEL  As Long = &HB1

Private Sub cmdGo_Click()
    Dim lngHandle As Long
    Dim lngSelStart As Long
    Dim lngSelLength As Long
    With Me!txtInhalt
        .SetFocus
        lngHandle = GetFocus
        lngSelStart = Me!txtSelStart
        lngSelLength = Me!txtSelLength
      SendMessage lngHandle, EM_SETSEL, lngSelStart, ByVal lngSelStart + lngSelLength
    End With
End Sub
```

Listing 1: API-Code zum Markieren eines Textabschnitts

– allerdings sieht das auf diesem Wege immer ein wenig komplizierter aus.

Für unsere Anforderung nutzen wir den Code aus Listing 1. Hier definieren wir zunächst zwei API-Funktionen, um diese innerhalb des VBA-Codes zu nutzen. Die erste API-Funktion heißt **SendMessage** und wird für das Senden aller möglichen Nachrichten verwendet.

Um anzugeben, um welche Art Nachricht es sich handelt, übergibt man mit dem zweiten Parameter eine Konstante für den Typ der Nachricht.

Der erste Parameter erwartet ein Handle auf das anzusprechende Element der Benutzeroberfläche, also eine Zahl, mit welcher das entsprechende Element referenziert wird. Die beiden folgenden Parameter nehmen je nach Aufgabe verschiedene Werte entgegen.

Die zweite API-Funktion, die wir nutzen wollen, heißt **GetFocus** und soll uns dabei unterstützen, das Handle für das anzusprechende Textfeld im Access-Formular zu ermitteln. Normalerweise haben Elemente wie Schaltflächen, Textfelder et cetera eine Eigenschaft namens **Handle** oder **hWnd**, um diesen Wert direkt zu ermitteln, aber unter Access läuft dies etwas anders.

Hier erhält nur das Steuerelement ein Handle, das aktuell den Fokus besitzt. Deshalb müssen wir erst den Fokus auf das Steuerelement einstellen und dann mit der API-Funktion **GetFocus** das entsprechende Handle ermitteln.

In der Prozedur **cmdGo** definieren wir nun zunächst entsprechende Variablen für das Handle und für die beiden zu übergebenen Parameter, also **lngSelStart** und **lngSelLength**. Dann setzen wir wie bereits im vorherigen Beispiel den Fokus auf das Textfeld und ermitteln mit der **GetFocus**-Funktion das Handle des Textfeldes, welches dann in der Variablen **lngHandle** landet.

Dann schreiben wir die Werte der beiden Textfelder **txtSelStart** und **txtSelLength** in die Variablen **lngSelStart** und **lngSelLength**.

Für den ersten Parameter der gleich aufzurufenden **SendMessage**-Funktion nutzen wir die Konstante **EM_SETSEL**.

HTML5 als Grafikkomponente

Wenn es um das Erzeugen von Grafiken geht, lässt Access Sie weitgehend im Regen stehen. Maximal die Anzeige von Bilddateien über das Bild- oder Anlagesteuerelement ist in Formularen und Berichten vorgesehen. Für alles darüber hinaus kommen Sie um den Einsatz aufwändiger Windows-API-Funktionen nicht herum. Wir zeigen hier aber, dass Sie auch mithilfe des Webbrowser-Steuerelements und HTML5 auf einfache Weise zu verblüffenden Ergebnissen kommen.

Die HTML5- und SVG-Erweiterungen von HTML

Grafikelemente können auf dynamischen Webseiten über zwei Schnittstellen erstellt werden. Die eine ist **SVG** und die andere **HTML5**. **SVG** ist eine auf **XML** basierende Auszeichnungssprache, die beliebige grafische Objekte mitsamt Effekten erzeugen und die resultierenden Anweisungen in einer Datei mit der Endung **svg** abspeichern kann.

Solche Dateien stellen Sie dann mit einem Grafikprogramm dar und bearbeiten sie dort auch, wobei sich vor allem **Inkscape** als Open-Source-Lösung etabliert hat. Seit geraumer Zeit beherrschen aber auch alle gängigen Webbrowser diese Schnittstelle, wobei sie keinesfalls auf das Laden solcher Dateien angewiesen sind, sondern die Grafikelemente auch direkt über Objekte aufbauen können. In der Regel kommt für die Steuerung **JavaScript** zum Einsatz.

Dasselbe gilt für **HTML5**. Auch hier gibt es vordefinierte Objekte, die Sie mit Leben füllen. Der wesentliche Unterschied zu **SVG** besteht allerdings darin, dass die damit getätigten Grafikoperationen nicht mehr rückgängig gemacht werden können. Während **SVG** eine Ansammlung von interaktiven (!) Grafikobjekten darstellt, die Sie später einzeln im Dokument ansprechen können, zeichnet **HTML5** alles passiv auf eine Malfläche (**canvas**), wie etwa das **Windows-GDI** auch. Dafür ist seine Handhabung aber auch deutlich einfacher – einer der Gründe, die Schnittstelle als erste zu betrachten. In einem Folgebeitrag wenden wir uns dann der alternativen **SVG**-Schnittstelle zu.

HTML5-Objektmodell

Für alles, was mit HTML und dem **Internet Explorer** zusammenhängt, stellt Microsoft die Bibliothek **MSHTML** bereit. Laden Sie diese über die Auswahl **Microsoft HTML Object Libary** in die Verweise Ihres VBA-Projekts. Im Objektkatalog erscheint sie danach unter dem Namen **MSHTML**.

Diese Bibliothek ist die umfangreichste, die Sie unter Windows überhaupt finden können. Zu jedem HTML-Element gibt es hier ein Pendant in Form einer Klasse. So entspricht ein Web-Dokument etwa der Klasse **HTMLDocument**, eine Tabelle dem **HTMLTable**, ein **div**-Bereich dem **HTMLDivElement**, und so weiter.

Jede dieser Klassen weist genau jene Eigenschaften und Methoden auf, die das **W3C**-Konsortium für das HTML-Element definiert hat, auch wenn jeder Browser-Hersteller hier sein eigenes Süppchen kocht und manchmal nur eine Teilmenge oder modifizierte Parameter implementiert sind. Die Angleichung ist aber in vollem Gange, so dass sich derzeit der Internet Explorer in seinem Verhalten nur noch marginal von anderen Browsern, wie Firefox oder Chrome unterscheidet.

Häufig wird der Text eines HTML-Dokuments über Strings aufgebaut. Das ist mit **MSHTML**, mit dem Sie den Internet Explorer oder das Webbrowser-Steuerelement fernsteuern, eigentlich nicht mehr notwendig. Sie erzeugen stattdessen neue Klasseninstanzen der Elemente unter VBA und verschachteln diese im **HTMLDocument** oder dessen **HTMLBody** über das Setzen bestimmter

Eigenschaftenparameter. Damit können Sie den HTML-Text komplett über Klassenobjekte erzeugen.

Das Stichwort **HTML5** selbst werden Sie indessen im Objektkatalog nicht finden. Seine Implementierung versteckt sich hinter jenen Klassen, die den String **Canvas** enthalten, wie etwa **HTMLCanvasElement**, **CanvasGradient**, **CanvasPattern**, **ICanvasPixelArray** oder **CanvasRenderingContext2D**. Letztere ist von besonderer Bedeutung, weil eben sie alle benötigten Grafikmethoden aufweist.

Bild 1 zeigt einen Ausschnitt der Methoden im Objektkatalog. An ihren Namen lässt sich schon ablesen, was sie bewirken sollen. **lineTo** etwa zeichnet eine Line vom aktuellen Ort zu den Koordinaten **x** und **y**. **arc** erzeugt einen Bogen. Mit den **Path**-Methoden legen Sie geschwungene Linienzüge an. **rect** malt ein Rechteck. Die **Fill**-Methoden erlauben das Füllen dieser Grafikobjekte mit beliebigen Farben oder Mustern, wobei diverse Verlaufsformen möglich sind (**Gradient**). Die Aufgabe besteht also darin, ein HTML-Dokument zu erzeugen, diesem ein **Canvas**-Element über eine VBA-Objektvariable zu verabreichen und anschließend über die Methoden das **CanvasRenderingContext2D** Zeichenoperationen darauf auszuführen. **JavaScript** entfällt. Die Ausgabe geschieht im eingebauten **Access-Webbrowsersteuerelement**.

Webbrowser-Steuerelement

HTML5 und **SVG** hielten mit der Version 9 des **Internet Explorers** Einzug und reflektierten sich auch gleich im **MSHTML**-Objektmodell. Das Webbrowser-Steuerelement hostet ja lediglich den Internet Explorer und kann damit praktisch alles, was der auch kann. Bis zur aktuellen Version 11 (**Edge** verwendet eine andere Technik und hat mit dem **Webbrowser Control** nichts zu tun; auch unter **Windows 10** nutzt das Control den Kompatibilitätsmodus des **IE11**) ereignete sich eine Evolution, die den Umfang der **W3C**-Implementierung kontinuierlich steigerte. Sie kann als weitgehend abgeschlossen betrachtet werden.

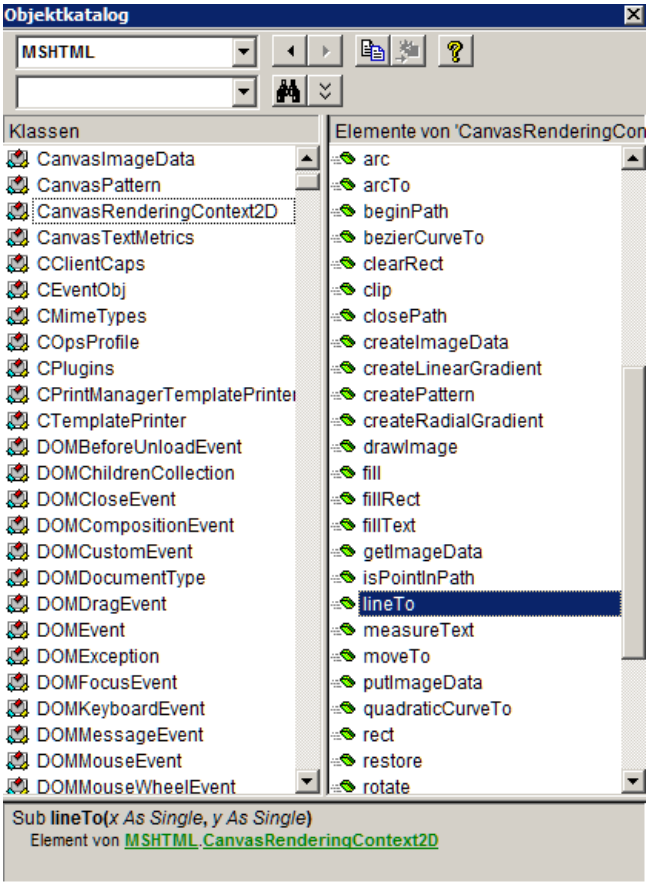


Bild 1: Die Zeichenoperationen der HTML5-Canvas-Klasse im Objektkatalog

Dumm leider, dass das Steuerelement im Urzustand nur den **Internet Explorer 7** hostet. Dies bedeutet, dass Microsoft aus Sicherheitsgründen den Internet Explorer im Steuerelement in den Zustand dieser alten Version versetzt. Und diese kennt überhaupt noch keine **HTML5**- oder **SVG**-Elemente! In diesem Modus ignoriert das Control schlicht alle entsprechenden Anweisungen. Zum Glück gibt es aber einen Workaround, den Sie schon im Beitrag der Ausgabe **3/2017 GoogleMaps als Kartendienst** (**Shortlink 1089**) kennenlernten. Mithilfe des Moduls **mdlWebControl** und dem Aufruf der Prozedur **SetWebControlAsIE11** oder der übergeordneten Prozedur **PrepareBrowser** können Sie über API-Funktionen erzwingen, dass der Internet Explorer im Webbrowser-Steuerelement in den Modus der Version 11 übergeht und zusätzlich noch weitere Limitierungen aufgehoben werden. Das alles ist legitim. Dasselbe Modul findet nun

auch in der Beispieldatenbank **HTMLImage.accdb** zum vorliegenden Beitrag Anwendung. Von einer Funktionsbeschreibung sehe ich an dieser Stelle ab, weil dazu alles erschöpfend bereits im erwähnten Beitrag erläutert ist.

Dergestalt vorbereitet kann das Webbrowser-Steuerelement dann auch mit **HTML5** und **SVG** umgehen.

Die erste HTML5-Grafik

Um die grundlegenden Vorgänge zu beleuchten, die die Anlage einer **HTML5**-Grafik im Webbrowser-Steuerelement erfordern, laden Sie das Formular **frmHTML5** der Beispieldatenbank. Zunächst sehen Sie hier nur einen augenscheinlich leeren Detailbereich, sowie einen Button rechts oben im Formularkopf. Nach dessen Betätigung kommt es zur Ausgabe der ziemlich sinnfreien Grafik in Bild 2. Aber immerhin demonstriert das Beispiel einige Grafikelemente, die andernorts vielleicht nützlicher verwandt werden könnten.

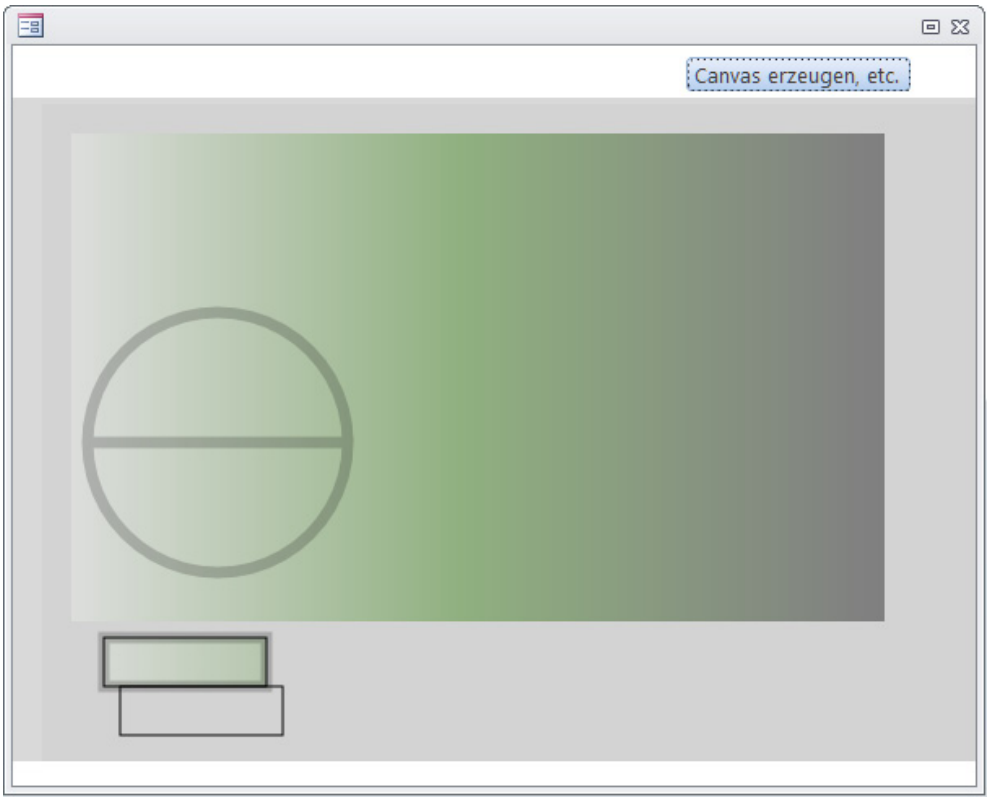


Bild 2: Diese Grafik im Formular **frmHTML5** wird mit nur recht wenigen Zeilen VBA-Code erzeugt

Im Entwurf des Formulars finden Sie das Webbrowser-Steuerelement mit dem Namen **ctlWeb** im Detailbereich. Es füllt diesen fast ganz aus und hat die Einstellung **Nach unten und quer dehnen** für die Eigenschaft **Verankern**. Damit macht das Steuerelement alle Größenänderungen des Formulars mit. Der Steuerelementinhalt ist an kein Tabellenfeld gebunden, sondern auf den festen Wert **="about:blank"**. Ohne diese Einstellung fänden Sie nämlich nach dem Laden des Formulars im Webbrowser die unschöne Ausgabe **Die Adresse ist ungültig vor**. **about:blank** hingegen rendert gleich zu Beginn eine leere Fläche und legt ein rudimentäres Dokument an, mit dem Sie anschließend weiterarbeiten können, um ihm die HTML5-Elemente unterzujubeln. Schauen wir uns dazu einmal den Code der Formularklasse an. Beim Laden des Formulars wird zunächst diese Zeile abgesetzt:

```
Me!ctlWeb.Object.Silent = True
```

Aufgepasst! Das Webbrowser-Steuerelement schleift keineswegs alle Methoden des zugrundeliegenden **Internet Explorer-**Objekts durch. Deshalb muss dessen Instanz erst über die Eigenschaft **Object** des Steuerelements erhalten werden, ähnlich, wie auch bei ActiveX-Steuerelementen üblich. Und eine seiner Eigenschaften ist **Silent**, die hier auf **True** eingestellt wird. Das weist ihn an, jegliche Meldungen zu fehlerhaften Skripten oder Ähnliches zu unterlassen. Das hat zwar direkt nichts mit unserem HTML5-Vorhaben zu tun, ist jedoch grundsätzlich sinnvoll, wenn

Sie dem Anwender Ihrer Datenbank im Zweifel nicht die Herkunft des dargestellten Inhalts verraten möchten ...

Der Klick auf den Button **cmdCanvas** ruft nun die Ereignisprozedur in Listing 1 auf. Hier wird zunächst die Objektvariable **oDoc** vom Typ **HTMLDocument** (**MSHTML**-Bibliothek) auf das im Webbrowser-Steuerelement befindliche leere Dokument gesetzt. Der Hintergrund wird dann über das **style**-Attribut **bgColor** des **body**-Elements auf hellgrau (**lightgray**) eingestellt. In der Abbildung wird das durch die leicht graue Umrandung der Grafik deutlich. Das Setzen solcher Attribute wirkt sich übrigens immer unmittelbar im Browser aus. Bis hierhin hat das Ganze noch nichts mit **HTML5** zu tun, was sich jedoch in der folgenden Zeile ändert. Die erzeugt über die Methode **createElement** des Dokuments ein allgemeines **HTMLElement** vom Typ **canvas**. Unterstützt der Browser **HTML5** nicht, etwa, weil sein Modus noch unverändert auf der Version 7 des Internet Explorers beruht, so ereignet sich hier kein Fehler, sondern die Objektvariable **oElement** hat schlicht den Wert **Nothing** oder aber sie hat keinen spezifischen Typ. Deshalb fragt die nächste Zeile über die Hilfsroutine **IsCanvasSupported** noch ab, ob das **Canvas**-Element erfolgreich angelegt wurde und verlässt andernfalls die Routine.

Dem **Canvas**-Element wird nun die eindeutige **Id zeichnung** verabreicht, damit später gegebenenfalls auf es Bezug genommen werden kann. Da die Variable **oElement** nur vom allgemeinen Typ **HTMLElement** ist, unterstützt Intellisense von VBA auf sie nicht die speziellen Methoden des **Canvas**-Elements. Deshalb kommt die Hilfsvariable **oCanv** ins Spiel, die dezidiert den Typ **IHTMLCanvasElement** aufweist. Sie wird direkt auf die Variable **oElement** gecastet. Kleiner Einschub zur Hilfsroutine **IsCanvasSupported**: Die Funktion ermittelt einfach den Typ des in der Variablen **oElement** gespeicherten HTML-Elements:

```
If TypeName(oElement) <> "HTMLCanvasElement" Then
...(IsCanvasSupported = False)...  Else = True ...
```

```
Private Sub cmdCanvas_Click()
    Dim oDoc As HTMLDocument
    Dim oElement As IHTMLElement
    Dim oCanv As IHTMLCanvasElement
    Dim CTX As ICanvasRenderingContext2D
    Dim oGrad As ICanvasGradient

    Set oDoc = Me!ctlWeb.Object.Document
    oDoc.body.setAttribute "bgColor", "lightgray"

    Set oElement = oDoc.createElement("canvas")
    If Not IsCanvasSupported(oElement) Then Exit Sub

    oElement.Id = "zeichnung"
    Set oCanv = oElement
    oCanv.Width = 600
    oCanv.Height = 400
    oDoc.body.appendChild oElement

    Set CTX = oCanv.getContext("2d")
    With CTX
        Set oGrad = .createLinearGradient(0, 0, 500, 0)
        oGrad.addColorStop 0, "#e0e0e0"
        oGrad.addColorStop 0.5, "#90b080"
        oGrad.addColorStop 1, "#808080"
        .FillStyle = oGrad
        .fillRect 10, 10, 500, 300
        .RECT 30, 320, 100, 30
        .Fill
        .stroke
        .strokeRect 40, 350, 100, 30
        .globalAlpha = 0.2
        .moveTo 20, 200          '180, 200
        .lineWidth = 7
        .arc 100, 200, 80, 0, 6.4, 0
        .stroke
    End With
    Debug.Print oDoc.all(0).outerHTML
End Sub
```

Listing 1: Komplette Routine zur Anlage der **HTML5**-Grafik

Nun haben wir zwar ein Element vom Typ **Canvas** angelegt, doch dieses ist damit noch lange nicht Bestandteil des Dokuments, sondern existiert rein im Speicher. Die Methode **appendChild** des **Body**-Objekts erledigt dies in der Folge aber.

Das **Canvas**-Element selbst kennt noch keinerlei Methoden zum Zeichnen. Nur der sogenannte **Render-Kontext** kann das. Sie erhalten diesen über den Methodenausdruck **getContext("2D")**. Das resultierende Objekt ist vom Typ **ICanvasRenderingContext2D** und wird in der Objektvariablen **CTX** abgespeichert. Der Parameter **2D** ist übrigens der einzige, der hier aktuell erlaubt ist. Definiert ist über **W3C** zwar auch der Parameter **3D** für dreidimensionale Grafiken, doch der Internet Explorer unterstützt das, wie viele andere Browser auch, noch gar nicht.

Die Vorarbeiten sind damit abgeschlossen und es kann nun direkt an die Zeichenoperationen gehen. Da derer mehrere folgen, ist die Variable **CTX** mit einem **With**-Block versehen, der fast bis ans Ende der Prozedur reicht.

Wir picken uns aus der Reihe der Anweisungen die Methode **fillRect** heraus. Die erzeugt ein gefülltes Rechteck. Als Parameter erwartet sie die Koordinaten der linken oberen Ecke, sowie Breite und Höhe des Rechtecks. Allein ausgeführt resultierte sie in einem komplett schwarzen Rechteck. Denn Angaben etwa zur Linienstärke der Umrandung und zur Art und Farbe der Füllung werden der Methode ja nicht übergeben. Die Methode greift nämlich auf den Kontext des **Canvas** zurück, in dem bereits einige Voreinstellungen zu diesen Stilen festgelegt sind. Um diese Stileinstellungen zu ändern, bedienen Sie sich weiterer Methoden des Objekts, und das übernehmen die Zeilen vor dem Aufruf von **fillRect**.

Soll ein Grafikelement nicht nur einfarbig gefüllt werden, sondern mit einem Verlauf, so muss die Eigenschaft **FillStyle** des Grafikkontexts auf ein Verlaufsobjekt gesetzt werden. Dieses Objekt vom Typ **ICanvasGradient**, zugewiesen der Variablen **oGrad**, erzeugt die Prozedur über die Methode **createLinearGradient**. Sie erwartet als Parameter zwei Punktkoordinaten in der Einheit **Pixel**, die der gewünschten Ausdehnung des Verlaufs entsprechen. Hier ist das einerseits die linke obere Ecke (**0,0,**) und dann die rechte obere (**500,0**). Der Verlauf erstreckt sich damit von links bis fast nach rechts. Sie können hier über andere Werte natürlich auch einen diagonalen Verlauf erzeugen, der problemlos auch über den Rand des Canvas hinausgehen darf. Zu erwähnen wäre an dieser Stelle auch, dass die alternative Methode **createRadialGradient** einen kreisförmigen Verlauf erzeugen könnte.

Anschließend müssen die Farben für den Verlauf bestimmt werden, was die Methode **addColorStop** ermöglicht. Sie erwartet zwei Parameter: Der erste gibt dabei den Abstand innerhalb des Verlaufs an, auf den die Farbe wirken soll. Der Wertebereich des **Single**-Werts erstreckt sich von **0** bis **1**. **0** bedeutet in unserem Beispiel ganz links, **1** wäre ganz rechts, also bei der Breite **500** Pixel. Mit **0,5** treffen Sie genau die Mitte der Fläche, die sich bei der Breite **250** Pixel befindet. Es können beliebig viele solcher Farbbereiche durch Wiederholung der Methode **addColorStop** angelegt werden. Wenn Sie wollen, also ein ganzes buntes Farbspektrum. Die Farben selbst geben Sie jeweils im zweiten Parameter als hexadezimalen String an, wie sonst auch für solche HTML-Farbattribute üblich. Unsere drei Farbbereiche sind auf hellgrau, grün und dunkelgrau gesetzt. Die Methode **fillRect** füllt das Rechteck nun mit diesem im Kontext neu bestimmten Verlauf. Es nimmt fast den gesamten Raum des **Canvas** ein.

Es gibt noch eine alternative Methode, ein gefülltes Rechteck zu erzeugen. Das machen die nächsten drei Zeilen deutlich. Die Methode **RECT** legt zunächst ein Rechteck auf dem Hintergrund an, welches ohne Füllung daherkommt. Ein einfacher Aufruf von **fill** ändert dies. Sie füllt die zuletzt angelegten Grafikobjekte mit der im Kontext weiter oben festgelegten Verlaufseigenschaft. Zu sehen bekommen Sie das tatsächlich jedoch erst, nachdem Sie die Methode **stroke** aufrufen. Sie erst rendert alle zuvor angelegten Grafikobjekte in den **Canvas**. Im Klartext heißt das, dass sie mehrere Grafikobjekte, etwa auch Kreise (**arc**), hintereinander erzeugen können, sie auf einen Schlag mit **fill** füllen und dann ebenso

komplett über **stroke** physisch anzeigen können. In der Abbildung sehen Sie das verlaufsgefüllte kleine Rechteck links unten.

Darunter befindet sich noch ein ungefülltes. Das könnte ebenfalls über die Methode **RECT**, gefolgt von **stroke**, erzeugt werden. Einfacher aber ist die kombinierende Methode **stokeRect**, welche das auf einen Schlag erledigt.

Nun fehlt noch der Kreis in der Abbildung, der über die Methode **arc** erzeugt wird, welche auch Teilbogen darstellen kann. Die Reihenfolge der Parameter: Erst die **x**- und **y**-Koordinaten des Mittelpunkts (**100, 200**), dann der Radius – alle Angaben in Pixel. Darauf folgen Start- und Endwinkel im Bogenmaß (**0, 6.4**). **6.4** entspricht ungefähr zweimal **Pi**, weshalb ein kompletter Kreis gemalt wird. Der Winkel **0** befindet sich übrigens ganz rechts vom Mittelpunkt und die Zeichenrichtung geschieht im Uhrzeigersinn, wenn der allerletzte Parameter der Methode auf **0** steht. Mit dem Wert **1** würde gegen den Uhrzeigersinn gezeichnet.

Sie können mit dieser Methode also beliebige Kreisabschnitte erzeugen, nicht jedoch ellipsenförmige. Die Breite des Zeichenstifts wurde zuvor über die Eigenschaft **lineWidth** des Kontexts auf 7 Pixel eingestellt. Wo aber kommt die Querlinie her, die den Kreis zu einem Stoppschild macht? Eine weitere Methode zum Linienzeichnen findet sich ja in der Routine nicht.

Grund dafür ist die Tatsache, dass dich der Canvas den Endpunkt der letzten Zeichenoperation hernimmt und ihn als Startpunkt der nächsten verwendet. Und die letzte Aktion war das Rechteck ganz unten. Der Startpunkt des Kreises begänne damit bei dessen Ecke links oben. Genauer: Es würde zusätzlich eine gerade Linie von dieser Ecke zum Kreisanfang gezeichnet werden. Auf diese Weise kommt es durch mehrere aufeinanderfolgende Grafikoperationen zu einem Linienzug. Der kann aber durch die Methode **moveTo** unterbrochen werden. In der Prozedur wird der Stift gleichsam zur Koordinate 80 Pixel links vom Mittelpunkt des Kreises bewegt (**x=100, r=80, r-x = 20**). Von hier aus entsteht dadurch eine waagrechte Linie zum Startpunkt des Kreises rechts. Die ganze Geschichte wirkt sich auch hier erst dann sichtbar aus, nachdem die Methode **stroke** aufgerufen wird.

Eine weitere **Style**-Eigenschaft des **Canvas** ist außerdem **globalAlpha**. Die entspricht unter HTML sonst dem Attribut **opacity** und gibt den Grad der Durchsichtigkeit der folgenden Grafikoperationen an. Der Kreis ist deshalb durchscheinend. Dieses Feature, den Alpha-Kanal zu beeinflussen, ermöglicht unter HTML5 überlagernde Grafikelemente! **globalAlpha** hat allerdings einen etwas eigentümlichen Wertebereich. **1** bedeutet opak, **0** völlig durchsichtig. Doch **0.5** bedeutet noch keinesfalls halbdurchsichtig! Dies kommt eher durch einen Wert von **0.3** zustande. Selbst beim Wert **0.01** ist der Kreis noch deutlich zu erkennen. Offenbar verwendet **HTML5** hier eine exponentielle Kurve.

Sie sehen, dass sich über **HTML5** mit nur wenigen Zeilen Code Grafikelemente erzeugen lassen, die mit dem **Windows-API** (**GDI32** oder **GDIPlus**) hunderte Zeilen erfordert hätten! An dieser Stelle kann nicht jede Methode des **Canvas** und deren Parameter erläutert werden. Die Referenz dazu finden Sie in der **MSDN** von Microsoft (**https://msdn.microsoft.com/de-de/library/ff976025(v=vs.85).aspx**). Hier ging es zunächst nur um die grundliegenden Schritte, die zu einer **HTML5**-Grafik im Webbrowser-Steuerelement führen. Die letzte Zeile der Routine gibt den erzeugten Quelltext des Dokuments über die Methode **outerHTML** im VBA-Direktfenster aus. Und der ist reichlich übersichtlich:

```
<html>
  <head></head>
  <body bgcolor="lightgray">
    <canvas width="600" height="400" id="zeichnung"></canvas>
  </body>
</html>
```

Hier wird offenbar, dass die Zeichenoperationen sich, anders, als bei **SVG**, in keiner Weise im Quelltext wiederspiegeln. Sie lassen sich durch erneutes Laden des Quelltexts nicht mehr reproduzieren. Die erstellte Grafik ist ein reines Bitmap im Browser. Wie Sie an die Daten dieses Bitmaps herankommen, um sie in einer Datei oder direkt in Tabellenfeldern abzuspeichern, folgt später. Neben Grafikelementen kann **HTML5** natürlich auch Texte ausgeben. Beides kombiniert eröffnen sich weite Anwendungsbereiche. Denken Sie an Flussdiagramme, technische Zeichnungen oder auch spezielle Grids, die sich aus den Tabellen der Datenbank speisen. Der einzige Nachteil von **HTML5** ist die fehlende Interaktivität. Sie können etwa nicht auf ein Grafikelement klicken und es über ein Ereignis identifizieren. Dafür brauchen Sie die in einem Folgebeitrag dargestellte Alternative **SVG**.

Diagramme per HTML5 ausgeben

Das Grundwissen ist nun vermittelt und wir schreiten zu einem nützlicheren Beispiel, dem Anlegen von Balken- und Tortendiagrammen über **HTML5**. Implementiert ist das im Formular **frmHTML5Diagram** der Beispieldatenbank (siehe Bild 3).

Dessen Funktionen sind schnell beschrieben. Nach dem Öffnen finden Sie im Detailbereich zunächst eine weiße Fläche vor, bei der es sich tatsächlich um ein Webbrowser-Steuerelement handelt. Im Formularkopf sind einige Controls untergebracht, die die Anlage der Diagramme steuern. Sie können zum einen wählen, ob ein Balken- oder ein Tortendiagramm generiert werden soll. Zum anderen ermitteln Abfragen auf eine Kunden- und Bestelldatenbank die aggregierten Umsätze nach Monaten oder die Umsätze nach Herkunftsländern der Besteller.

Je nach Auswahl wird dann ein anderes Diagramm über den Button links oben erzeugt. Das fertige Diagramm können Sie über die Schaltflächen rechts als Grafikdatei abspeichern oder auch direkt ausdrucken.

Das zugehörige an die bekannte **Nordwind**-Datenbank angelehnte Datenmodell zeigt Bild 4. Es gibt Kunden (**tblKunden**), die Artikel (**tblArtikel**) bestellen. Die Verknüpfung geschieht über die **1:n-Tabelle tblBestellungen**. Jeder Kunde kann ja mehrmals bestellen. Jede Bestellung wiederum kann mehrere Artikel enthalten, weshalb hier zusätzlich die **n:m-Tabelle tblBestellDetails** benötigt wird.

Zur Auswertung der Umsätze pro Zeiteinheit sind nach diesem Modell alle drei rechten Tabellen zu befragen, denn jene ergeben sich aus der Summe des Felds **Einzel-**

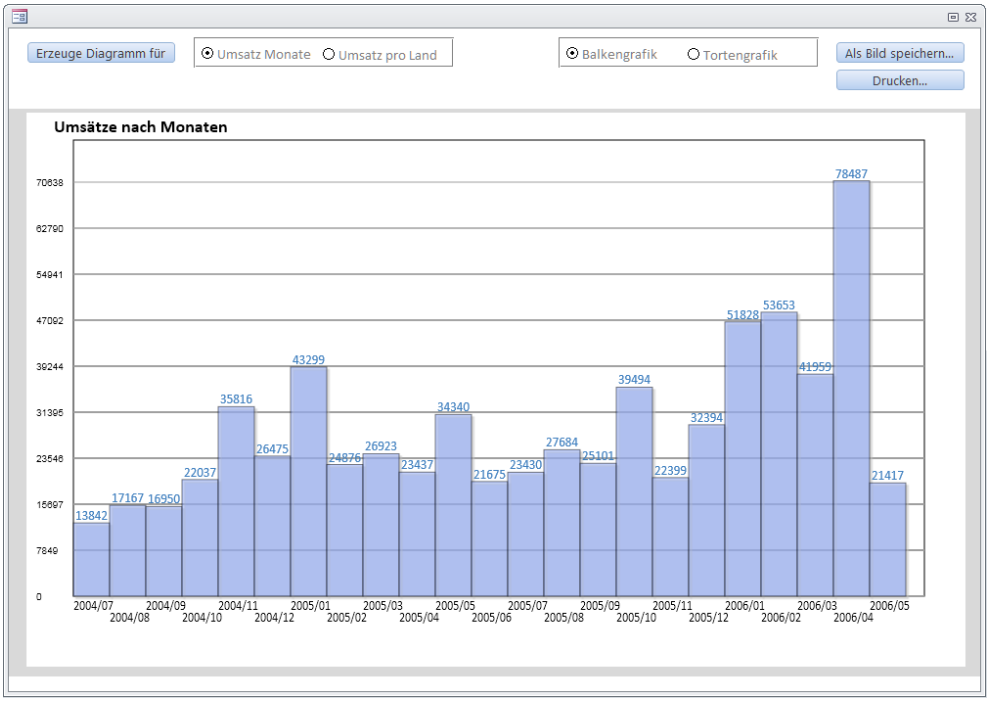


Bild 3: Das HTML5-Balkendiagramm zu Umsätzen nach Monaten im Formular **frmHTML5Diagram**

preis in der Artikeltabelle. Das ist warenwirtschaftlich zwar nicht korrekt, weil im richtigen Leben der Bestellpreis eigentlich zusätzlich in der Tabelle **tblBestelldetails** stehen müsste – beim Versand könnte sich der Artikelpreis ja bereits geändert haben

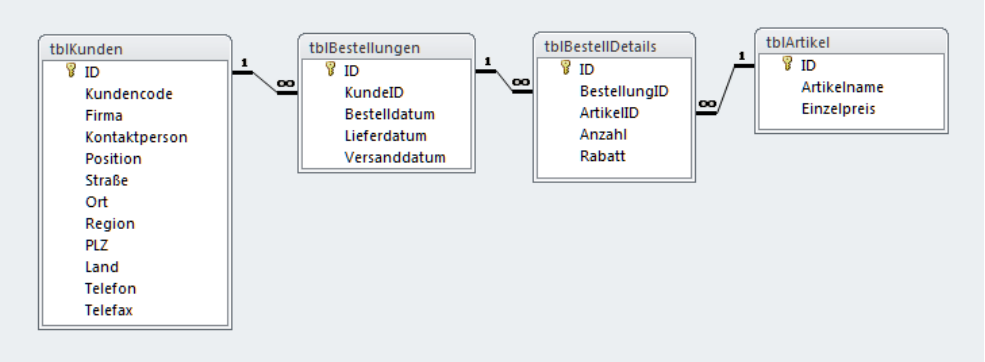


Bild 4: Am Datenmodell zu Kunden, Bestellungen und Artikeln sind vier relationale Tabellen beteiligt

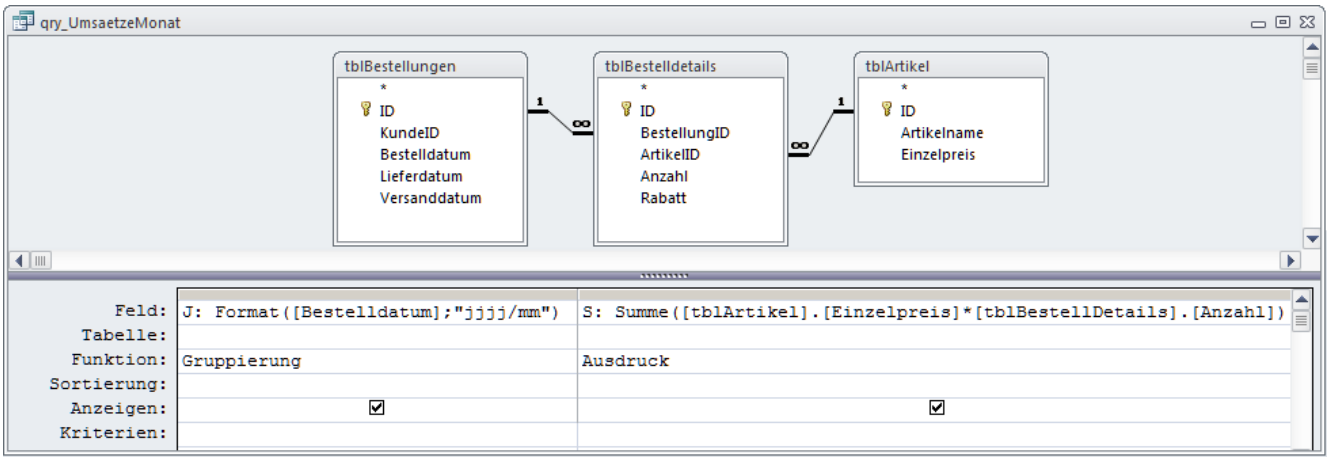


Bild 5: Zur Auswertung der Umsätze nach Monaten für die Diagramme kommt die Gruppierungsabfrage **qry_UmsatzMonat** zum Einsatz

und die Rechnung enthielte dann einen falschen Betrag. Dieses uralte Modell kam so aber bereits in verschiedenen Beispieldatenbanken von **Access im Unternehmen** zum Einsatz, weshalb wir es hier einfach unverändert übernehmen.

Wie auch immer, die zur Umsatzauswertung gehörige Abfrage stellt Bild 5 dar. Gruppiert wird hier nach dem **Bestelldatum**, wobei die **Format**-Funktion Jahre und Monate zusammenfasst. Für den **24.11.2015** etwa ergäbe sich damit der Textausdruck **2015/11**. Der Umsatz ergibt sich aus der Summe der **Einzelpreise** multipliziert mit der

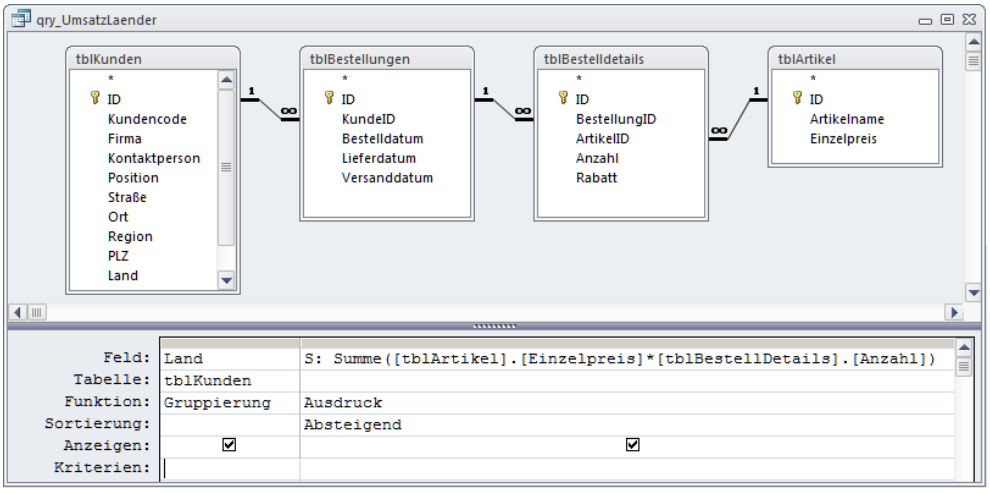


Bild 6: Zur Auswertung der Umsätze nach Land für die Diagramme kommt die Gruppierungsabfrage **qry_UmsatzLaender** zum Einsatz

Mit HTML5 zeichnen und malen

Wie Sie mithilfe des Webbrowser-Steuerelements und HTML5 programmgesteuert Grafiken erzeugen können, beleuchtete bereits unser Beitrag 'HTML5 als Grafikkomponente'. Dass sich mit diesem Gespann aber Bilder auch interaktiv anlegen lassen, zeigen wir in diesem zweiten Teil anhand einer kleinen Zeichenanwendung, die ohne externe Komponenten auskommt.

Referenz

Die Ausführungen dieses Beitrags setzen jene aus dem Artikel **HTML5 als Grafikkomponente** fort. Dort sind die Grundlagen zum Einsatz des **Webbrowser-Steuerelements** im Verein mit **HTML5** beschrieben, auf die hier nicht noch einmal eingegangen wird.

MS Access als Malprogramm?

Bilder und Grafiken scheinen, wie man den Beiträgen verschiedener Foren entnehmen kann, für Access-Entwickler ein immer wiederkehrendes Thema zu sein. Sicher ist die Anzeige von Bilddateien in Formularen und Berichten, sei es über das Laden aus dem System, über **OLE-Objekte** oder **Anlagefelder**, eine häufige Anforderung. Das reicht vom Logo im Berichtskopf über ein Konterfei im Mitarbeiterdatensatz bis hin zu technischen Zeichnungen. Gerade für letztere eignet sich **HTML5** gut, weil Sie hier eben nicht nur statisch Grafiken aus Dateien laden können, sondern in diese per VBA zusätzliche Elemente einbauen können. Oder Sie erstellen die Grafiken komplett neu, wie die Diagramme im Beitrag **HTML5 als Grafikkomponente** zeigen.

Das Problem bei **HTML5** ist allerdings, dass die erzeugten Grafiken gerenderte Bitmaps sind, auf deren Elemente Sie später keinen Einfluss mehr haben. Ein Undo ist schlecht realisierbar. Möglich ist aber die sequenzielle Bearbeitung der Grafik, die auch interaktiv geschehen kann. Eben davon macht unser Malprogramm Gebrauch.

Doch wer erstellt unter Access überhaupt Gemälde? Ein Beispiel wären technische Zeichnungen, die vervollständigt werden. Als Anbieter von Raumausstattungen etwa könnten Sie einen Raum gestalten lassen, indem Sie dessen leere Ansicht laden und in diese interaktiv mit der Maus Elemente hineinziehen und positionieren.

Sie besitzen ein mobiles Gerät, wie etwa ein Windows-Tablet? Dann können Sie mit diesem etwa jene Terminals von DHL oder DPD nachbilden und in den Grafikbereich eines Formulars Unterschriften eingeben lassen, um sie in Tabellen abzuspeichern.

Günstig sind in diesem Fall jedenfalls Geräte mit einem Touch-Screen, seien es Tablets oder Laptops. Natürlich erweitert ein Grafiktablett Ihren PC ebenfalls entsprechend.

Malen im Formular

Nach dem Start der Beispieldatenbank **HTMLImage2.accdb** finden Sie das Formular **frmHTML5Draw** zur Laufzeit vor, wie in Bild 1. Hier können Sie pro Datensatz ein Bild anzeigen lassen oder zeichnen. Ist ein Bild gespeichert, so wird es automatisch aus einer Tabelle geladen, während bei einem neuen Datensatz ein leeres Blatt auf Sie wartet. Wenden wir uns zunächst den Funktionen des Formulars und seiner Steuerelemente zu.

Die Grafikfläche im Detailbereich besteht aus einem **Webbrowser-Steuerelement**, in dessen Eigenschaft **URL** der String **="about:blank"** hinterlegt ist, damit es keinen Navigationsfehler ausgibt, sondern eine weiße Fläche. Der Zeichnung können Sie oben einen Namen geben, der sie eindeutig identifiziert. Zeichnung und Namen gelangen beide in eine Tabelle, an die das Formular gebunden ist. Das Speichern geschieht allerdings nicht

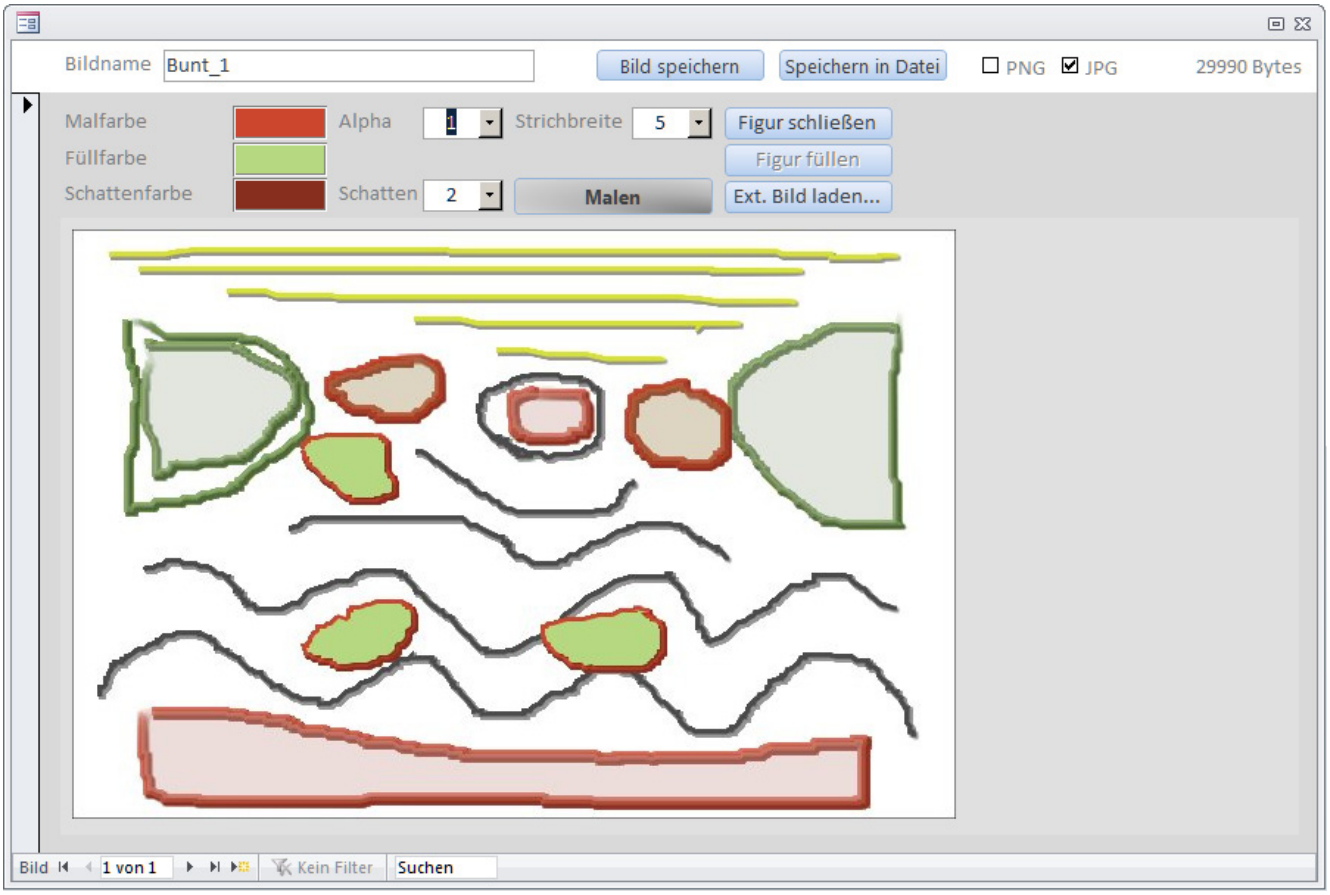


Bild 1: So präsentiert sich die Malanwendung im Formular **frmHTML5Draw** der Beispieldatenbank mit einer bereits geladenen Grafik

automatisch mit Verlassen des Datensatzes, sondern nur mit Klick auf den Button **Bild speichern**. Alternativ kann die Grafik auch in eine Datei exportiert werden, wobei Ihnen die Formate **PNG** und **JPG** zur Auswahl stehen, welche Sie mit den beiden Checkboxen einstellen. Übrigens beeinflusst diese Auswahl auch die interne Speicherung des Bilds in der Tabelle. Das Label rechts oben gibt Auskunft über den dafür benötigten Speicherbedarf.

Mit der Maus zeichnen Sie nun in die Grafikfläche. Die Farbe des Stifts stellen Sie mit Klick auf das Rechteck **Malfarbe** ein, die Füllung einer Figur mit dem Rechteck **Füllfarbe** und einen etwaigen Schattenwurf der Linien mit der **Schattenfarbe**. Der Klick auf diese Rechtecke ruft jeweils den Windows-Farbauswahldialog auf (Bild 2). In der Combobox **Strichbreite** wählen Sie die gewünschte Dicke des Stifts aus und dessen Deckkraft in der

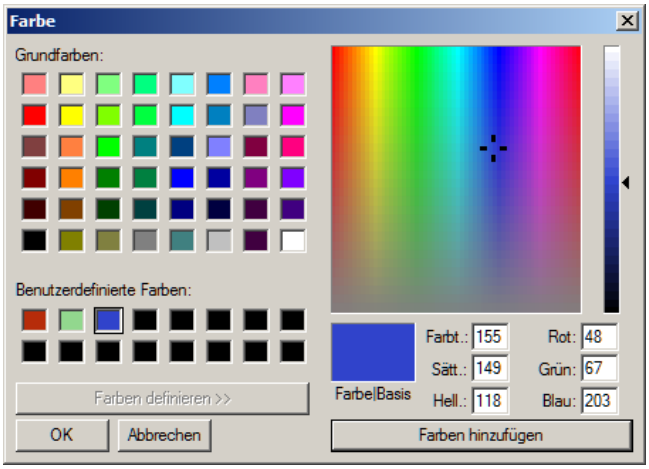


Bild 2: Farbauswahldialog von Windows zur Einstellung der Farben

Combobox **Alpha**. Stellen Sie **Schatten** auf einen Wert größer als **0** ein, so wird die gemalte Linie zusätzlich mit einem **Drop Shadow** hinterlegt, einem Verlaufsschatten, dessen Radius dem gewählten Wert entspricht.

Diese Einstellungen wirken sich jeweils auf den nächsten Zeichenvorgang aus. Malen Sie nun etwa einen Kringel, der nicht unbedingt geschlossen sein muss. Mit Klick auf die Schaltfläche **Figur schließen** geschieht dies jedoch automatisch. Ein weiterer Klick auf den nun aktivierten Button **Figur füllen** versieht die jetzt umrandete Fläche mit der Füllfarbe. Sowohl Linien, wie Schatten und Füllungen berücksichtigen dabei die in **Alpha** eingestellte Transparenz. Sie können somit überlagernd malen!

Ein Klick auf den **Toggle-Button Malen** ändert dessen Beschriftung in **Radieren**. Nun können Sie mit der Maus Teile der Zeichnung ausradieren, was technisch so gelöst ist, dass einfach mit weißer Farbe ohne Transparenz und mit einer Stiftbreite von etwa 10 Pixeln gemalt wird. Ein weiterer Klick auf den Button stellt den vorigen Malmodus wieder her.

Sie können auch ein externes Bild in die Grafik laden, indem Sie den Dateiauswahldialog über den Button **Ext. Bild laden...** aufrufen. Die Position des eingefügten Bilds können Sie damit allerdings eben so wenig beeinflussen, wie dessen Skalierung. Das hätte einigen zusätzlichen Aufwand erfordert, den wir uns erspart haben. Wohl aber beeinflusst wieder der Wert von **Alpha**, wie transparent das Bild eingefügt wird.

Dass die Qualität der Zeichnung etwas zu wünschen übrig lässt, können Sie der Abbildung entnehmen. Mit der Maus ist das Malen gar nicht so einfach. Zudem kommen die Linien etwas krakelig daher, was weniger an **HTML5** liegt, das **Antialiasing** durchaus unterstützt, sondern am eingesetzten Verfahren. Das wollten wir so einfach halten, wie möglich: Bei jeder Mausbewegung wird zwischen den beiden abgetasteten Punkten eine einzelne Linie mit der **lineTo**-Anweisung des **HTML5-Canvas** erzeugt. Deshalb sind Bögen nicht wirklich rund, sondern weisen an den Punkten Kantensprünge auf, die zu einer vermeintlichen Rasterung führen. Sollte das vermieden werden, so müssten mehrere Abtastpunkte in einer **Bezier**-Linie (**bezierCurveTo**) zusammengeführt werden, was den Aufwand deutlich erhöht hätte.

Tabelle als Grafikspeicher

Damit die Zeichnungen nicht nur in Dateien abgespeichert werden können, gibt es die Tabelle **tblBilderDraw**, deren Aufbau Bild 3 verdeutlicht. Neben dem Namen der Zeichnung in **BildName** und dem Autowert **ID** gibt es hier nur ein OLE-Feld **BildOLE**, in das indessen keineswegs **OLE-Objekte** hineinkommen, sondern reine Binärdaten, die exakt der Datei entsprechen, die Sie auch extern abspeichern würden.

Im Formular ist lediglich der **BildName** direkt an die entsprechende Textbox gebunden. Das Webbrowser-Steuerelement ist natürlich ungebunden, denn mit den Binärdaten kann es nichts anfangen. Das Laden und Schreiben der Browser-Daten müssen VBA-Routinen übernehmen.

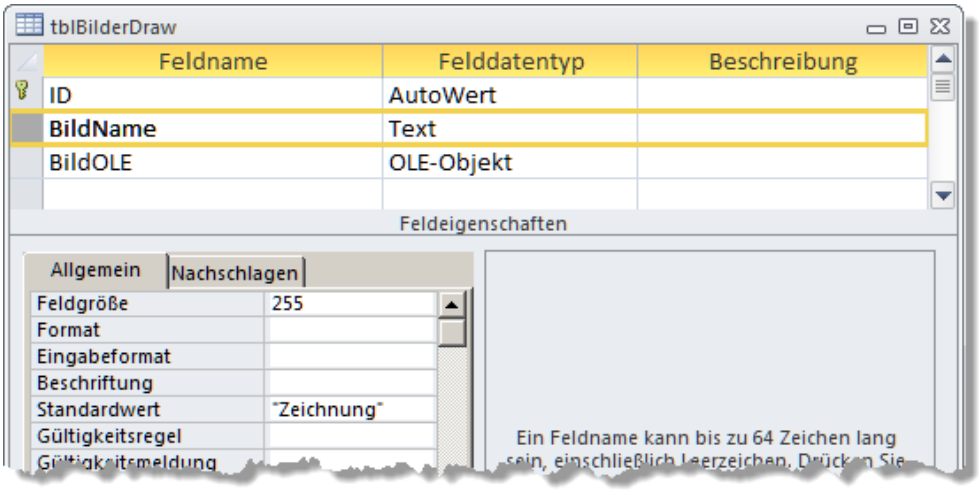


Bild 3: Die Tabelle **tblBilderDraw** speichert nur die gemalten Grafiken und ihren Namen neben der ID

Vorgänge beim Laden und Anzeigen des Formulars
Ein Teil des gerade einmal 250 Zeilen langen Formular-Codes läuft beim Öffnen (**Form_Load**) und im Ereignis **Beim Anzeigen** (**Form_Current**) ab. **Form_Load** stellt lediglich den Formularzeitgeber auf **100 ms** ein:

```
Me.TimerInterval = 100
```

Denn im Zeitgeberereignis steht die eigentliche Prozedur, die die initialen Vorgänge vornimmt. Auch **Form_Current** macht nicht viel mehr:

```
Me!ctlWeb.Object.Navigate2 "about:blank"
...
Form_Timer
```

Sie ruft ebenfalls das **Timer**-Ereignis auf. Grund dafür ist ein **Delay** des Browsers **ctlWeb** beim Navigieren zu einer neuen **URL**, was asynchron abläuft und das Formular zuweilen durcheinanderbringt. Zwar ist das Steuerelement im Entwurf auf **about:blank** eingestellt, doch dies wirkt sich nur beim ersten Laden aus, nicht aber beim Ansteuern eines neuen Datensatzes. Hier muss das leere **HTML**-Dokument ausdrücklich erneut angelegt werden. Interessant ist damit vor allem die Ereignisprozedur **Form_Timer**. Ihre einzelnen Abschnitte folgen nun.

```
Me.TimerInterval = 0
Do While Me!ctlWeb.Object.ReadyState < 3
    DoEvents
Loop
```

Der Zeitgeber wird hier gleich wieder mit Setzen auf **0** deaktiviert, so dass es zu keinen Wiederholungen kommt. Anschließend wartet eine Schleife ab, bis die Eigenschaft **ReadyState** des Browsers mindestens den Wert 3 (**readyStateInteractive**) angenommen hat. Das dann der Fall, wenn der Browser die aufgerufene Seite fertiggestellt hat. Erst dann haben Sie Zugriff auf dessen **HTML-Dokument**. Das wird in der Folge der Formularvariablen **oDoc** zugewiesen:

```
Set oDoc = Me!ctlWeb.Object.Document
oDoc.body.setAttribute "bgColor", "#e0e0e0"
```

Zusätzlich stellt die Routine hier die Hintergrundfarbe des **Dokument-Body** auf hellgrau ein. Im Kopf des Formularmoduls sind noch einige weitere globale Variablen deklariert, auf die in anderen Prozeduren Bezug genommen wird:

```
Private WithEvents oDoc As HTMLDocument
Private WithEvents oDiv As HTMLDivElement
Private oCanvE As IHTMLCanvasElement
Private CTX As ICanvasRenderingContext2D
```

Das HTML-Dokument steht also in **oDoc**. Der **HTML5-Canvas** kommt in die Elementvariablen **oCanvE**. Allerdings bauen wir das **Canvas**-Element nicht direkt in den **Body** des Dokuments ein, sondern in einen zusätzlichen **DIV**-Bereich, der gegebenenfalls verschoben werden kann. Die Variable **oDiv** nimmt einen Verweis auf diesen Bereich entgegen und ist außerdem mit dem Prädikat **WithEvents** versehen, wodurch dieser Ereignisse auslösen kann.

So etwa auch Mausereignisse, die wir später – Sie können es sich schon denken – dann zum Zeichnen auswerten werden. Und schließlich speichert die Variable **CTX** noch den Grafikkontext des **Canvas**, auf den erst ja die Zeichenanweisungen ausgeführt werden. Sie kennen das alles bereits aus dem ersten Beitrag zu **HTML5**.

Nun legt die Prozedur die **HTML-Elemente** im Dokument an und nimmt einige Grundeinstellungen für den **Style** des **Canvas** vor (Listing 1). Erst wird das **DIV**-Element erzeugt und dem **Body** hinzugefügt (**appendChild**). Dann generiert **createElement** einen neuen Canvas und weist ihn der modulglobalen Variablen **oCanvE** zu. Seine **Id** lautet **zeichnung** und seine Breite und Höhe werden fest auf die Abmessungen **600x400** festgelegt. Diese Werte für die Zeichenfläche können Sie natürlich nach Belieben modifizieren.

```
Private Sub Form_Timer()
    Dim oElement As IHTMLElement
...
Set oDiv = oDoc.createElement("div1")
oDiv.Id = "div1"
oDoc.body.appendChild oDiv

Set oCanvE = oDoc.createElement("canvas")
oCanvE.Id = "zeichnung"
oCanvE.Width = 600: oCanvE.Height = 400
oDiv.appendChild oCanvE

Set CTX = oCanvE.getContext("2d")
With CTX
    .globalAlpha = 1
    .strokeStyle = 0
    .lineWidth = 1
    .FillStyle = "#ffffff"
    .fillRect 0, 0, 600, 400
    .strokeRect 0, 0, 600, 400
    .strokeStyle = ColorToStr(rColor.BackColor)
    .FillStyle = ColorToStr(rColorFill.BackColor)
    .shadowBlur = val(cbShadow.Value)
    .shadowColor = ColorToStr(rColorShadow.BackColor)
    .shadowOffsetX = .shadowBlur / 2
    .shadowOffsetY = .shadowBlur / 2
    .lineWidth = CSng(cbLineWidth.Value)
End With
...
End Sub
```

Listing 1: Die **Timer**-Prozedur initialisiert den **HTML5-Canvas**

Weiter geht es mit dem über **getContext(2D")** ermittelten Grafikkontext des **Canvas**, den die Variable **CTX** entgegennimmt. Im **With**-Block auf sie werden nicht nur einige **Style**-Eigenschaften festgelegt, wie Hintergrund (**FillStyle**), Linienbreite (**lineWidth**) oder Schattenradius (**shadowBlur**), sondern auch gleich ein die Grafik umrahmendes Rechteck mit **strokeRect** gezeichnet.

Die Werte der Eigenschaften kommen dabei teilweise aus den Steuerelementen des Formulars. So entspricht der Schattenradius dem Wert der Combobox **cbShadow** oder die Malfarbe (**strokeStyle**) der Hintergrundfarbe (**BackColor**) des Rechtecksteuerelements **rColor**.

Damit sind unsere HTML5-Objekte fertiggestellt und dem Zeichnen steht nichts mehr im Wege. Der Teil der Prozedur, der eine bereits gespeicherte Grafik lädt, ist im Listing übrigens ausgespart. Dazu dann später mehr.

Zeichnen mit der Maus

Ob Sie es glauben oder nicht: Der Code zum Zeichnen umfasst nur 18 Zeilen, aufgeteilt auf drei Ereignisprozeduren! Die werten das Verhalten der linken Maustaste auf den **DIV**-Bereich **oDiv** aus (Listing 2).

Im Unterschied zu den Mausereignissen, die Sie von Access kennen, liefern diese keinerlei Parameter zurück, wie etwa die x- und y-Koordinaten des Ereignisses. Hier muss auf einen Trick zurückgegriffen werden. Alle Ereignisse eines HTML-Elements werden nämlich eigentlich von dessen Fenster verarbeitet, nicht vom Element selbst. An dieses Fenster gelangen Sie, indem Sie die Methode **parentWindow** des Dokuments abfra-

```
Private Sub oDiv_onmousedown()
    Set oEvent = oDoc.parentWindow.event
    If oEvent.Button = 1 Then
        CTX.beginPath
        CTX.moveTo oEvent.offsetX, oEvent.OffsetY
    End If
End Sub

Private Sub oDiv_onmousemove()
    Set oEvent = oDoc.parentWindow.event
    If oEvent.Button = 1 Then
        CTX.lineTo oEvent.offsetX, oEvent.OffsetY
        CTX.stroke
    End If
End Sub

Private Sub oDiv_onmouseup()
    Set oEvent = oDoc.parentWindow.event
    If oEvent.Button = 1 Then
        cmdFill.Enabled = False
    End If
End Sub
```

Listing 2: Auswertung der Ereignisse des HTML-Bereichs **oDiv**

Datenkompression leicht gemacht

Wenn Sie längere Texte in Tabellen abspeichern oder umfangreiche Binärdaten in OLE-Feldern ablegen, so bläht das die Datenbank auf. Während dieser Umstand bei lokalen Datenbanken heutzutage kein Problem mehr darstellt, sieht die Sache in einer Mehrbenutzerumgebung und Netzwerkzugriffen auf Backends schon anders aus. Hier sollte der Traffic minimiert werden. Da macht sich dann die Komprimierung solcher Datenfelder gut, was zu verbesserter Performance führen kann.

Komprimieren von Memo- und Long-Binary-Feldern
Über das Für und Wider von in Tabellen abgelegten Binärdaten oder Textdokumente möchten wir hier nicht diskutieren. Das kann ganz einfach immer wieder vorkommen, und es gibt ganze Serverumgebungen, die davon weidlich Gebrauch machen, wie etwa der **Microsoft Sharepoint-Server**. Die Kompression dieser Daten verkleinert die Datenbankdateien und hilft dabei den benötigten Traffic zu den Clients zu verringern.

An sich ist derlei bereits in Access integriert. Der Datentyp **Anlage** (**Attachment**) speichert binäre Daten und komprimiert diese nach Bedarf, wobei Microsoft sich weitgehend darüber ausschweigt, bei welchen Dateitypen dies zutrifft und welcher Kompressionsalgorithmus zum Einsatz kommt. Nicht jeder ist jedoch ein Freund von **Anlage-Feldern**, denn aufgrund des dahinter liegenden versteckten Datenmodells, welches sich tatsächlich von **Sharepoint**-Technik ableitet, ist der programmtechnische Umgang mit ihnen recht komplex. Zudem können Anlagefelder nicht in **Access 2003** oder früher angezeigt oder ausgelesen werden. Deshalb macht sich die Binärspeicherung in **OLE-Feldern**, bei denen es sich ja schlicht um **Long-Binary**-Felder handelt, besser.

Das Thema ist nicht neu. Bereits in der Ausgabe **3/2007** (**Shortlink 466**) von **Access Im Unternehmen** wurde es behandelt, weshalb wir an dieser Stelle zusätzlich auf jenen Beitrag verweisen. Dort wurde die Komprimierung über eine externe Komponente erreicht, eine **zlib-DLL**, auf die im VBA-Projekt verwiesen wird. Darauf kann verzichtet werden, wenn man sich der wenig beachteten Windows-API-Funktion **RTLCompressBuffer** bedient, die eigentlich zur Abteilung der Treiber-APIs gehört. Sie erreicht, vor allem bei Texten, durchaus vergleichbare Kompressionsraten. Der eingesetzte Algorithmus ist eine **LZ**-Variante (**Lempel-Ziv**).

Einschub: Kompression in Anlagefeldern
Da Microsoft keine komplette Liste veröffentlicht, welche Dateitypen in **Anlage-Feldern** denn nun komprimiert werden und welche nicht, sondern pauschal behauptet, dass jene unbearbeitet blieben, die bereits hinlänglich komprimiert sind, wie etwa **JPG**, haben wir einen einfachen Test unternommen. Eine Datenbank enthält eine Tabelle, die wiederum nur ein **Anlage-Feld** enthält. Wir messen die Dateigröße dieser **ACCDB** nach dem Hinzufügen jeweils **einer** größeren Datei eines bestimmten Typs und anschließendem **Komprimieren und Reparieren** der Datenbank. Verwendet werden gängige Dateitypen. Steigt die Größe der **ACCDB** um die Größe der hinzugefügten Datei, so dürfte die interne Kompression der **Anlage** nicht angesprochen haben. Die folgende Tabelle gibt Auskunft über das Ergebnis:

Dateityp	Größe Datei	Größe DB	Kompression, ca.
(leer)	-	404 kB	-
txt	3770 kB	1020 kB	6-fach
doc	3430 kB	1778 kB	2,5-fach
rtf	4400 kB	524 kB	37-fach
xls	3640 kB	1020 kB	6-fach
ppt	3800 kB	676 kB	14-fach
pdf	3280 kB	2208 kB	1,8-fach
bmp	4130 kB	516 kB	34-fach

jpg	4300 kB	4752 kB	keine
tiff	3050 kB	3305 kB	keine
xml	3830 kB	992 kB	6,5-fach
kml	6650 kB	540 kB	49-fach
csv	3380 kB	740 kB	10-fach

Zu erwähnen wäre noch, dass dezidiert Dateien verwendet wurden, die entweder viele Wiederholungen enthalten oder viele **Null-Bytes**, sich mithin also gut komprimieren lassen. Das Ergebnis ist eindeutig: Die Kompression scheint zum Glück der Default zu sein. Nur ausgewählte Dateiformate, wie J**PG, ZIP, DOCX, XLSX**, die bekannterweise schon eine Kompression aufweisen, werden ausgenommen. Erstaunlich hoch sind die Kompressionsraten. Dass etwa eine **RTF-Datei** von über **4 MB** Größe auf intern **120 kB** schrumpft, war nicht zu erwarten. Als Datei-Container eignen sich **Anlage-Felder** demnach hervorragend.

Leider kann der in diesem Beitrag vorgestellte Algorithmus da nicht mithalten. Dennoch hat er seine Existenzberechtigung, und wir führen kurz die Vorteile gegenüber **Attachments** auf:

- Die komprimierte Speicherung ist nicht nur auf Dateien beschränkt, sondern kann etwa auch auf beliebige per VBA generierte Strings angewandt werden.
- Die Speicherung in OLE-Feldern kann auch in Access 2003 und früher erfolgen.
- Auch EXE-Dateien können integriert werden, was Access bei Anlage-Feldern verbietet.
- OLE-Felder lassen sich über ODBC ansprechen und deren Inhalte sind damit auch für SQL-Server oder andere DBMS nicht tabu.
- Der Umgang mit Long-Binary-Feldern ist unter VBA und DAO, sowie ADODB, erheblich einfacher, als mit **Attachments**.

Kompressionsroutine

Das Modul **mdlCompression** der Beispieldatenbank enthält eine Funktion **CompressRTL**, die **Byte-Arrays** komprimieren kann. Ihr Code ist in Listing 1 abgebildet. Er nutzt einige **Windows-API-Funktionen**, die im Modulkopf deklariert sind (Listing 2). Der Rückgabewert ist wiederum ein **Byte-Array** mit den komprimierten Daten. Das Ganze kommt ausgesprochen übersichtlich daher. Auf die genaue Funktion des Codes und der **API-Aufrufe** möchten wir hier gar nicht eingehen. Einige Hinweise dürfen jedoch nicht fehlen.

So enthält die Prozedur keine Fehlerbehandlung. Übergeben Sie etwa ein undimensioniertes Array, so kracht es gleich in der ersten Zeile beim Ausdruck **UBound**. Auch sonst werden die Rückgabewerte der API-Funktionen nicht ausgewertet, so dass Fehlfunktionen nicht offensichtlich sind. Allerdings dürfte es auch kaum einen Fall geben, der hier bei einem ordnungsgemäßen Byte-Array zu Fehlern führt. Kritisch ist bestenfalls der Wert der von **RTLCompressBuffer** zurückgelieferten Variablen **retSize**, die angibt, wie viele Bytes im Ergebnis enthalten sind. Er wird im weiteren Verlauf dazu verwendet, um das Ergebnis-Array **binOut** neu zu dimensionieren. Hier könnten rein theoretisch Werte anfallen, die über den erlaubten Wertebereich hinausgehen. Uns ist bei allen Tests derlei aber nicht untergekommen.

```
Public Function CompressRTL(Buffer() As Byte) As Byte()
    Dim LSize As Long, memSize As Long, retSize As Long
    Dim lMem As Long
    Dim binOut() As Byte
    LSize = UBound(Buffer) * 1.13 + 4
    ReDim binOut(LSize)
    RtlGetCompressionWorkSpaceSize 2, memSize, 0
    NtAllocateVirtualMemory -1, lMem, 0, memSize, 4096, 64
    RtlCompressBuffer 2, VarPtr(Buffer(0)), UBound(Buffer) + 1, _
        VarPtr(binOut(0)), LSize, 0, retSize, lMem
    NtFreeVirtualMemory -1, lMem, 0, 16384
    ReDim Preserve binOut(retSize)
    CompressRTL = binOut
End Function
```

Listing 1: Routine **CompressRTL** zum Komprimieren von Byte-Arrays

```
Private Declare Function RtlGetCompressionWorkSpaceSize _
    Lib "NTDLL" (ByVal Flags As Integer, WorkSpaceSize As _
    Long, UNKNOWN_PARAMETER As Long) As Long

Private Declare Function NtAllocateVirtualMemory Lib _
    "ntdll.dll" (ByVal ProcHandle As Long, BaseAddress As _
    Long, ByVal NumBits As Long, regionsize As Long, ByVal _
    Flags As Long, ByVal ProtectMode As Long) As Long

Private Declare Function RtlCompressBuffer Lib "NTDLL" _
    (ByVal Flags As Integer, ByVal BuffUnCompressed As Long, _
    ByVal UnCompSize As Long, ByVal BuffCompressed As Long, _
    ByVal CompBuffSize As Long, ByVal UNKNOWN_PARAMETER As _
    Long, OutputSize As Long, ByVal WorkSpace As Long) As _
    Long

Private Declare Function RtlDecompressBuffer Lib _
    "NTDLL" (ByVal Flags As Integer, ByVal BuffUnCompressed _
    As Long, ByVal UnCompSize As Long, ByVal BuffCompressed _
    As Long, ByVal CompBuffSize As Long, OutputSize As _
    Long) As Long

Private Declare Function NtFreeVirtualMemory Lib _
    "ntdll.dll" (ByVal ProcHandle As Long, BaseAddress As _
    Long, regionsize As Long, ByVal Flags As Long) As Long
```

Listing 2: Die fünf im Modul verwendeten API-Funktionen

Nun möchten Sie möglicherweise nicht Byte-Arrays komprimieren, sondern Strings. Das macht eine Konvertierung dieser erforderlich. Gegeben sei etwa ein Text in der Variablen **sText**, der zu komprimieren wäre. Dann verwenden Sie die VBA-Funktion **StrConv**:

```
Dim bin() As Byte
Dim binRet() As Byte
bin = StrConv(sText, vbFromUnicode)
binRet = CompressRTL(bin)
```

StrConv kann Strings in Byte-Arrays überführen und umgekehrt. Da es sich bei VBA-Strings immer um **Unicode** handelt, ist der Parameter **vbFromUnicode** der richtige. Er macht aus dem String quasi ein **ANSI-Byte-Array**. Dieses übergeben Sie dann der Funktion **CompressRTL** des Moduls.

Dekompressionsroutine

Mit dem komprimierten **Byte-Array** selbst können Sie wenig anfangen. Entweder speichern Sie es in einer Datei ab, oder im **OLE-Feld** einer Tabelle. Es reicht dazu schlicht die Zuweisung an den Feldinhalt. Beispiel:

```
Dim rs As DAO.Recordset
Set rs = CurrentDb.
OpenRecordset("Binaertabelle")
rs.Edit 'oder rs.AddNew
rs!OleFeld1.Value = binRet
rs.Update
```

Einstmals mussten Sie sich mit der Methode **AppendChunk** eines **DAO**-Felds herumschlagen und in einer Schleife Teile des Byte-Arrays (**Chunks**) hinzufügen. Das ist veraltet. Die unmittelbare Zuweisung funktioniert bei heutigen Rechnerausstattungen auch bei großen Byte-Arrays. Das Limit liegt lediglich beim dem **DAO-Thread** zugewiesenen RAM-Bereich (**Heap Working Set**) der Datenbank-Engine.

```
Public Function DeCompressRTL(Buffer() As Byte) As Byte()
    Dim LSize As Long, memSize As Long
    Dim binOut() As Byte

    LSize = UBound(Buffer) * 12.5
    ReDim binOut(LSize)
    RtlDecompressBuffer 2& Or &H100, VarPtr(binOut(0)), LSize, _
        VarPtr(Buffer(0)), 1& + UBound(Buffer), memSize
    If (memSize > 0) And (memSize <= UBound(binOut)) And _
       (memSize < 2000000) Then
        ReDim Preserve binOut(memSize - 1)
        DeCompressRTL = binOut
    End If
End Function
```

Listing 3: Routine **DecompressRTL** zum Entpacken der Byte-Arrays

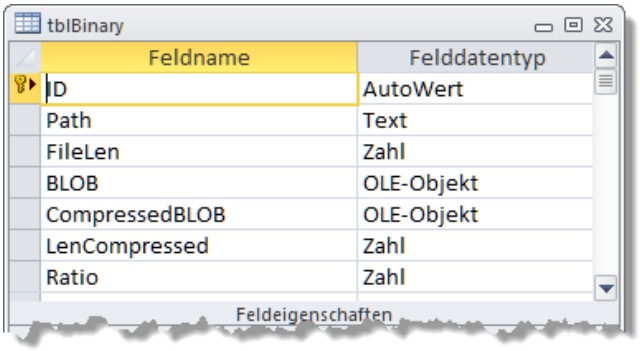


Bild 1: Aufbau der Tabelle **tblBinary** zum Abspeichern binärer Daten

```
Sub ImportFiles(Byval sDir As String)
    Dim sFile As String
    Dim bin() As Byte
    Dim rs As DAO.Recordset
    Dim F As Integer

    CurrentDb.Execute "DELETE FROM tblBinary"
    Set rs = CurrentDb.OpenRecordset( _
        "SELECT * FROM tblBinary", dbOpenDynaset)
    sFile = Dir(sDir & "\*.*")
    Do While Len(sFile) > 0
        F = FreeFile
        Open sDir & "\" & sFile For Binary As F
        ReDim bin(LOF(F) - 1)
        Get F, , bin
        Close F

        With rs
            .AddNew
            !Path = sDir & "\" & sFile
            !FileLen = FileLen(sPath)
            !BLOB.Value = bin
            !CompressedBLOB.Value = CompressRTL(bin)
            !LenCompressed = !CompressedBLOB.FieldSize
            !Ratio = Round(!FileLen.Value / _
                !LenCompressed.Value, 3)
            .Update
        End With
        sFile = Dir
    Loop
    rs.Close
End Sub
```

Listing 4: Einlesen der Dateien eines Verzeichnisses in die Tabelle

Nun möchten Sie die komprimierten Daten wieder in ihrer ursprünglichen Form zurückgewinnen. Das übernimmt die Dekompressionsroutine **DecompressRTL** des Moduls, welche in Listing 3 abgebildet ist. Auch ihr Umfang ist erfreulich gering. Ihr Aufruf entspricht genau dem für die Komprimierprozedur **CompressRTL**. Sie holen sich etwa die komprimierten Daten aus der Tabelle und erhalten die entpackten im Rückgabe-Array, welches Sie wieder per **StrConv** in einen Text verwandeln:

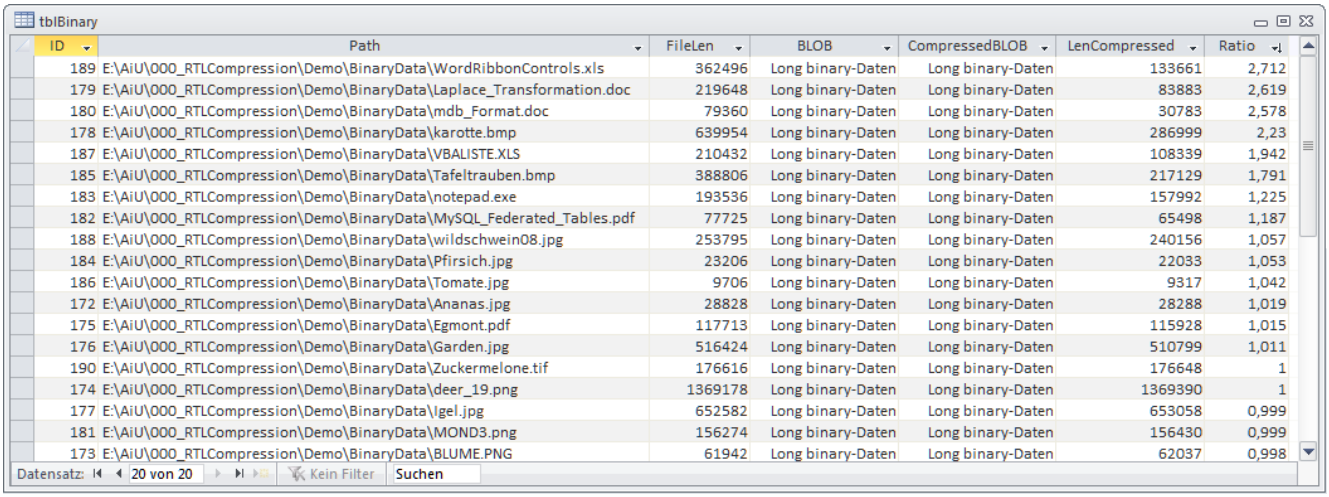
```
Dim rs As DAO.Recordset
Dim bin() As Byte
Dim binRet() As Byte
Dim sText As String

Set rs = CurrentDb.OpenRecordset("Binaertabelle")
bin = rs!OleFeld1.Value
binRet = DecompressRTL(bin)
sText = StrConv(binRet, vbUnicode)
```

Die String-Umwandlung passiert nun natürlich umgekehrt mittels des Parameters **vbUnicode**. Haben Sie nicht Texte, sondern Dateien komprimiert, so entfällt die Umwandlung per **StrConv** selbstverständlich jeweils.

Dateien komprimiert im OLE-Feld

Das bisher Gesagte zusammengefasst mündet in ein praktisches Beispiel, das so auch in der Demo-Datenbank vorhanden ist. Alle Dateien eines bestimmten Verzeichnisses sollen komprimiert als Datensätze in einer Tabelle abgespeichert werden. Die dafür vorgesehene Routine **ImportFiles** finden Sie in Listing 4, die Entwurfsansicht der Zieltabelle in Bild 1.

Das Feld **Path** nimmt den vollständigen Pfad zur Datei auf, **FileLen** deren Größe in Bytes. **BLOB** ist der Container für die unkomprimierten Daten. In der Praxis würden Sie dieses Feld nicht verwenden, da im OLE-Feld **CompressedBLOB** die komprimierten Daten stehen, aus denen die Datei ja wiederhergestellt werden kann. **LenCompressed** speichert die Größe der komprimierten



tblBinary

ID	Path	FileLen	BLOB	CompressedBLOB	LenCompressed	Ratio
189	E:\AiU\000_RTLCompression\Demo\BinaryData\WordRibbonControls.xls	362496	Long binary-Daten	Long binary-Daten	133661	2,712
179	E:\AiU\000_RTLCompression\Demo\BinaryData\Laplace_Transformation.doc	219648	Long binary-Daten	Long binary-Daten	83883	2,619
180	E:\AiU\000_RTLCompression\Demo\BinaryData\mdb_Format.doc	79360	Long binary-Daten	Long binary-Daten	30783	2,578
178	E:\AiU\000_RTLCompression\Demo\BinaryData\karotte.bmp	639954	Long binary-Daten	Long binary-Daten	286999	2,23
187	E:\AiU\000_RTLCompression\Demo\BinaryData\VBALISTE.XLS	210432	Long binary-Daten	Long binary-Daten	108339	1,942
185	E:\AiU\000_RTLCompression\Demo\BinaryData\Tafeltrauben.bmp	388806	Long binary-Daten	Long binary-Daten	217129	1,791
183	E:\AiU\000_RTLCompression\Demo\BinaryData\notepad.exe	193536	Long binary-Daten	Long binary-Daten	157992	1,225
182	E:\AiU\000_RTLCompression\Demo\BinaryData\MySQL_Federated_Tables.pdf	77725	Long binary-Daten	Long binary-Daten	65498	1,187
188	E:\AiU\000_RTLCompression\Demo\BinaryData\wildschwein08.jpg	253795	Long binary-Daten	Long binary-Daten	240156	1,057
184	E:\AiU\000_RTLCompression\Demo\BinaryData\Pfirsich.jpg	23206	Long binary-Daten	Long binary-Daten	22033	1,053
186	E:\AiU\000_RTLCompression\Demo\BinaryData\Tomate.jpg	9706	Long binary-Daten	Long binary-Daten	9317	1,042
172	E:\AiU\000_RTLCompression\Demo\BinaryData\Ananas.jpg	28828	Long binary-Daten	Long binary-Daten	28288	1,019
175	E:\AiU\000_RTLCompression\Demo\BinaryData\Egmont.pdf	117713	Long binary-Daten	Long binary-Daten	115928	1,015
176	E:\AiU\000_RTLCompression\Demo\BinaryData\Garden.jpg	516424	Long binary-Daten	Long binary-Daten	510799	1,011
190	E:\AiU\000_RTLCompression\Demo\BinaryData\Zuckermelone.tif	176616	Long binary-Daten	Long binary-Daten	176648	1
174	E:\AiU\000_RTLCompression\Demo\BinaryData\deer_19.png	1369178	Long binary-Daten	Long binary-Daten	1369390	1
177	E:\AiU\000_RTLCompression\Demo\BinaryData\Igel.jpg	652582	Long binary-Daten	Long binary-Daten	653058	0,999
181	E:\AiU\000_RTLCompression\Demo\BinaryData\MOND3.png	156274	Long binary-Daten	Long binary-Daten	156430	0,999
173	E:\AiU\000_RTLCompression\Demo\BinaryData\BLUME.PNG	61942	Long binary-Daten	Long binary-Daten	62037	0,998

Datensatz: 20 von 20 | Kein Filter | Suchen

Bild 2: Die Datenblattansicht der Tabelle tblBinary zeigt den Ursprung der Dateien neben zusätzlichen Angaben zur Kompression

Daten und **Ratio** als Gleitkommazahl das Kompressionsverhältnis. Diese beiden letzten Felder könnte man ebenfalls weglassen, da sich **LenCompressed** aus **CompressedBLOB.FieldSize** ergäbe, und **Ratio** aus Selbigem geteilt durch **FileLen**. Die beiden Felder sind hier integriert, damit man in der Datenblattansicht der Tabelle gleich eine Übersicht über den Erfolg der Kompression hat.

Die mit einigen Beispieldateien gefüllte Tabelle zeigt Bild 2. Sie ist absteigend nach dem Kompressionsfaktor (**Ratio**) sortiert. Es wird deutlich, dass sich bei den Bilddateien und PDFs keine nennenswerte Kompression einstellt. **Notepad.exe** verringert sich immerhin um ein Viertel. Bei den Bitmaps hängt der Faktor vom Bildinhalt ab. Je mehr gleichfarbige Flächen enthalten sind, desto besser die Kompression. Dasselbe gilt für Excel- und Word-Dokumente. Die Kompression ist umso erfolgreicher, je mehr Wortdoppelungen in diesen Dokumenten enthalten sind.

Wir haben alle Dateien zum Vergleich einmal in **ZIP-Archive** gepackt. Die resultierenden Dateigrößen sind allgemein deutlich geringer, als die mit der API-Funktion erreichten. Das Nonplusultra stellt der Algorithmus von **RTLCompressBuffer** also nicht dar. Dafür arbeitet die Funktion allerdings außerordentlich schnell. Möchten Sie etwa eine größere Zahl von Office-Dokumenten in Ihrer Datenbank ablegen, so verkleinern Sie deren Dateigröße mit der Kompression etwa um den Faktor zwei bis drei.

Doch zurück zur Prozedur in **Listing 4**. Hier werden in einer **Do While**-Schleife alle Dateien des als Parameter übergebenen Verzeichnisses **sDir** abgearbeitet und in die Tabelle **tblBinary** übertragen. Auf diese ist, nachdem sie in der ersten Zeile geleert wird, ein Recordset **rs** geöffnet. Eine Datei liest die **Open-Anweisung** im Binärmodus in das Byte-Array **bin** ein. Dessen Inhalt wird direkt dem Feld **BLOB** als Wert übergeben, während das Feld **CompressedBLOB** das Ergebnis des Funktionsaufrufs der weiter oben behandelten Prozedur **CompressRTL** speichert. Das Einlesen der Dateien von Bild 2 dauerte bei uns mit dieser Routine gerade einmal eine Sekunde! Die Kompression selbst geschah jeweils in wenigen Millisekunden.

Texte komprimiert im OLE-Feld

Für das Abspeichern komprimierter Textdateien gibt es in der Beispieldatenbank eine gesonderte Tabelle **tblTexte** (siehe Bild 3) und eine eigene Prozedur **ImportTexts**. Diese Routine entspricht ziemlich genau der vorigen Prozedur **ImportFiles**, weshalb ein Listing entfallen kann. Nur benutzt sie zusätzlich die Konvertierung des eingelesenen Byte-Arrays per **StrConv** in Strings. Die

Volltextsuche mit Fundstellen

Neulich wollte ich mal meine Artikeldatenbank nach bestimmten Suchbegriffen durchforsten. Also habe ich mir eine Abfrage gebaut, die nur die relevanten Felder der zu durchsuchenden Tabelle enthält. Dort dachte ich dann, ich könnte mit der eingebauten Filter-Funktion des Datenblatts schnell zu den gesuchten Fundstellen gelangen. Das Ergebnis war ernüchternd, denn Access lieferte zwar alle passenden Datensätze, aber in den recht langen Texten innerhalb der Datensätze musste ich dann nochmals nach dem gesuchten Text suchen. Das konnte ich nicht auf mir sitzen lassen und habe eine Suchfunktion mit Fundstellen programmiert, die dieser Beitrag vorstellt.

Normalerweise würde man im Datenblatt mit der Maus auf Pfeil nach unten rechts im Spaltenkopf klicken, dann den Menüeintrag **Textfilter|Enthält...** auswählen, den Suchbegriff eingeben und dann die Suchergebnisse betrachten. Das gelingt auch grundsätzlich, wie Bild 1 zeigt. Aber wie eingangs erwähnt, wird auf diese Weise nicht die Fundstelle markiert.

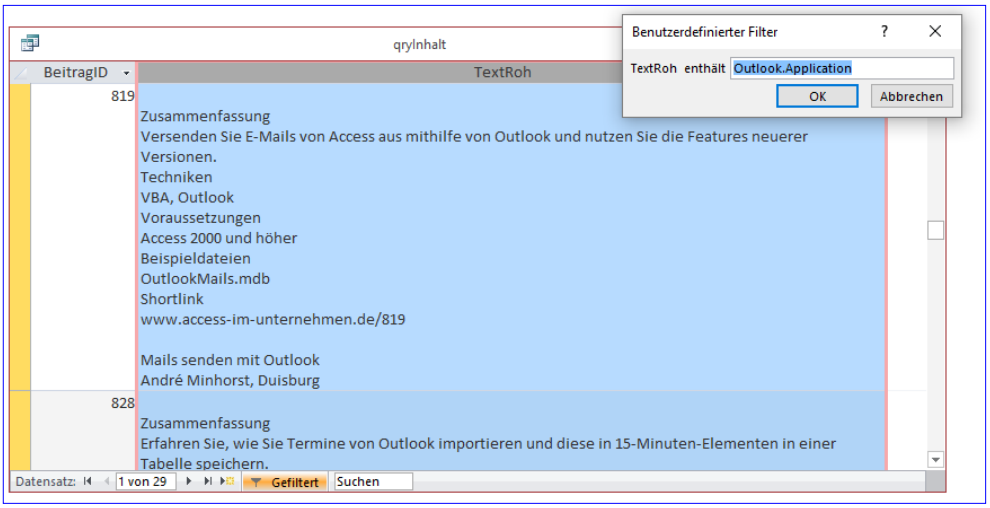


Bild 1: Filtern nach Suchausdruck

Dies erreichen wir etwas besser, wenn wir die **Suchen und Ersetzen**-Funktion nutzen, die wir über den Ribbon-Eintrag **Start|Suchen|Suchen** öffnen. Diese springt so zur Fundstelle, dass diese meist ganz unten im sichtbaren Teil des Textes angezeigt wird (s. Bild 2). Auch das ist nur bedingt praxistauglich, da so nur

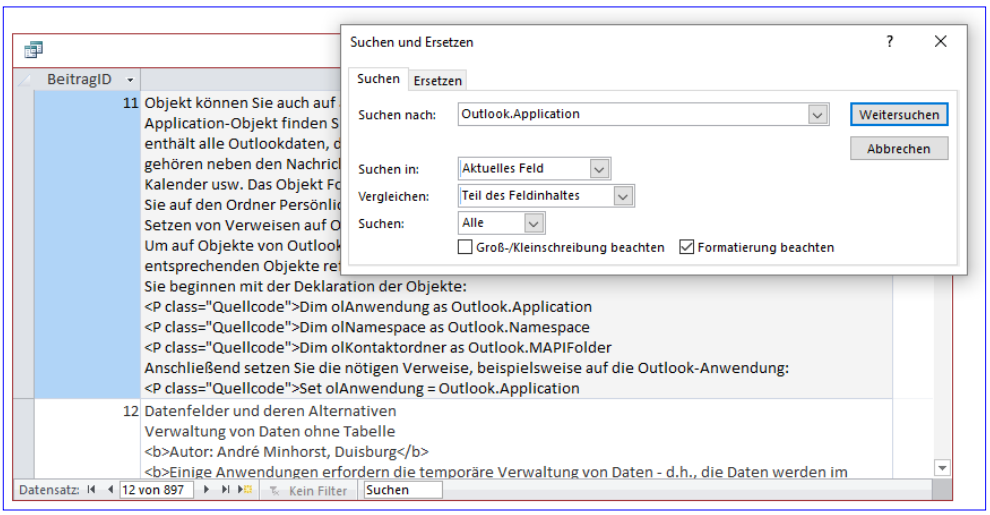


Bild 2: Einsatz der **Suchen und Ersetzen**-Funktion

jeweils ein Eintrag gleichzeitig angezeigt wird. Also benötigen wir eine alternative Lösung.

Ziel

Statt der nicht perfekten Lösung durch die eingebauten Elemente der Access-Datenblattansicht wollen wir eine benutzerdefinierte Lösung entwickeln. Diese soll die Fundstellen ebenfalls in einer Datenblattansicht anzeigen. Allerdings wollen wir auch nur die Fundstelle erhalten und nicht den kompletten Text des durchsuchten Inhalts.

Und der gesuchte Begriff soll farbig hervorgehoben werden, damit wir ihn direkt erkennen! Da wir meist Texte mit mehreren Absätzen haben, wollen wir außerdem einstellen können, wie viele Absätze vor und nach dem Absatz mit dem Suchbegriff ausgegeben werden sollen.

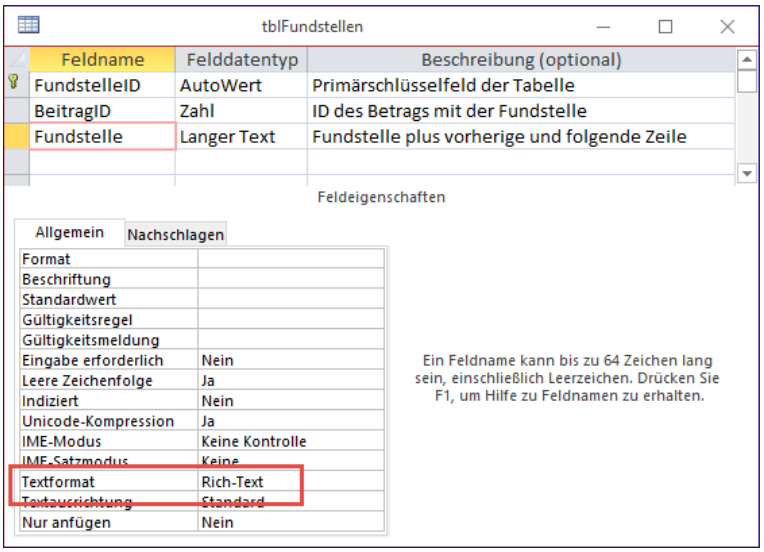


Bild 3: Tabelle zum Speichern der Fundstellen

Wir kümmern uns zunächst um die einfache Variante der Prozedur, die nach dem Eingeben eines Suchbegriffs und dem Betätigen der Taste **cmdSuchen** ausgelöst wird.

Tabelle zum Speichern der Fundstellen

Wenn wir einen oder mehrere längere Texte nach einem Suchbegriff durchsuchen, erhalten wir gegebenenfalls mehrere Suchergebnisse beziehungsweise Fundstellen. Um dem Benutzer diese anzuzeigen, speichern wir sie in einer Tabelle und geben den Inhalt der Tabelle dann in einem Formular aus. Die Tabelle zum Speichern der Fundstellen heißt **tblFundstellen** und sieht im Entwurf wie in Bild 3 aus. Neben dem Primärschlüsselfeld enthält die Tabelle ein Feld zum Speichern der ID des Beitrags, aus dem die Fundstelle stammt, sowie die Fundstelle selbst.

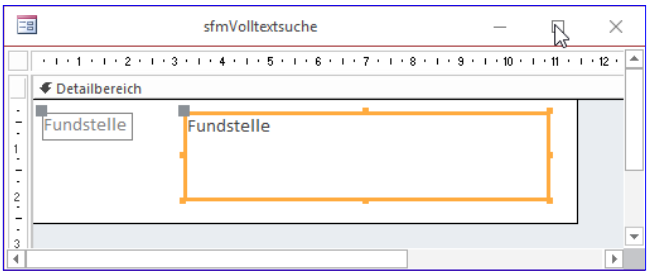


Bild 4: Entwurf des Unterformulars zur Anzeige der Fundstellen

Dabei soll es sich zunächst um die Zeile mit dem gefundenen Begriff und zur zusätzlichen Orientierung noch die vorherige und die folgende Zeile handeln.

Da wir den Suchbegriff selbst in der Fundstelle als Richtext farbig markieren wollen, stellen wir für die Eigenschaft **Textformat** des Feldes **Fundstelle** den Wert **Rich-Text** ein.

Unterformular zur Anzeige der Fundstellen

Die Daten der Tabelle sollen in einem Unterformular namens **sfmVolltextsuche** ausgegeben werden. Dazu binden wir dieses Unterformular an die Tabelle **tblFundstellen** und ziehen das Feld **Fundstelle** aus der Feldliste in den Entwurf dieses Formulars (s. Bild 4).

Da wir in dieses Unterformular weder Daten eingeben noch vorhandene Daten bearbeiten oder löschen wollen, stellen wir die Eigenschaften **Anfügen zulassen**, **Löschen zulassen** und **Bearbeiten zulassen** jeweils auf den Wert **Nein** ein.

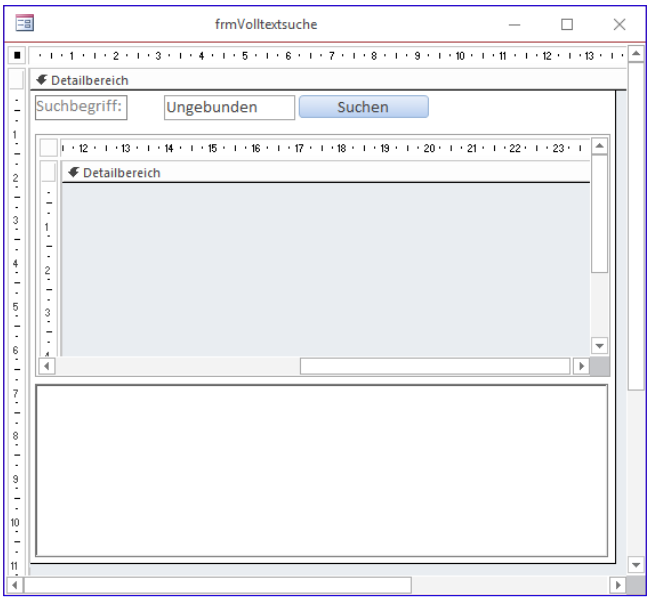


Bild 5: Entwurf des Hauptformulars

Hauptformular mit der Suchfunktion

Das Hauptformular der Lösung enthält die folgenden Steuerelemente:

- Ein Textfeld namens **txtSuchbegriff** zur Eingabe des Suchbegriffs,
- eine Schaltfläche namens **cmdSuchen**, welche die Suche startet,
- das Unterformular **sfmVolltextsuche** und
- das MSForms-Textfeld **txtFundstelle**.

Die Steuerelemente werden wie in Bild 5 angeordnet. Damit die Suche nach der Eingabe des Suchbegriffs in das Textfeld **txtSuchbegriff** sowohl durch das Betätigen der Eingabetaste als auch durch einen Mausklick auf die Schaltfläche **cmdSuchen** ausgelöst werden kann, stellen wir die Eigenschaft **Standard** der Schaltfläche **cmdSuchen** auf **Ja** ein.

Damit das Unterformular mit den Suchstellen vergrößert wird, wenn der Benutzer das Hauptformular vergrößert, stellen wir die beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** jeweils auf den Wert **Beide** ein. Damit das Textfeld **txtFundstelle** nach unten verschoben wird, wenn der Benutzer die Höhe des Formulars vergrößert, stellen wir die Eigenschaft **Vertikaler Anker** dieses Steuerelements auf den Wert **Unten** ein.

Zieht der Benutzer das Formular in die Breite, soll auch das Textfeld verbreitert werden, also erhält die Eigenschaft **Horizontaler Anker** den Wert **Beide**.

Das Formular soll an die Tabelle mit den zu durchsuchenden Texten gebunden sein, damit sie den kompletten Text zur aktuell im Unterformular markierten Fundstelle schnell anzeigen kann.

Deshalb stellen wir die Eigenschaft **Datenherkunft** des Formulars auf eine Abfrage namens **qryInhalt** ein, die wiederum folgenden SQL-Ausdruck enthält und somit nur den Primärschlüsselwert des betroffenen Beitrags sowie seinen vollständigen Text enthält:

```
SELECT BeitragID, Inhalt, TextRoh
FROM tblBeitraege;
```

Die MSForms-TextBox binden wir über die Eigenschaft **Steuerelementinhalt** an das Feld **Inhalt** dieser Abfrage.

Initialisieren des Formulars

Um das Formular und die Steuerelemente zu initialisieren, legen wir zwei Ereignisprozeduren an. Die erste soll durch das Ereignis **Beim Öffnen** des Formulars ausgelöst werden und die gegebenenfalls noch vorhandenen Einträge in der Tabelle **tblFundstellen** löschen.

Diese Prozedur sieht wie folgt aus:

```
Private Sub Form_Open(Cancel As Integer)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "DELETE FROM tblFundstellen", dbFailOnError
End Sub
```

Die zweite Prozedur löst das Ereignis **Beim Laden** aus. Dort initialisieren wir eine Variable zum Referenzieren der MSForms-TextBox, die wie folgt deklariert wird:

```
Dim WithEvents objFundstelle As MSForms.TextBox
```

Die durch das Ereignis **Beim Laden** ausgelöste Prozedur hat folgenden Code:

```
Private Sub Form_Load()
    Set objFundstelle = Me!txtFundstelle.Object
    With objFundstelle
        .SelectionMargin = False
        .Font = "Calibri"
        .Font.Size = "9"
        .MultiLine = True
        .BorderStyle = fmBorderStyleSingle
        .SpecialEffect = fmSpecialEffectFlat
        .BorderColor = Me!txtSuchbegriff.BorderColor
        .ScrollBars = fmScrollBarsVertical
    End With
    Set sfm = Me!sfmVolltextsuche.Form
    With sfm
        .OnCurrent = "[Event Procedure]"
    End With
End Sub
```

Hier stellen wir vor allem die Eigenschaften der MSForms-TextBox **txtFundstelle** ein (Details siehe Beitrag **Die MSForms-TextBox**, **www.access-im-unternehmen.de/1114**). Außerdem referenzieren wir auch noch das Unterformular mit einer Variablen namens **sfm**. Auch diese soll im Kopf des Klassenmoduls deklariert werden, und zwar wie folgt:

```
Private WithEvents sfm As Form
```

Das Schlüsselwort **WithEvents** verwenden wir, weil wir im Hauptformular auf die Ereignisse des Unterformulars reagieren wollen, in diesem Fall auf das Ereignis **Beim Anzeigen**. Deshalb stellen wir die Eigenschaft **OnCurrent** von **sfm** auf den Wert **[Event Procedure]** ein. Die passende Ereignisprozedur können wir auch gleich vorstellen:

```
Private Sub sfm_Current()
    EintragAnzeigen
End Sub
```

Die hier aufgerufene Prozedur **EintragAnzeigen** besprechen wir jedoch weiter unten.

Fundstellen ermitteln

Wenn der Benutzer einen Suchbegriff in das Textfeld **txtSuchbegriff** eingegeben hat und entweder die Eingabetaste betätigt oder auf die Schaltfläche **cmdSuchen** klickt, löst er damit die Ereignisprozedur **cmdSuchen_Click** aus, die Sie in Listing 1 finden.

Diese Prozedur prüft zunächst, ob das Textfeld **txtSuchbegriff** überhaupt einen Suchbegriff enthält. Ist das nicht der Fall, erscheint eine entsprechende Meldung und die Prozedur wird beendet.

Anderenfalls speichert die Prozedur den Suchbegriff aus dem Textfeld in der Variablen **strSuche** und aktiviert die Anzeige der Sanduhr. Dann füllt sie die Variable **db** mit einem Verweis auf das aktuelle **Database**-Objekt. Die Variable **rst** erhält ein Recordset, das alle Datensätze der Tabelle **tblBeitraege** enthält, deren Feld **TextRoh** das Suchergebnis enthält.

Damit filtern wir alle Einträge der Tabelle heraus, die als Quelle für Fundstellen infrage kommen. Die folgende Anweisung löscht dann alle gegebenenfalls noch vorhandenen Einträge der Tabelle **tblFundstellen**.

Anschließend durchläuft die Prozedur alle Datensätze des Recordsets aus **rst**, also die Datensätze der Tabelle **tblBeitraege**, die den gesuchten Begriff enthalten.

Dabei ruft sie für jeden Datensatz die Prozedur **Text-Durchsuchen** auf und übergibt den zu durchsuchenden

```
Private Sub cmdSuchen_Click()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strSuche As String
    If Len(Nz(Me!txtSuchbegriff, "")) = 0 Then
        MsgBox "Bitte geben Sie einen Suchbegriff ein."
        Exit Sub
    End If
    strSuche = Me!txtSuchbegriff
    DoCmd.Hourglass True
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblBeitraege WHERE TextRoh LIKE '*" & strSuche & "*'", dbOpenDynaset)
    db.Execute "DELETE FROM tblFundstellen", dbFailOnError
    Do While Not rst.EOF
        TextDurchsuchen rst!TextRoh, strSuche, rst!BeitragID, db
        rst.MoveNext
    Loop
    Me!sfmVolltextsuche.Form.Requery
    If Not DCount("*", "tblFundstellen") = 0 Then
        Me.Filter = ""
        EintragAnzeigen
    Else
        Me!txtCode = Null
        Me.Filter = "1=2"
    End If
    Me.FilterOn = True
    DoCmd.Hourglass False
End Sub
```

Listing 1: Diese Prozedur wird beim Klick auf die Schaltfläche **cmdSuchen** ausgelöst.

Text, den Suchbegriff, die ID des Beitrags sowie einen Verweis auf das **Database**-Objekt der aktuellen Datenbank. Diese Prozedur schauen wir uns gleich im Detail an. Nur so viel: Sie durchsucht den kompletten Text nach Fundstellen und trägt diese dann in die Tabelle **tblFundstellen** ein.

Ist dies erledigt und die Tabelle **tblFundstellen** mit einigen Einträgen gefüllt, wird das Unterformular, das ja an diese Tabelle gebunden ist, aktualisiert.

Sollte keine Fundstelle vorhanden sein, wird der Filter des Hauptformulars über die Eigenschaft **Filter** auf **1=2** eingestellt, was einem Filter entspricht, der keinen Datensatz zurückliefert. Wurden jedoch Einträge gefunden, wird der Filter geleert, sodass alle Einträge der zugrunde liegenden Tabelle über die Datenherkunft verfügbar sind, und die Prozedur **EintragAnzeigen** aufgerufen. Auch diese schauen wir uns weiter unten an. Mit dem Ausblenden des Sanduhr-Symbols ist die Suche beendet.

Durchsuchen eines Beitrags

Die Prozedur **TextDurchsuchen** nimmt sich den in einem Datensatz der Tabelle **tblBeitraege** gespeicherten Text vor und durchsucht ihn nach allen Vorkommen des gesuchten Textes (s. Listing 2). Sie erwartet den Text (**strInhalt**), den Suchbegriff (**strSuche**), die ID des Datensatzes, aus dem

```
Private Sub TextDurchsuchen(strInhalt As String, strSuche As String, lngBeitragID As Long, db As DAO.Database)
    Dim bolErsteZeile As Boolean
    Dim lngPosStart As Long, lngPosEnde As Long, lngStart As Long, lngEnde As Long
    Dim strAusschnitt As String, strSQL As String
    bolErsteZeile = False
    lngPosStart = InStr(1, strInhalt, strSuche)
    Do While Not lngPosStart = 0
        lngStart = InStrRev(strInhalt, vbCrLf, lngPosStart)
        If lngStart = 0 Then
            lngStart = 1
            bolErsteZeile = True
        Else
            bolErsteZeile = False
        End If
        lngStart = InStrRev(strInhalt, vbCrLf, lngStart)
        If lngStart = 0 Then bolErsteZeile = True
        lngEnde = InStr(lngPosStart, strInhalt, vbCrLf)
        lngEnde = InStr(lngEnde + 2, strInhalt, vbCrLf)
        If lngEnde = 0 Then
            lngEnde = Len(strInhalt)
        End If
        If Not bolErsteZeile Then
            If lngEnde > lngStart Then
                strAusschnitt = Mid(strInhalt, lngStart + 2, lngEnde - lngStart - 2)
            Else
                strAusschnitt = Mid(strInhalt, lngStart + 2)
            End If
        Else
            strAusschnitt = Mid(strInhalt, lngStart + 1, lngEnde - lngStart - 1)
        End If
        strAusschnitt = Replace(strAusschnitt, "<", "&lt;")
        strAusschnitt = Replace(strAusschnitt, ">", "&gt;")
        strAusschnitt = Replace(strAusschnitt, strSuche, "<font color=red>" & Mid(strInhalt, lngPosStart, _
            Len(strSuche)) & "</font>")
        strAusschnitt = Replace(strAusschnitt, " <font", " <font")
        lngPosEnde = InStr(lngPosStart + 1, strInhalt, vbCrLf)
        If lngPosEnde = 0 Then
            lngPosEnde = Len(strInhalt)
        End If
        If Not bolErsteZeile Then
            lngPosStart = InStr(lngPosEnde + 1, strInhalt, strSuche)
        Else
            lngPosStart = InStr(lngPosEnde + 1, strInhalt, strSuche) ' + 1
        End If
        strSQL = "INSERT INTO tblFundstellen(BeitragID, Fundstelle) VALUES(" & lngBeitragID & ", '" _
            & Replace(Replace(Replace(strAusschnitt, "'", "''"), vbCrLf, "<br>"), """", """""") & "')"
        db.Execute strSQL, dbFailOnError
    Loop
End Sub
```

Listing 2: Diese Prozedur ermittelt alle Vorkommen des Suchbegriffs.

der Inhalt stammt und einen Verweis auf das **Database**-Objekt der aktuellen Datenbank als Parameter.

Die Prozedur stellt zunächst den Wert der Variablen **bolErsteZeile** auf den Wert **False** ein. Diese Variable gibt an, ob sich bereits in der ersten Zeile eine Fundstelle befindet – wir gehen davon aus, dass dies nicht der Fall ist. Dann ermittelt die folgende Anweisung mit der **InStr**-Funktion die tatsächliche Position der ersten Fundstelle. Ist **lngPosStart**, also die Position der Fundstelle, ungleich **0**, steigt die Prozedur in eine **Do While**-Schleife ein, die solange läuft, bis keine weitere Fundstelle mehr gefunden werden kann, sprich: bis **lngPosStart** gleich **0** ist.

Ist also eine Fundstelle vorhanden, ermittelt die Prozedur zunächst die Position des letzten Zeilenumbruchs (**vbCrLf**) vor der Fundstelle. Befindet sich die Fundstelle also etwa irgendwo in der dritten Zeile des Textes aus **strInhalt**, dann sucht die Prozedur nun die Position des Starts der Zeile und trägt die Position in die Variable **lngStart** ein.

Die hier verwendete Funktion **InStrRev** erhält den Inhalt von **strInhalt** als zu durchsuchenden Text, die Konstante für einen Zeilenumbruch zu suchende Zeichenfolge und die Position der aktuellen Fundstelle als Beginn der Suche.

Diese läuft bei der Funktion **InStrRev** im Gegensatz zur Funktion **InStr** rückwärts, also in Richtung des Anfangs des Textes. Liefert **lngStart** den Wert **0**, befindet sich die Fundstelle tatsächlich in der ersten Zeile. In diesem Falle wird **lngStart** auf **1** eingestellt und **bolErsteZeile** erhält den Wert **True**.

In der aktuellen Version der Prozedur wollen wir immer die Zeile mit der Fundstelle, die Zeile davor und die Zeile danach als Fundstelle speichern. Wir haben nun in **lngStart** die Startposition der Zeile mit der Fundstelle gespeichert.

Nun rufen wir erneut **InStrRev** auf, diesmal mit der Position aus **lngStart** als Startposition, und speichern die Position des letzten vor dieser Position befindlichen Zeilenumbruchs wieder in der Variablen **lngStart**. Wir springen also mit der in **lngStart** gespeicherten Position nicht an den Anfang der Zeile, in der sich der Suchbegriff befindet, sondern an den Anfang der vorherigen Zeile (sofern sich dort noch eine Zeile befindet).

Danach ermitteln wir das Ende der Zeile, in der sich das Suchergebnis befindet, in dem wir mit der **InStr**-Funktion das nächste Vorkommen der Konstanten **vbCrLf** suchen und dessen Position in der Variablen **lngEnde** speichern.

Dies wiederholen wir nochmals ab der Position, die sich zwei Zeichen hinter dieser Position befindet, und ermitteln so die Position des Endes der Zeile hinter der Zeile mit der Fundstelle.

Auch hier kann es wieder sein, dass die Zeile mit der Fundstelle nicht mit einem Zeilenumbruch abgeschlossen wird – was bedeutet, dass die Zeile mit der Fundstelle die letzte Zeile in **strInhalt** ist.

Nun kommt eine Fallunterscheidung, welchen Ausschnitt des Textes wir speichern wollen. Die erste **If...Then**-Bedingung prüft, ob sich die Fundstelle in der ersten Zeile befindet. Falls nicht, folgt eine weitere Fallunterscheidung – nämlich die, ob der Wert von **lngEnde** größer ist als der von **lngStart**. Dies ist nur dann nicht der Fall, wenn **lngEnde** gleich **0** ist, also die Zeile mit der Fundstelle die letzte Zeile in **strInhalt** ist. In diesem Fall liest die Prozedur den Bereich von der Position **lngStart + 2** (also ohne das führende **vbCrLf**) bis zur Position **lngEnde - 2** (ohne das abschließende **vbCrLf**) in die Variable **strAusschnitt** ein.

Sollte **lngEnde** gleich **0** sein und somit nicht größer als **lngStart**, dann liest die Prozedur nur den Ausschnitt von der Position **lngStart + 2** bis zum Ende von **strInhalt** in die Variable **strAusschnitt** ein.

Fehlt noch der Fall, dass es sich bei der Zeile mit der Fundstelle um die erste Zeile handelt, also **bolErsteZeile**