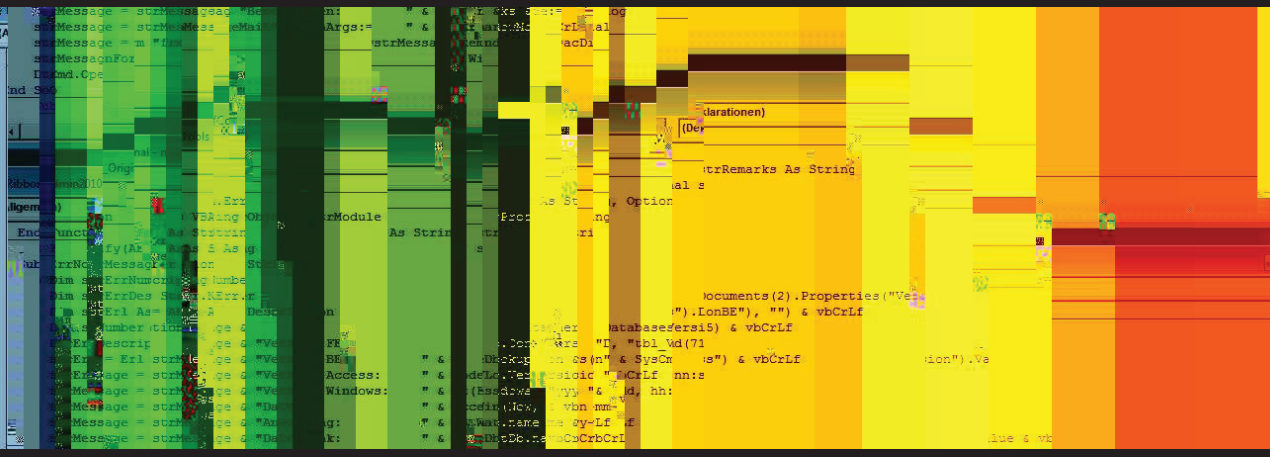
ANWENDUNGEN ENTWICKELN MIT ACCESS



KUNDEN, ARTIKEL, BESTELLUNGEN UND KOMMUNIKATION VERWALTEN MIT ACCESS 2007 UND 2010

ANDRÉ MINHORST

André Minhorst

Anwendungen entwickeln mit Access

Kunden, Artikel, Bestellungen und Kommunikation verwalten mit Access 2007 und 2010

André Minhorst – Anwendungen entwickeln mit Access

Lektorat André Minhorst
Korrektur Rita Klingenstein
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst
Druck und Bindung Kösel, Krugzell (www.koeselbuch.de)

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

ISBN 978-3-944216-00-3

1. Auflage 2012

So klingt Liebe.

Inhalt

Vorwort

Das Vorwort zuletzt zu schreiben, hat einige Vorteile. So kann ich Ihnen beispielsweise jetzt mitteilen, dass ich mir das alles ganz anders vorgestellt hatte. Vor allem die Sache mit dem eigenen Verlag: Ich hatte mir das so gedacht, dass ich damit wesentlich entspannter an das Buch herangehen könnte als bei den vorherigen Büchern für »richtige« Verlage. Die wollen nämlich immer Termine festlegen – vor allem den Abgabetermin.

Und wenn man, wie ich, nicht nur zwei Magazine betreut und mit Texten füllt, sondern auch noch Programmier-Projekte durchführt, kann man nur schlecht einen Abgabetermin für ein Buch angeben. Das heißt: Man gibt natürlich schon einen Termin an, in den man einige Unwägbarkeiten einkalkuliert. Den hält man dann in der Regel nicht ein, denn zu den bereits geplanten Unwägbarkeiten gesellen sich gern noch ungeplante Unwägbarkeiten hinzu (und das passiert nicht nur mir, wie die Erfahrung zeigt).

Also dachte ich mir: Wenn ich selbst der Verlag bin, gibt es zumindest keinen Lektor, den ich immer wieder vertrösten muss, weil ich den Abgabetermin wieder mal nicht halten kann – also gibt es auch kein schlechtes Gewissen. Wenn ich Zeit habe, schreibe ich weiter, wenn nicht, erledige ich halte die übrigen Aufgaben. Leider habe ich aber einen Fehler gemacht: Um abzuklopfen, ob das geplante Buch auf ausreichend Interesse stößt, habe ich am 1. Juni 2012 eine Webseite online gestellt, die erste Informationen zum Buch und eine Möglichkeit zum Bestellen enthielt.

Unverhoffterweise gab es gleich innerhalb weniger Wochen hunderte Bestellungen! Für ein Buch, von dem noch nicht einmal klar war, ob es nur als eBook erscheinen oder auch gedruckt werden würde. Damit hatte sich der Plan, das Buch in ruhigen Phasen zwischen den übrigen Projekten zu schreiben, zerschlagen. Statt einer Lektorin, die sich regelmäßig nach dem aktuellen Stand erkundigte, gab es nun einige hundert zukünftige Leser, die auf das Buch warteten! Ich war noch nie in meinem Leben so euphorisch, denn damit hatte ich in meinen kühnsten Träumen nicht gerechnet. Viele von Ihnen haben zwar schon das eine oder andere von mir gelesen, aber ein Buch blind vorzubestellen, von dem es gerade einmal einen Titel und ein grobes Konzept gab – da kann ich einfach nur danke sagen.

Nach diesen Ereignissen blieb mir nichts anderes übrig, als diesem Projekt eine andere Priorität als geplant einzuräumen. Das fiel mir nicht sonderlich schwer, und an dieser Stelle muss ich mich nochmals bei allen Lesern bedanken, die das Buch bereits vor der Fertigstellung bestellt haben. Da ich Ihnen, lieber Leser, regelmäßig fertige Kapitel zum Download bereitgestellt habe, konnten Sie mir genau das Feedback liefern, um das Buch noch besser an Ihre Wünsche anzupassen und es somit noch besser zu machen.

Nobody Is Perfect

Nun ist es so, dass jedes Manuskript immer noch Fehler enthält – auch wenn es noch so genau geprüft wurde. Das ist auch hier der Fall: Die sprachlichen Fehler hat mir meine langjährige Korrektorin Rita Klingenstein gemeldet. Und die sachlichen Ungenauigkeiten konnte ich durch das umfangreiche Feedback der Vorab-Leser des Buchs verbessern. Allen voran ist hier Thomas Engel zu nennen. Er hat das Buch aus der Sicht eines Einsteigers durchgearbeitet und viele wichtige Anregungen geliefert, die das Buch auch für diese Lesergruppe verständlich machen.

Gut, besser, eBook

Das Buch erläutert die zur Entwicklung einer Anwendung nötigen Schritte. Dabei geht es zu Beginn recht gemächlich zu, sodass Einsteiger normalerweise alles verstehen sollten. In den späteren Kapiteln wird nicht mehr jeder einzelne Schritt erklärt. Dies liegt auch daran, dass sonst der Platz nicht gereicht hätte, um auch für Fortgeschrittene interessante Themen unterzubringen. Sollte ein großes Interesse daran bestehen, die hinteren Kapitel um Schritt-für-Schritt-Anweisungen zu erweitern, werde ich die eBook-Version des Buchs gern erweitern. Schreiben Sie mir in diesem Fall eine E-Mail an *andre@minhorst.com*.

Das eBook ist Bestandteil des Buchs. Wenn Sie das Buch über meine Webseite *www.minhorst.com* gekauft haben, sollten Sie bereits Zugangsdaten für den Zugriff auf das eBook und die Beispieldatenbank erhalten haben. Sollten Sie das Buch anderswo gekauft haben, finden Sie auf der Verpackung einen Aufkleber mit einem Registrierungsschlüssel. Damit können Sie sich auf meiner Webseite registrieren und die gewünschten Dateien herunterladen. Und sollte Ihr Registrierungsschlüssel verloren gegangen sein, melden Sie sich einfach unter *andre@minhorst.com* bei mir – wir werden eine Lösung finden.

Schließlich ist das eBook nicht nur eine einfache Zugabe zum Buch. Ich habe im Buch bewusst auf einen Index verzichtet. Der Grund ist einfach: Ich habe schon einige EDV-Fachbücher gelesen, und es gab noch keines, dessen Index mir die entscheidenden Stellen im Buch offenbart hat. Dies geschah auch bei Büchern, deren Index mehr als 20 Seiten füllte. Statt eines Index verwenden Sie also einfach das eBook, das Sie mit der Suchfunktion des Acrobat Readers leicht nach den gewünschten Stichwörtern durchsuchen können.

Ausblick

Dieses Buch ist der Beginn einer Reihe von Büchern rund um die Anwendungsentwicklung mit Access. Auch wenn es zum Zeitpunkt der Drucklegung dieses Buches noch keine konkrete Planung gibt, so gibt es doch einige Ideen – denen Sie sicher noch einige hinzufügen können.

AEMA, Teil II?

Die ersten Ideen drehen sich um einen zweiten Teil zu diesem Buch, der die beschriebene Anwendung um weitere Funktionen erweitert – zum Beispiel die Folgenden:

» Erweiterung der Anwendung um ein Mahnwesen

» Erstellung eines Setups zur Weitergabe

» Schützen der Anwendung

» Integration einer Benutzerregistrierung

» Automatische Aktualisierung über einen Server beziehungsweise über das Internet

» Umwandlung in eine Mehrbenutzeranwendung

» Prüfung der Gültigkeit von Adressen über das Internet

» Einbindung des Zahlungsverkehrs über Paypal

» Import und Export von Adressen aus Outlook

» Kunden-E-Mails gleich nach dem Eingang in Outlook in die Datenbank übertragen

» Versenden von Newslettern an Kunden

» Auswertungen von Umsätzen et cetera in Berichten

Migration und Betrieb nach SQL Server/MySQL?

Ein weiteres mögliches Thema für ein weiteres Buch ist die Migration der im vorliegenden Buch beschriebenen Anwendung nach Microsoft SQL Server oder MySQL. Dieses würde Themen wie die folgenden behandeln:

» Migration der Tabellen in das DBMS

» Einbinden der Tabellen in Access

» Migration der Abfragen zu Sichten oder gespeicherten Prozeduren

» Übertragung von VBA-Routinen in gespeicherte Prozeduren, Funktionen oder Trigger

» Einführung in T-SQL

» Optimierung der Anwendung hinsichtlich der Datenbindung von Formularen, Berichten und Steuerelementen sowie des Datenzugriffs von VBA-Routinen aus

Teilen Sie mir einfach mit, was Sie von diesen Ideen halten. Und wenn Ihnen eines dieser geplanten Bücher gefallen sollte, Sie aber noch Themen vermissen, ist Ihre Meinung herzlich willkommen – einfach per E-Mail an *andre@minhorst.com*.

Auf geht's!

Ich wünsche Ihnen viel Spaß und viele neue Erkenntnisse bei der Lektüre. Die hier beschriebene Lösung wird nicht auf jeden Anwendungsfall passen, aber das ist auch nicht das Ziel – zunächst einmal sollen Sie erfahren, wie Sie selbst eine solche Anwendung aufbauen. Davon abgesehen bin ich sicher, dass Sie viele der vorgestellten Techniken für Ihre eigenen Anwendungen nutzen können.

Über Feedback freue ich mich jederzeit. Sollten Sie Korrekturen und Verbesserungsvorschläge haben, freue ich mich – ich werde diese in das eBook einarbeiten und den jeweils aktuellen Stand auf der Webseite für alle Buchkäufer zum Download bereitstellen.

Danke nochmals an Sie – dafür, dass Sie meine Arbeit durch den Kauf dieses Buchs unterstützen.

Duisburg, 17. September 2012

André Minhorst

1 Vorbereitungen

Bevor Sie sich an die Entwicklung einer Anwendung begeben, sollten Sie einige Dinge vorbereiten. Dies betrifft vor allem die Entwicklungsumgebung, aber auch eventuell benötigte zusätzliche Tools.

1.1 Access-Optionen

Einige der nachfolgend vorgestellten Optionen finden Sie in den Access-Optionen, andere in den Optionen des VBA-Editors. Die Access-Optionen öffnen Sie, indem Sie unter Access 2007 das Office-Menü durch einen Klick auf den Office-Button öffnen und dort auf die Schaltfläche *Access-Optionen* klicken. Unter Access 2010 zeigen Sie den Backstage-Bereich an und wählen dort den Eintrag *Optionen* aus.

1.1.1 Dokumentfensteroptionen

Bis zur Version 2003 hat Access Formular- und Berichtsfenster standardmäßig als eigene Fenster innerhalb des Access-Fensters angezeigt. Diese konnten Sie durch Maximieren komplett in das Access-Fenster einpassen. Wenn Sie dieses Verhalten wünschen, müssen Sie in den Access-Optionen unter *Aktuelle Datenbank* die Option *Anwendungsoptionen|Dokumentfensteroptionen* auf den Wert *Überlappende Fenster* einstellen (siehe Abbildung 1.1).

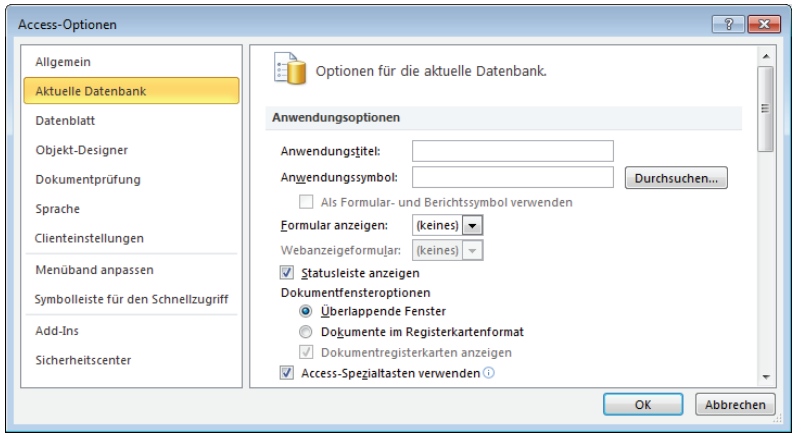


Abbildung 1.1: Aktivieren der Anzeige für überlappende Fenster

Die Alternative sind Formulare und Berichte im Registerkartenformat. Diese werden immer ins Access-Fenster eingepasst. Zusätzlich können Sie dafür sorgen, dass der Formular-/Berichtsname

beziehungsweise der festgelegte Fenstertitel in einer Registerkarte angezeigt wird. Dazu aktivieren Sie zusätzlich die Option *Dokumentregisterkarten anzeigen*. Dies ermöglicht den schnellen Wechsel zwischen mehreren zeitgleich geöffneten Access-Objekten.

Wenn Sie keine Registerkarten anzeigen, gibt es allerdings keine eingebaute Möglichkeit mehr, ein Formular zu schließen. Sie müssen dann also eine Schaltfläche zum Schließen vorsehen oder den Benutzern der Anwendung die Tastenkombination *Strg + F4* nahebringen.

Dies gilt natürlich auch für Tabellen und Abfragen. Da eine professionelle Anwendung dem Benutzer nur Daten in Form von Formularen oder Berichten präsentiert, spielt dies hier keine Rolle.

Grundsätzlich ist die Verwendung des Registerkartenformats eine gute Idee. Formulare und Berichte werden so immer maximiert angezeigt. Gleichzeitig können Sie Formulare und Berichte als überlappende Fenster öffnen, wenn dies nötig ist.

Dazu geben Sie für die Methoden *DoCmd.OpenForm* beziehungsweise *DoCmd.OpenReport* einfach den Wert *acDialog* für den Parameter *WindowMode* an – mehr dazu in den folgenden Kapiteln.

Im Rahmen der Entwicklung der Lösung zu diesem Buch stellen wir die Option jedoch auf *Überlappende Fenster* ein. Warum dies? Ganz einfach: Weil es so wesentlich einfacher ist, Screenshots zu erstellen – und davon gibt es in diesem Buch eine ganze Reihe.

1.1.2 Entwurf im Datenblatt deaktivieren

Für Neulinge und Quereinsteiger sehen Access 2007 und 2010 einen Modus vor, der das Anlegen von Tabellenfeldern in der Datenblattansicht erlaubt. Letztlich ist dies eine alternative Ansicht zur Entwurfsansicht, die einen intuitiveren Umgang ermöglichen soll. In diesem Buch wird jedoch die Entwurfsansicht verwendet.

Und wenn wir dies schon tun, können wir das Bearbeiten von Tabellen in der Datenblattansicht auch gleich deaktivieren. Dies erledigen Sie ebenfalls im Dialog *Access-Optionen*, und zwar unter *Aktuelle Datenbank|Anwendungsoptionen|Entwurfsänderungen für Tabellen in der Datenblattansicht aktivieren* (siehe Abbildung 1.2, obere Markierung).

1.1.3 Quellbildformat beibehalten

In der Lösung zum Buch werden einige Bilder gespeichert – sei es als Icon einer Schaltfläche, als Hintergrund eines Formulars oder Berichts oder auch als Icon eines Ribbon-Eintrags.

Standardmäßig speichert Access solche Bilder in einem proprietären Format, das die Bilder unnötig aufbläht und so die Datenbank zu Ungunsten der zu speichernden Daten vergrößert. Dies können Sie verhindern, indem Sie die Option *Bildeigenschaften-Speicherformat* auf den Wert *Quellbildformat beibehalten* einstellen (siehe Abbildung 1.2, untere Markierung).

1.1.4 Optionen für Indizes

Wenn Sie Felder zu einer Tabelle hinzufügen, erstellt Access in manchen Fällen automatisch Indizes für diese Felder. Dies geschieht beispielsweise, wenn der Feldname den Ausdruck *ID* enthält.

Dies ist kein Zufall: In den Access-Optionen finden Sie unter *Objekt-Designer|Entwurfsansicht für Tabellen* den Eintrag *AutoIndex beim Importieren/Erstellen* (siehe Abbildung 1.2). Wenn Sie selbst die komplette Kontrolle über die Erstellung von Indizes behalten möchten, leeren Sie diese Option (siehe Abbildung 1.3).

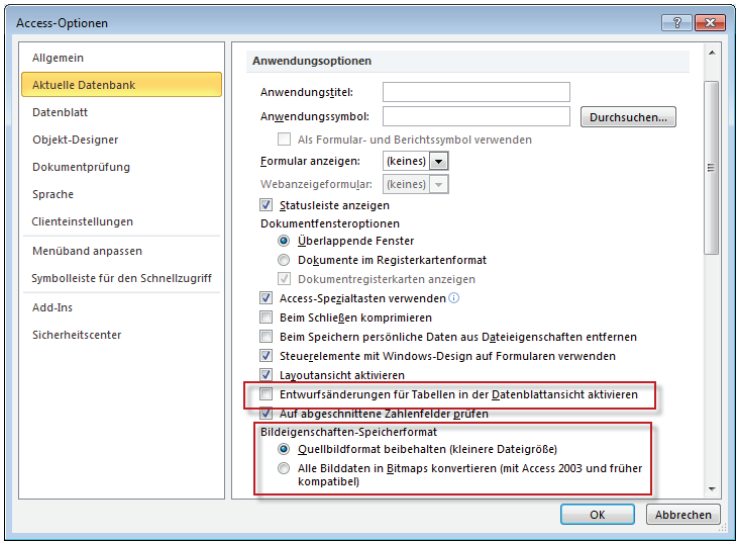


Abbildung 1.2: Weitere wichtige Optionen

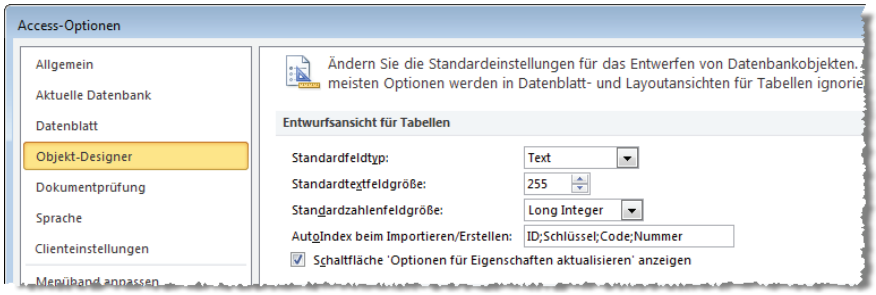


Abbildung 1.3: Einstellungen für das Erstellen von Tabellenfeldern

In diesem Dialog finden Sie noch weitere Einstellungen, die sich auf das Erstellen von Tabellenfeldern auswirken. Sie können diese jedoch bei den Standardwerten belassen.

1.1.5 Voreinstellungen für Formular- und Steuerelementeigenschaften

Access verwendet beim Erstellen von Steuerelementen eine Standardschriftart und -größe. Mir persönlich ist die Schriftgröße *11* bei der Schriftart *Calibri* zu groß. Sie können jedoch die Schrifteigenschaften für Steuerelemente mit Text auf einfache Weise festlegen. Dazu legen Sie je eine Formular- und eine Berichtsvorlage für die aktuelle Datenbank an. Wie diese Access-Objekte heißen, legen die Access-Optionen *Formularvorlage* und *Berichtsvorlage* fest (siehe Abbildung 1.4).

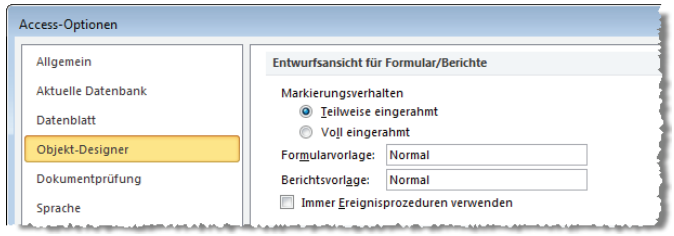


Abbildung 1.4: Vorgeben der Namen für Standardformular und -bericht

Die Formulare der Anwendung sollen standardmäßig schwarzen Text mit der Schriftart *Calibri* und der Schriftgröße *9* verwenden. Um dies zu realisieren, erstellen Sie ein neues Formular und speichern es unter dem für die Option *Formularvorlage* angegebenen Namen. Danach legen Sie die Standardeigenschaften fest. Im Detail sieht das so aus:

» Öffnen Sie ein neues Formular mit dem Ribbon-Eintrag *Erstellen|Formularentwurf*.

» Klicken Sie auf das Steuerelement, dessen Standardeigenschaften Sie ändern möchten, also beispielsweise ein Textfeld (siehe Abbildung 1.5).

» Das Eigenschaftsfenster zeigt nun die Eigenschaften für das Element *Auswahltyp: Standard: Textfeld* an.

» Ändern Sie die gewünschten Eigenschaften, beispielsweise die Schriftgröße.

» Fügen Sie das Steuerelement nicht wie üblich hinzu, sondern speichern und schließen Sie das Formular unter dem Namen *Normal*.

Wenn Sie nun ein neues Formular anlegen und ein Textfeld hinzufügen, wird dieses gleich mit der zuvor festgelegten Schriftgröße angelegt. Dummerweise erscheint das Bezeichnungsfeld noch in der alten Schriftart (siehe Abbildung 1.6). Das ist jedoch kein Problem: Sie ändern einfach die Standardeigenschaften für das Bezeichnungsfeld so, dass die Schriftgröße zu der des Textfeldes passt. Diese Schritte führen Sie für alle Steuerelemente durch, die Text anzeigen, also Bezeichnungsfelder, Textfelder, Kombinationsfelder, Listenfelder, Schaltflächen, Umschaltflächen und Registersteuerelemente.

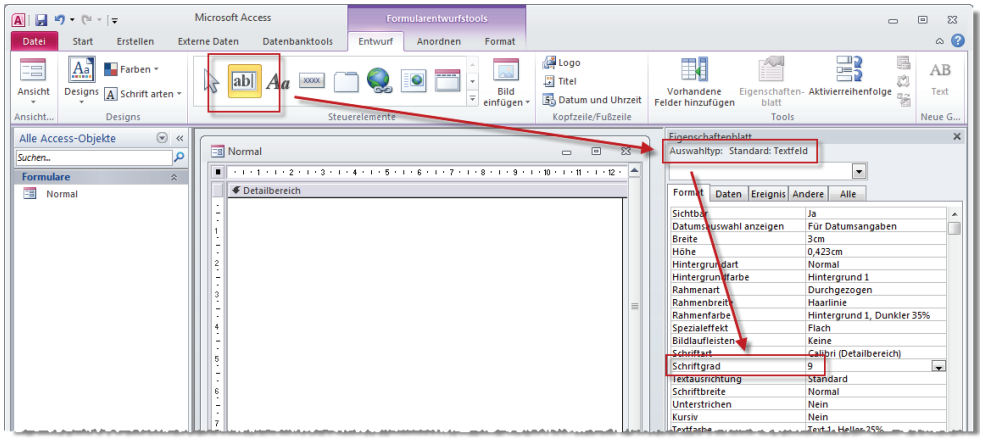


Abbildung 1.5: Standardeigenschaften für Steuerelemente setzen

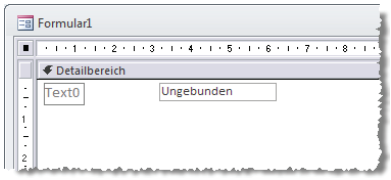


Abbildung 1.6: Anlegen eines Textfeldes mit neuer Standard-Schriftgröße

Es gibt noch weitere recht unbekannte, aber interessante Eigenschaften für Standardsteuerelemente: Mit der Eigenschaft *Mit Bezeichnungsfeld* können Sie festlegen, ob Sie überhaupt ein Bezeichnungsfeld für ein Text-, Kombinations- oder Listenfeld anzeigen möchten. Und die Eigenschaft *Mit Doppelpunkt* legt fest, ob Access automatisch einen Doppelpunkt zum Bezeichnungsfeld hinzufügt. Dies ist vor allem bei Verwendung gebundener Felder in Formularen interessant, welche die Daten einer Tabelle oder Abfrage anzeigen (siehe Abbildung 1.7). Die weiteren Eigenschaften *BezeichnungsfeldX*, *BezeichnungsfeldY* und *Bezeichnungsausrichtung* geben an, wie das Bezeichnungsfeld standardmäßig platziert und ausgerichtet werden soll.

1.1.6 Voreinstellungen für Berichte

Auch in Berichten können Sie Voreinstellungen vornehmen. Dies geschieht analog zur Vorgehensweise in Formularen. Auch den Bericht mit den Voreinstellungen speichern Sie unter dem Namen *Normal*.

Für die Berichte dieser Datenbank wurde vor allem eine Einstellung vorgenommen: Standardmäßig werden Textfelder und andere Steuerelemente von Access mit Rahmen zu einem Bericht hinzugefügt. Das ist zumindest für einen Rechnungsbericht, wie wir ihn später anlegen werden, nicht gewünscht.

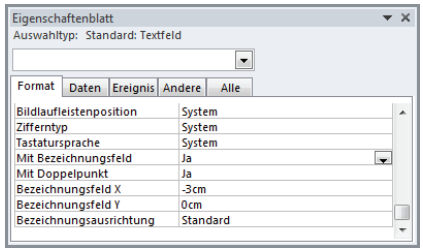


Abbildung 1.7: Bezeichnungsfeld-Eigenschaften für Textfelder, Kombinationsfelder und Listenfelder einstellen

Also stellen Sie für alle Standardsteuerelemente die Eigenschaft *Rahmenart* auf *Transparent* ein. Auch Schriftart und Farbe sollten Sie so anpassen, dass diese Eigenschaften für die meisten Steuerelemente passen – so müssen Sie später am wenigsten Aufwand für die Anpassung betreiben.

Außerdem sind in der *Normal*-Vorlage alle Berichtsbereiche auf einen weißen Hintergrund eingestellt und die Eigenschaft *Alternierende Hintergrundfarbe* auf den gleichen Wert wie *Hintergrundfarbe*.

1.1.7 Ereignisprozeduren schnell anlegen

Sie werden beim Nachbauen der Lösung dieses Buchs eine Reihe Ereignisprozeduren anlegen. Normalerweise bietet Access nach einem Klick auf die Schaltfläche mit den drei Punkten neben einer Ereigniseigenschaft einen Dialog mit mehreren Werten zur Auswahl an. Sie wählen dann den Eintrag *Code-Generator* aus und klicken auf die Schaltfläche mit den drei Punkten, damit Access die entsprechende Ereignisprozedur im VBA-Editor anlegt.

Wir benötigten aber ohnehin nur Ereigniseigenschaften. Wenn Sie die Option *Immer Ereignisprozeduren verwenden* aus Abbildung 1.8 aktivieren, brauchen Sie nur noch auf die Schaltfläche mit den drei Punkten zu klicken. Der Dialog entfällt, Access trägt automatisch den Wert *[Ereignisprozedur]* für die jeweilige Ereigniseigenschaft ein und öffnet den VBA-Editor mit der neuen Ereignisprozedur.

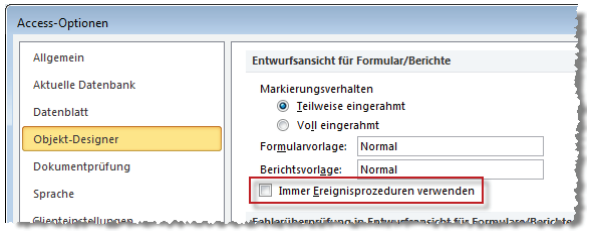


Abbildung 1.8: Standardmäßig Ereignisprozeduren für Ereigniseigenschaften hinterlegen

1.2 Optionen im VBA-Editor

Die Optionen des VBA-Editors zeigen Sie an, indem Sie den Eintrag *Extras|Optionen* der Menüleiste des VBA-Editors auswählen.

Option Explicit aktivieren

Viele Laufzeitfehler resultieren daraus, dass Variablen nicht deklariert wurden. Dies können Sie einfach unterbinden, indem Sie im Kopf eines jeden VBA-Moduls die folgende Zeile unterbringen:

```
Option Explicit
```

Dies sollten Sie jedoch nicht manuell erledigen, da Sie es so doch einmal vergessen könnten. Stattdessen aktivieren Sie gleich als Erstes die Option *Variablendeklaration erforderlich* (siehe Abbildung 1.9). Wenn Sie danach ein neues Modul anlegen, wird die Anweisung *Option Explicit* automatisch im Kopf des Moduls eingefügt.

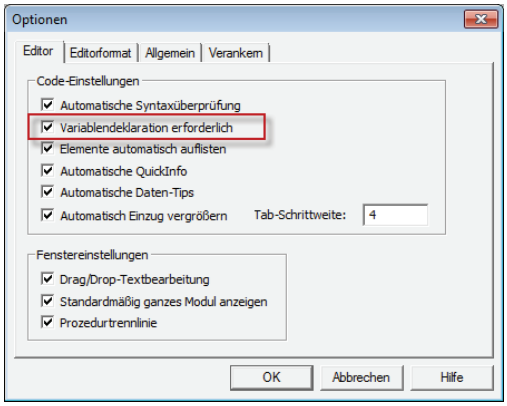


Abbildung 1.9: Erzwingen der Variablendeklaration per Option

1.3 Tools installieren

Bei der Programmierung können einige kleine Helferlein sehr nützlich sein. Diese werden in den folgenden Abschnitten vorgestellt.

1.3.1 MZ-Tools

Die MZ-Tools sind ein COM-Add-In für den VBA-Editor. Sie finden diese unter folgendem Link:

```
http://www.mztools.com/v3/download.aspx
```

Diese Toolsammlung liefert folgende wichtige Funktionen:

» Archivieren und Wiederherstellen von Codevorlagen

» Nummerieren von Zeilen

» Hinzufügen von Fehlerbehandlungen

» Suche mit übersichtlicher Liste der Suchergebnisse

» Suchen nach Aufrufen bestimmter Prozeduren

» Anzeige nicht verwendeter Variablen und Parameter

Meine Sammlung von Code-Vorlagen finden Sie im Download zu diesem Buch. Wenn Sie die Einstellungen ändern oder erweitern möchten, scheitern Sie möglicherweise an den Sicherheitseinstellungen.

Um dies zu umgehen, zeigen Sie die Datei *MZTools3vba.ini* im Windows Explorer an. Wählen Sie den Eintrag *Eigenschaften* aus dem Kontextmenü aus und betätigen Sie die Schaltfläche *Bearbeiten* auf der Registerkarte *Sicherheit*.

Im Dialog *Berechtigungen für "MZTools3VBA.ini"* stellen Sie die Eigenschaft *Ändern* auf *Zulassen* ein (siehe Abbildung 1.10).

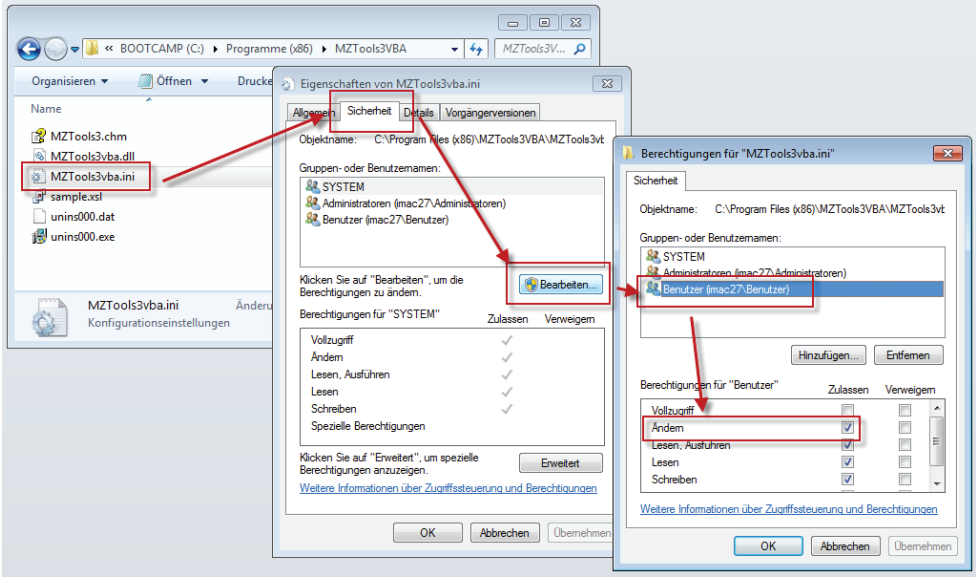


Abbildung 1.10: Sicherheitseinstellungen für die Konfigurationsdatei von MZ-Tools

Das Nummerieren von Zeilen und das Hinzufügen von Fehlerbehandlungen entfällt, wenn Sie das Tool *vbWatchdog* verwenden – mehr dazu weiter unten.

1.3.2 vbWatchdog

Dieses Tool von Wayne Philips liefert eine komplette Fehlerbehandlung mit viel mehr Funktionen als die eingebaute Fehlerbehandlung von VBA. Wir stellen dieses Tool im Kapitel »Fehlerbehandlung« (Seite 449) vor. Das Tool ist kostenpflichtig, aber es lohnt sich:

```
http://www.everythingaccess.com/vbwatchdog.htm
```

1.3.3 ProcBrowser

Der *ProcBrowser* von Sascha Trowitzsch zeigt alle Elemente des aktuellen VBA-Moduls in einem eigenen Fenster an. Dies ist vor allem enorm hilfreich, wenn ein Modul mehr Code enthält, als eine einzige Bildschirmseite anzeigen kann. Sie brauchen dann nur auf den Namen etwa einer Prozedur zu klicken, um im Codefenster zur gewünschten Stelle zu springen (siehe Abbildung 1.11). Den Download finden Sie hier:

```
http://www.mosstools.de/index.php?option=com_content&view=article&id=58&Itemid=67
```

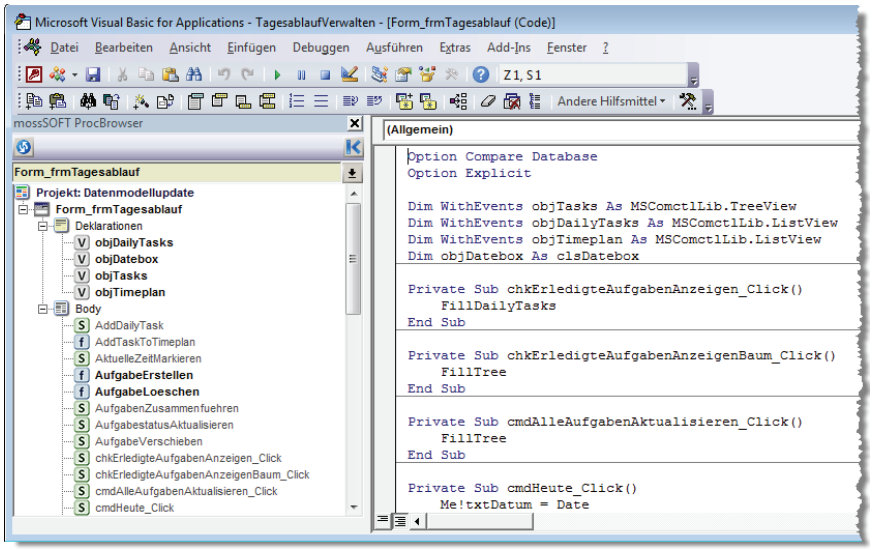


Abbildung 1.11: Der Procbrowser in Aktion

Hinweise zur Installation finden Sie auf der Webseite.

1.3.4 Beispieldaten-Assistent

Der Beispieldaten-Assistent unterstützt Sie beim Anlegen von Testdaten, insbesondere für Adressen-Tabellen. Sie finden das Add-In im Download zum Buch.

Das Add-In installieren Sie, indem Sie es zunächst herunterladen und entpacken. Öffnen Sie dann Access und rufen Sie mit dem Ribbon-Eintrag *Datenbanktools|Add-Ins|Add-In-Manager* den Add-In-Manager auf.

Klicken Sie dort auf die Schaltfläche *Hinzufügen...* (siehe Abbildung 1.12).

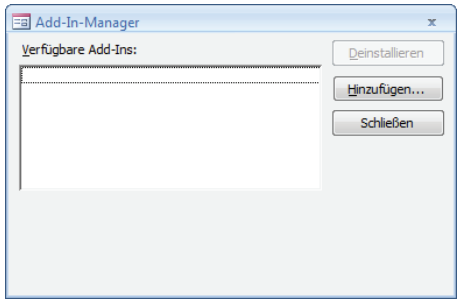


Abbildung 1.12: Der Add-In-Manager von Access

Wählen Sie im folgenden Dialog die Add-In-Datenbank namens *Beispieldaten-Assistent.mda* aus. Anschließend wird diese in der Liste der verfügbaren Add-Ins angezeigt (siehe Abbildung 1.13).

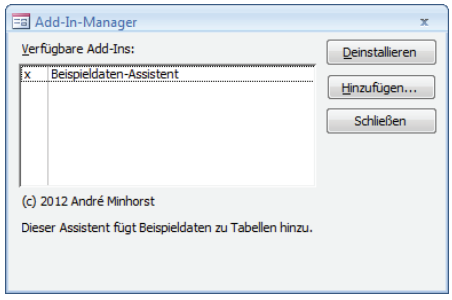


Abbildung 1.13: Add-In-Manager mit installiertem Beispieldaten-Assistent

1.3.5 Ribbon-Admin 2010

Der *Ribbon-Admin 2010* unterstützt Sie beim Erstellen von Ribbon-Definitionen unter Access 2007 und 2010. Normalerweise ist dies ein mühseliger Vorgang, weil Sie erst von Hand das Aussehen und die Attribute des Ribbons im XML-Format zusammenstellen und dann noch die eventuell benötigten VBA-Callback-Prozeduren hinzufügen müssen.

Der Ribbon-Admin 2010 bietet hierzu eine Benutzeroberfläche und erleichtert die Arbeit enorm (siehe Abbildung 1.14). Auch dieses Tool ist kostenpflichtig, allerdings rentiert sich die Investition bereits nach kurzer Zeit. Den Download finden Sie hier:

```
http://www.ribbon-admin.de
```

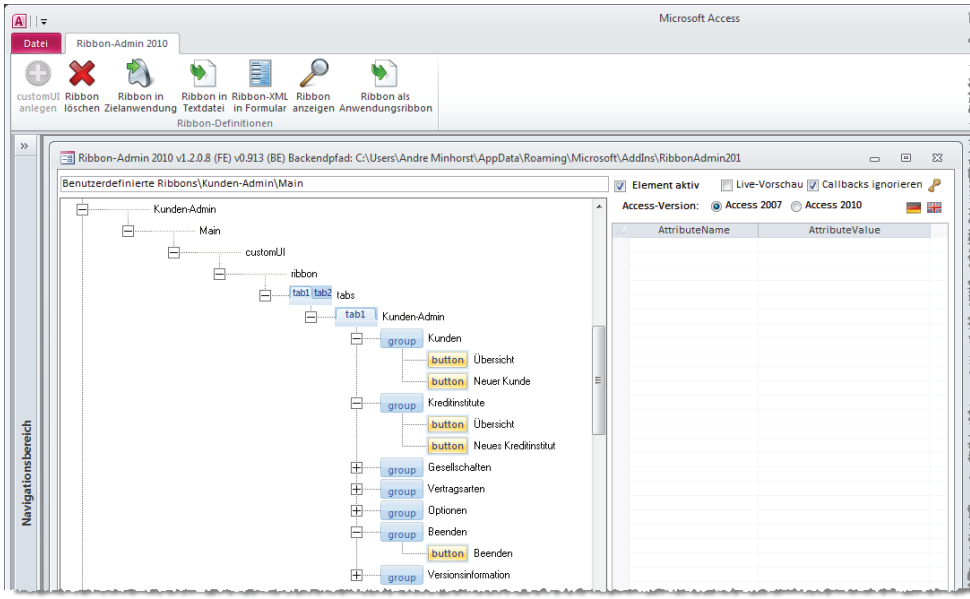


Abbildung 1.14: Der Ribbon-Admin 2010 erstellt Ribbon-Definitionen per grafischer Benutzeroberfläche.

Ribbon-Admin 2010 installieren

Den Ribbon-Admin 2010 installieren Sie genau wie den Beispieldaten-Assistent.

1.3.6 Control-Renamer

Der *Control-Renamer* hilft, gebundene Steuerelemente mit Präfixen zu versehen. In vielen Fällen kann es wichtig sein, dass Sie per Code auf ein Steuerelement zugreifen und nicht auf das Feld der Datenherkunft, an das dieses Steuerelement gebunden ist. Dies ist nämlich durchaus ein Unterschied: So bieten zwar beide Zugriff auf den enthaltenen Wert, aber nur das Steuerelement lässt das Einstellen von Eigenschaften wie etwa *Standardwert* zu.

Wenn Sie ein Feld aus der Feldliste in den Entwurf eines Formulars ziehen, um ein darauf basierendes Steuerelement zu erstellen, benennt Access das Steuerelement entsprechend dem Feldnamen. Dies ändern Sie ganz einfach mit dem *Control-Renamer* aus Abbildung 1.15.

Dieser zeigt automatisch alle Formulare und Berichte an, die noch gebundene Steuerelemente mit dem Originalnamen enthalten, und fügt nach Wunsch Präfixe wie *txt* für Textfelder oder *cbo* für Kombinationsfelder hinzu.

Dieses Add-In finden Sie ebenfalls im Download zum Buch.

Der *Control-Renamer* ist genau wie der Beispieldaten-Assistent ein Access-Add-In, deshalb können Sie diesen wie den Beispieldaten-Assistent installieren.

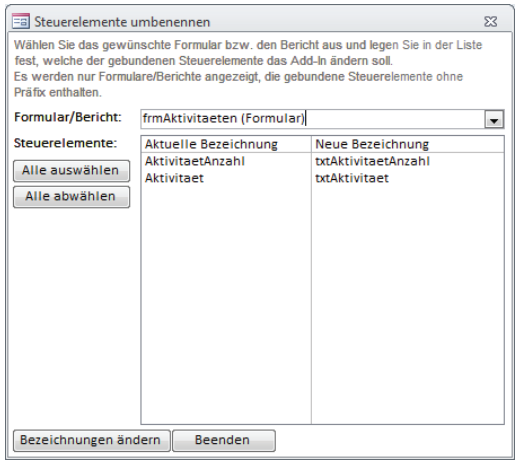


Abbildung 1.15: Einfaches Umbenennen gebundener Steuerelemente

1.3.7 Datenzugriffscode

Wenn Sie mit Access programmieren, werden Sie früher oder später Code erzeugen, mit dem Sie per DAO oder SQL auf Tabellen und ihre Felder zugreifen, um Datensätze zu einer Tabelle hinzuzufügen oder zu ändern. Gerade wenn es sich um Änderungen mit mehreren Feldern handelt, die beispielsweise über die Parameter der Prozedur geliefert werden, macht diese Arbeit nicht unbedingt viel Spaß und ist überdies fehleranfällig. Mit dem Access-Add-In *aiuDatenzugriffscode* erledigen Sie solche Aufgaben mit wenigen Mausklicks. Falls nicht, legen Sie aber zumindest den Grundstein. Abbildung 1.16 zeigt die Benutzeroberfläche dieses Add-Ins, das Sie genau wie die übrigen Add-Ins über den Add-In-Manager anlegen. Das Tool bietet folgende Features:

» Auswahl der Tabelle

» Auswahl der zu ändernden Felder

» Auswahl der als Kriterium beim Öffnen zu verwendenden Felder (mit *AND* oder *OR* verknüpfbar)

» Übergabe der Variablen als Parameter der Prozedur oder Festlegen der Variablen innerhalb der Prozedur

» Zugriffsarten *Anfügen* oder *Bearbeiten*

» Zugriffstechniken *DAO* oder *SQL*

» Kopieren des Codes in die Zwischenablage

» Verwenden der Funktion *IsoDatum* zum Formatieren von Datumsangaben in SQL-Strings

» Ersetzen von Komma durch Punkt in Dezimalzahlen

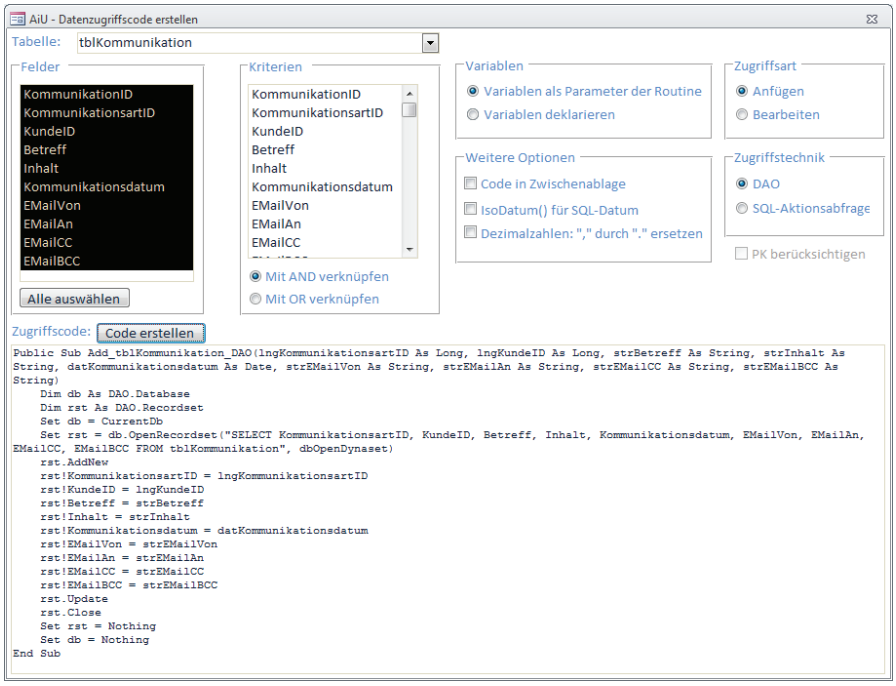


Abbildung 1.16: Das Formular zum Add-In *aiuDatenzugriffscode*

Das Tool finden Sie ebenfalls im Download zum Buch.

1.3.8 Die Access-Runtime

Microsoft bietet seit *Access 2007* den kostenlosen Download der Runtime-Version von Access an. Die Runtime-Version ist eine Version, mit der Endbenutzer die von Ihnen programmierten Anwendungen ausführen können. Sie dürfen diese kostenlos weitergeben, daher entstehen dem Benutzer keine zusätzlichen Kosten für die Anschaffung von Access. Den Download für die Runtime in den Versionen für Access 2007 und Access 2010 finden Sie hier:

`http://acciu.de/aemaruntime2007`
`http://acciu.de/aemaruntime2010`

Die Runtime-Version bietet allerdings ausschließlich die in Ihrer Anwendung bereitgestellten Funktionen an – jegliche in der Vollversion enthaltenen Features wie der Navigationsbereich, die Entwurfsansichten oder der VBA-Editor fehlen. Das bedeutet, dass Sie alle Funktionen Ihrer Anwendung über entsprechende Übersichtsformulare oder das Ribbon anbieten müssen. Außerdem gibt es ein paar weitere Einschränkungen – so können Sie mit der Runtime keinerlei Änderungen am Entwurf von Datenbankobjekten wie Tabellen, Abfragen, Formularen oder Berichten vornehmen. Dazu gehört auch, dass der Benutzer etwa die Spaltenbreiten in

Formularen in der Datenblattansicht ändern kann, aber im Gegensatz zur Vollversion speichert die Runtime diese Änderungen nicht!

Es gibt noch einen weiteren wichtigen Punkt, den Sie bei der Weitergabe von Anwendungen mit der Access-Runtime beachten müssen: Gerade seit Access 2007 stellt sich heraus, dass verschiedene Access-Versionen auf dem gleichen Rechner sich nicht besonders gut vertragen.

Wenn der Benutzer also beispielsweise Access 2003 benutzt, um eigene Datenbanken zu entwickeln, und parallel immer wieder mit einer Access 2007-Datenbank auf Basis der Access-Runtime arbeitet, dauert das abwechselnde Öffnen der verschiedenen Access-Versionen sehr lange – zu lange, um dies zu übergehen.

Ein zweites Problem ist es, dass Sie sicherstellen müssen, dass die mit der Runtime gelieferte Anwendung in einem Verzeichnis liegen muss, das als vertrauenswürdig gekennzeichnet ist. Dies lässt sich per VBA-Code nicht so einfach bewerkstelligen, vor allem dann nicht, wenn die Datenbank-Anwendung, die diesen VBA-Code ausführen soll, nicht vertrauenswürdig ist und daher keinen VBA-Code ausführen darf.

Während die Vollversion von Access in ihren Optionen eine Einstellung für vertrauenswürdige Verzeichnisse bereithält, fehlt diese in der Runtime-Version – der Endbenutzer kann die Anwendung also auch nicht so einfach starten.

Abhilfe für diese beiden und weitere Probleme bietet ein Tool der Firma Sagekey (*http://sagekey.com/installation_access.aspx*). Sagekey bietet mit den *Access Deployment Tools* ein Werkzeug, mit dem Sie Setup-Dateien erstellen können, welche Ihre Anwendung so installieren, dass weder Sicherheitsprobleme noch Verzögerungen beim Wechseln zwischen zwei Access-Versionen auftreten.

Dieses Tool ist nicht ganz billig (650,- $ je Access-Version)! Ein ernsthafter Access-Entwickler, der Software für die Weitergabe an Kunden programmiert, die nicht alle über die Access-Vollversion verfügen, wird aber um diese Investition nicht herumkommen.

Kleiner Tipp: Wenn Sie beispielsweise eine Individualsoftware für einen Kunden programmieren und diesem mit der Runtime-Version die Anschaffung einiger Access-Lizenzen ersparen, sollten Sie ein paar hundert Euro für die Anschaffung des Sagekey-Tools herausschlagen können.

1.3.9 Bereitstellen von Testumgebungen

Im Rahmen der Entwicklung der in diesem Buch vorgestellten Lösung wurden Tests auf verschiedenen Plattformen durchgeführt. Dies mag nicht auf den ersten Blick zwingend erforderlich sein, denn eigentlich sollte eine weitgehend mit Bordmitteln erstellte Anwendung zumindest unter aktuellen Kombinationen aus Betriebssystem und Office-Paket laufen. Auf den zweiten Blick werden die Anwender Ihrer Software diese durch die pure Benutzung auf Herz und Nieren testen, und zwar unter Bedingungen, die ein einzelner Software-Entwickler gar nicht alle berücksichtigen kann. Für die zu berücksichtigenden Konstellationen beziehen wir uns also auf die

zu erwartenden Szenarien. Nach der Auslieferung werden ohne Zweifel Fehlermeldungen eintrudeln, die unter speziellen Bedingungen entstanden sind. Dabei kann es beispielsweise sein, dass ein Formular mit einem TreeView-Steuerelement nicht geladen werden kann, weil auf dem Rechner eine veraltete Version der Datei *MSCOMCTL.OCX* vorliegt.

Kommen wir zurück zu den standardmäßig zu berücksichtigenden Szenarien. Wenn Sie Ihre Software speziell für einen Kunden programmieren, werden Sie mit diesem klären, welche die Mindestkonfiguration bezüglich des Betriebssystems ist – beispielsweise Access 2007 und Windows 7. Dann testen Sie Ihre Anwendung auch nur auf Kombinationen wie Access 2007/Windows 7 und Access 2010/Windows 7.

Wenn Sie eine Anwendung programmieren, die Sie einer breiten Kundengruppe zur Verfügung stellen möchten, indem Sie diese beispielsweise auf Ihrer Webseite zum Verkauf anbieten, sollten Sie ein möglichst breites Spektrum abdecken – dies gilt vor allem für die Betriebssysteme. Es gibt noch eine große Menge Kunden, die mit Windows XP arbeiten – und dabei gibt es mit Windows 8 schon den dritten Nachfolger.

Virtuelle Maschinen

Auch wenn der eine oder andere Entwicklerkollege auf »echte« Testmaschinen schwört, die je nach Bedarf mit Images bestückt werden, habe ich persönlich nicht genügend Platz für mehrere Rechner und vertraue darauf, dass virtuelle Maschinen adäquat arbeiten.

Ich habe mich vor einiger Zeit für den Einsatz von *VMWare Workstation* entschieden, aktuell arbeite ich mit Version 6.5.5. *VMWare* gibt es sogar in einer für den privaten Gebrauch kostenlosen Version namens *VMWare Player*. Weitere Informationen hierzu finden Sie hier:

```
http://acciu.de/aemavmware
```

1.4 Anwendungsoptionen

Es gibt nicht nur Optionen für Access und den VBA-Editor, sondern auch anwendungsbezogene Optionen. Dabei handelt es sich um solche Optionen, die in einer eigenen Tabelle der Anwendung gespeichert und bei Bedarf gelesen oder geändert werden.

Das einfachste Beispiel für eine solche Option ist die Anwendungsversion. Sie sollten diese mit jeder Änderung der Anwendung aktualisieren. Der Hintergrund ist ganz einfach: Beim Auftreten von Laufzeitfehlern soll die Anwendung dem Entwickler eine E-Mail mit Fehlerinformationen schicken, damit der Entwickler diesen beheben kann.

Diese E-Mail soll nicht nur Informationen über den Fehler selbst enthalten, sondern auch über die verwendete Version der Anwendung. Es kann ja immerhin vorkommen, dass der Benutzer eine ältere Version verwendet, obwohl der Fehler in einer neueren Version längst behoben ist – und darüber gibt die Versionsnummer einer Anwendung Auskunft.

Legen Sie also eine Tabelle zum Speichern der Optionen in der Datenbank an. Diese soll ein Feld je Option enthalten, damit der Datentyp des betroffenen Wertes der Option entsprechend berücksichtigt werden kann. Die Optionentabelle soll *tblOptionen* heißen und zunächst nur ein Feld zum Speichern der Anwendungsversion enthalten. Im weiteren Verlauf des Buchs wird die Tabelle allerdings noch erweitert werden.

Legen Sie mit *Erstellen|Tabellen|Tabellenentwurf* eine neue Tabelle in der Entwurfsansicht an und fügen Sie das Feld *Version_FE* mit dem Datentyp *Text* hinzu. Wofür stehen die beiden Buchstaben *FE*? Sie stehen für den Begriff *FrontEnd*.

Es kann sein, dass Sie die Datenbank in Frontend und Backend aufteilen, um mit mehreren Benutzern und entsprechend vielen Frontends auf die im Backend befindlichen Daten zugreifen zu können.

Die Frontend-Datenbank enthält dann die Version im Feld *Version_FE* der Tabelle *tblOptionen*, das Backend ein Feld namens *Version_BE* in einer gleichnamigen Tabelle.

Der vorläufige Entwurf dieser Tabelle sieht wie in Abbildung 1.17 aus.

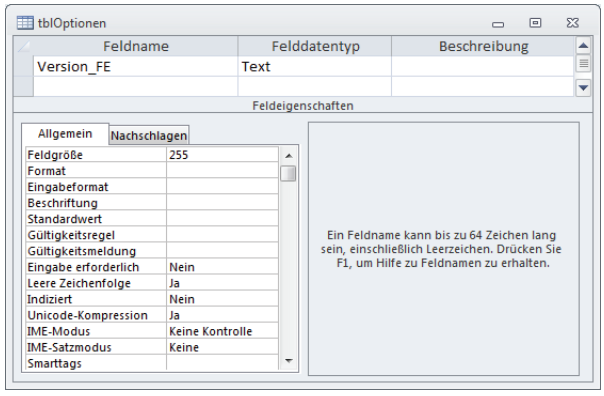


Abbildung 1.17: Die Optionentabelle mit dem Feld zum Speichern der Version

Wie Sie die Versionsnummer vergeben, bleibt Ihnen überlassen. In der Beispielanwendung haben wir eine aus vier Elementen bestehende Versionsnummer verwendet. Zum Start enthält das Feld *Version_FE* der Tabelle also beispielsweise den Wert *0.9.0.0* (siehe Abbildung 1.18).

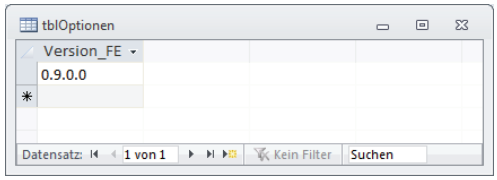


Abbildung 1.18: Speichern der Frontend-Version in der Tabelle *tblOptionen*

1.5 Startformular erstellen

Früher oder später wird Ihre Anwendung beim Start Operationen ausführen, die den Startvorgang verzögern – zum Beispiel das erneute Einbinden der Tabellen des Backends, das Verbinden mit einem SQL-Server oder das Einlesen von Konfigurationsdateien.

Da Sie den Benutzer nicht warten lassen möchten, fügen Sie ein Startformular zur Anwendung hinzu. Dieses wird einfach für ein paar Sekunden angezeigt (wenn die Hintergrundprozesse beim Start zügig vonstatten gehen) oder stellt den Fortschritt in Form eines Fortschrittsbalkens dar.

Solch ein Formular können Sie noch für andere Zwecke einsetzen: Wenn Sie beispielsweise mit einer aufgeteilten Datenbank arbeiten, kann es aus Performancegründen sinnvoll sein, eine Verbindung zum Backend aufrechtzuerhalten.

Dies gelingt am einfachsten, indem Sie ein Formular an eine Tabelle im Backend binden und dieses Formular während der gesamten Sitzung geöffnet halten. Natürlich soll der Benutzer dieses Formular nicht zu Gesicht bekommen, aber das ist kein Problem: Sie können es ja einfach unsichtbar machen.

Für den Beginn erstellen wir ein einfaches Startformular, das ein Logo und eine Schaltfläche zum Schließen beziehungsweise Ausblenden des Formulars enthält.

Das Formular soll *frmStart* heißen. Erstellen Sie es mit dem Ribbon-Befehl *Erstellen|Formulare|Formularentwurf* und ändern Sie seine Größe, beispielsweise auf 5 x 8 cm.

Passen Sie nun einige Eigenschaften an:

» *Rahmenart*: *keine*

» *Automatisch zentrieren*: *Ja*

» *Datensatzmarkierer*: *Nein*

» *Navigationsschaltflächen*: *Nein*

» *Trennlinien*: *Nein*

» *Bildlaufleisten*: *Nein*

Anschließend fügen Sie, soweit vorhanden, ein Logo oder ein Bild ein und legen die benötigten Bezeichnungsfelder an. Das Bild fügen Sie je nach Access-Version ein und zeigen es in einem Bild-Steuerelement an – weitere Informationen hierzu erhalten Sie im Kapitel »Bilder« (Seite 473). Unter Access 2010 sieht der Aufruf des Dialogs zum Auswählen der Bilddatei beispielsweise wie in Abbildung 1.19 aus.

Nachdem Sie ein Bild, einige Bezeichnungsfelder mit den Texten und eine Schaltfläche hinzugefügt haben, sieht das Startformular im Entwurf wie in Abbildung 1.20 aus.

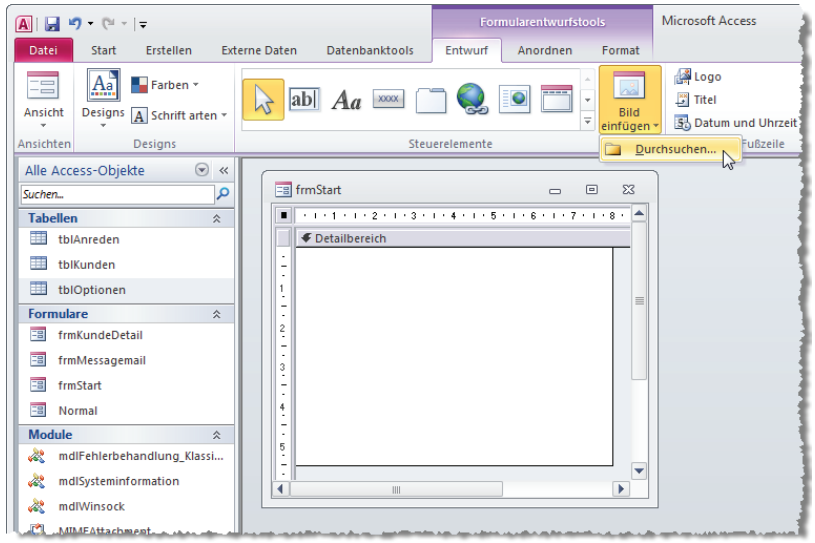


Abbildung 1.19: Einfügen eines Bildes in ein Formular

Abbildung 1.20: Das Startformular in der Entwurfsansicht

Damit die Schaltfläche nicht wie eine herkömmliche Schaltfläche aussieht, sondern sich in das Gesamtbild integriert, stellen Sie auch hier einige Eigenschaften wie *Hintergrundart* (*Transparent*), *Rahmenart*, *Rahmenfarbe*, *Schriftart* et cetera nach Ihrem Geschmack ein.

Da es aktuell noch keine Prozesse gibt, die durch Anzeige des Startformulars überbrückt werden sollen, soll die Anwendung das Startformular einfach beim Öffnen anzeigen und beim Klicken auf die Schaltfläche schließen. Dazu sind zwei Schritte nötig: das Festlegen des Formulars als Startformular und das Bestücken der Schaltfläche mit einer entsprechenden Ereignisprozedur.

Das Startformular legen Sie in den Access-Optionen fest, und zwar im Bereich *Aktuelle Datenbank* unter der Eigenschaft *Formular anzeigen* (siehe Abbildung 1.21).

2 Datenmodell

Am Anfang eines jeden Access-Projekts steht eine Idee. Im vorliegenden Fall soll eine Kunden- und Bestellverwaltung erstellt werden, die dem Benutzer so viel Arbeit wie möglich abnimmt. Das heißt in erster Linie, dass die Anwendung Prozesse abbildet und vereinfacht. Bei diesen Prozessen fallen Daten an, die irgendwo gespeichert werden sollen. Da dieses Buch von der Anwendungsentwicklung mit Microsoft Access handelt, schlagen wir gleich zwei Fliegen mit einer Klappe – Access ist sowohl eine Entwicklungsumgebung für die Benutzeroberfläche als auch ein Datenbankmanagementsystem.

Im einfachsten Fall, also bei einer Umgebung mit einem einzigen Arbeitsplatz, finden Benutzeroberfläche, Anwendungslogik und Daten in einer einzigen Datei Platz. In vielen Fällen arbeitet nicht nur eine Person mit einer solchen Anwendung, was zur Aufteilung in mehrere Dateien führt – in der Regel in ein Frontend, das die Benutzeroberfläche und die Anwendungslogik enthält, und ein Backend mit den in Tabellen gespeicherten Daten. Gegebenenfalls wachsen mit der Zeit die Anforderungen oder die Daten sollen auch über das Internet verfügbar sein – dann verwenden Sie einen SQL-Server wie MySQL oder den Microsoft SQL Server zum Speichern der Daten und greifen über das Access-Frontend auf diese Daten zu.

Unabhängig davon, wo Sie die Daten nun speichern, sollten Sie eine gehörige Portion Hirnschmalz in die Entwicklung des Datenmodells stecken. Das Datenmodell umfasst die Tabellen zum Speichern der Daten, die Beziehungen zwischen den Tabellen sowie einige weitere Kleinigkeiten wie die Definition von Indizes et cetera.

Für die meisten Anwendungsfälle gibt es Standards, was die Definition der Tabellen und der Beziehungen zwischen den Tabellen angeht, aber sehr oft treffen Sie auf verschiedene Möglichkeiten, um einen Sachverhalt in den Tabellen einer Datenbank abzubilden. In diesem Buch gehen wir sowohl auf Standards als auch auf solche Konstellationen ein, wo Sie sich für eine Möglichkeit entscheiden müssen.

2.1 Entwicklung eines Datenmodells

Die Entwicklung des Datenmodells hängt unmittelbar mit den Prozessen zusammen, zu denen Daten gespeichert, geändert oder bereitgestellt werden sollen. Wenn Sie eine Bestellung verarbeiten, nehmen Sie Kundendaten auf, legen eine Bestellung an und fügen die einzelnen Bestellpositionen samt weiteren Informationen wie etwa die Menge hinzu.

Allein dieser Vorgang erzeugt eine Menge Daten, die alle in entsprechend verknüpften Tabellen gespeichert werden sollen. Welche Tabellen Sie benötigen und wie diese verknüpft werden, leiten wir in den folgenden Abschnitten her. Anschließend folgt jeweils gleich der praktische Teil: die Erstellung der Tabellen, Beziehungen und Indizes.

2.2 Kunden erfassen

Kundendaten sind für den Geschäftserfolg ungemein wichtig. Sie nutzen diese nicht nur, um dem Kunden die bestellten Artikel und gegebenenfalls Dokumente wie eine Rechnung zukommen zu lassen, sondern auch für weitere Aktionen. Dazu gehören erstens solche, die unmittelbar mit der Bestellung zusammenhängen und beispielsweise das Management von Retouren, Kündigungen, Stornierungen, Rechnungen, Zahlungen, Mahnungen, Reklamationen et cetera umfassen.

Zweitens sind Kundendaten sehr interessant für weitere Maßnahmen wie etwa den Versand von Produktinformationen. Also sollten Sie die notwendige Sorgfalt beim Aufnehmen und Verwalten der Kundendaten walten lassen.

Liefer- und Rechnungsanschrift

Bei den Kunden einer Bestellverwaltung müssen Sie abwägen, welche Kundendaten gespeichert werden sollen. Bei Amazon beispielsweise können Sie gleich mehrere Adressen angeben – eine Rechnungsadresse und beliebig viele Versandadressen. Amazon ist jedoch ein Händler, zu dem die Kunden eine jahrelange Beziehung aufbauen, weil Amazon schlicht und einfach fast alles anbietet, was das Herz begehrt, und darüberhinaus auch noch günstig ist. Ich selbst habe schon einige Produkte als Geschenk für Freunde bestellt. Dafür ist natürlich ein Datenmodell nötig, das mehr als eine Lieferadresse berücksichtigt.

Wir wollen jedoch eine einfachere Bestellverwaltung schaffen. Für diesen Fall sollte es standardmäßig ausreichen, wenn Sie die Adressdaten je Kunde auf maximal eine Liefer- und eine Rechnungsadresse beschränken. Für Lieferungen an alternative Adressen können wir immer noch eine weitere Tabelle vorsehen.

Bleibt noch die Frage, wie diese maximal zwei Adressen gespeichert werden sollen. Es gibt die folgenden Möglichkeiten:

» Sie speichern eine Standardadresse und, sofern diese abweicht, noch eine Rechnungsanschrift.

» Sie speichern eine Standardadresse und, sofern diese abweicht, noch eine Lieferanschrift.

» Sie speichern Liefer- und Rechnungsanschrift explizit. Wenn Liefer- und Rechnungsanschrift gleich sind, werden dennoch beide Anschriften gespeichert.

Alle Möglichkeiten haben Vor- und Nachteile. Wenn Sie eine der ersten beiden Varianten wählen, müssen Sie jeweils prüfen, ob für einen Verwendungszweck wie den Versand eines Paketes oder der Rechnung jeweils eine eigene Adresse angegeben wurde oder ob eine Adresse für beide Fälle eingesetzt werden soll. Wenn Sie hingegen Liefer- und Rechnungsanschrift explizit speichern, kann es zu Problemen kommen, wenn der Benutzer die gleiche Anschrift für Lieferung und Rechnung angibt: Sie speichern dann nämlich redundante Daten und müssen sicherstellen, dass bei Änderung der einen Adresse auch die andere angepasst wird.

Ansprechpartner

In einer wirklich umfassenden Kundenverwaltung müsste man zu einem Kunden beziehungsweise einer Firma einen oder mehrere Ansprechpartner verwalten. Dies ist für die vorliegende Lösung nicht vorgesehen: Sie können zu jedem Kunden Firma und/oder Vor- und Nachname speichern.

Kundenbezeichnung

Im späteren Verlauf werden wir in Formularen per Kombinations- oder Listenfeld auf Kundendaten zugreifen – beispielsweise, um einen Kunden aus einer Übersichtsliste auszuwählen. Zur Identifizierung könnten wir die Kundennummer, die Firma, Vor- und Nachname oder weitere Daten wie etwa die PLZ heranziehen. Gerade bezüglich Firma und Vor- und Nachname werden die Kundendaten jedoch kein einheitliches Bild abgeben: Manche Kunden geben beides an, manche nur die Firma, andere nur den Namen. Wie also stellt man einen Ausdruck zusammen, der zur Auswahl eines Kunden angezeigt werden kann? Dies ließe sich beispielsweise mit einer entsprechenden Abfrage lösen. Alternativ verwenden Sie ein eigenes Feld namens Kundenbezeichnung, das beim Anlegen des Kunden dynamisch aus den vorliegenden Daten zusammengestellt wird und manuell angepasst werden kann. Solch ein Ausdruck könnte beispielsweise die Form *<Firma>, <Vorname> <Nachname> (<Kundennummer>)* aufweisen.

Bestellungen aus dem Ausland

Kunden aus dem europäischen Ausland wünschen unter Umständen eine Rechnung ohne Mehrwertsteuer. Voraussetzung dafür ist, dass diese Ihnen ihre USt.-IdNr. mitteilen. Diese speichern Sie der Einfachheit halber gleich mit den Kundendaten in der entsprechenden Tabelle.

Kommunikationsdaten

Heutzutage ist es nicht mehr damit getan, pro Kunde eine Telefon- und eine Telefaxnummer zu speichern. Es kommt mindestens eine mobile Telefonnummer und eine E-Mail-Adresse hinzu. Wenn Sie viel Zeit und Lust haben, können Sie Kontaktdaten in eigene Tabellen auslagern und sogar speichern, wann ein Kunde unter welcher Rufnummer erreichbar ist. Für eine Anwendung, die Kundendaten in Zusammenhang mit einfachen Bestellungen speichert, reichen die vier Felder Telefon, Telefax, Mobil und E-Mail aus. Sollte dennoch einmal Bedarf an weiteren Kommunikationsdaten bestehen, können Sie diese in einem Bemerkungen-Feld speichern.

Adresszusätze

Nicht immer kommt ein Kunde mit den Feldern Firma, Vorname, Nachname, Straße, PLZ, Ort und Land aus, um seine Anschrift anzugeben. Manchmal soll etwa eine Abteilung statt eines Namens angegeben werden, oder es gibt ein Postfach statt einer Straße. Das Postfach lässt sich dann einfach in das Feld *Strasse* eingeben, aber für eine Abteilung ist eigentlich ein zusätzliches Feld notwendig. Alternativ können Sie eine mehrzeilige Eingabe der Firma erlauben. Ein

weiteres Feld ist jedoch für Kunden wie auch für Benutzer besser greifbar, also fügen wir den Adressfeldern noch ein Feld namens *Adresszusatz* hinzu.

2.2.1 Die Tabelle tblAnreden

Die Tabelle *tblAnreden* enthält nur drei Felder: das Feld *AnredeID* als Primärschlüsselfeld, das Feld *Anrede* mit der Bezeichnung der Anrede sowie eine Floskel, die wir später als Anrede in einem Brief verwenden (zum Beispiel *Sehr geehrter Herr ...*). Die Tabelle legen Sie mit dem Ribbon-Eintrag *Erstellen|Tabellen|Tabellenentwurf* an (siehe Abbildung 2.1).

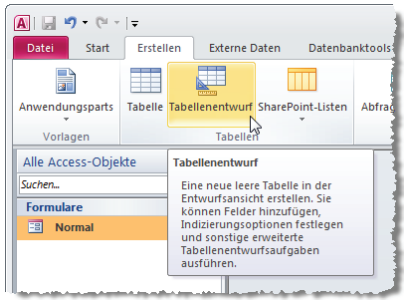


Abbildung 2.1: Anlegen einer Tabelle per Ribbon-Befehl

In der nun erscheinenden Entwurfsansicht fügen Sie den Namen des ersten Feldes ein und wählen den Datentyp aus. Das Feld heißt *AnredeID* und erhält als Felddatentyp den Wert *Autowert*. Außerdem klicken Sie, während das Feld markiert ist, auf die Schaltfläche *Entwurf|Tools|Primärschlüssel* des Ribbons (siehe Abbildung 2.2).

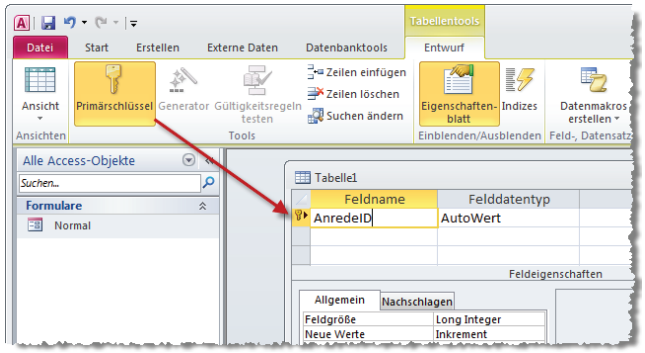


Abbildung 2.2: Anlegen eines Primärschlüsselfeldes

Das zweite Feld heißt *Anrede* und soll den Datentyp *Text* aufweisen. Diese Informationen tragen Sie in die zweite Zeile des Entwurfsrasters ein (siehe Abbildung 2.3). Der Wert *255* für die Eigenschaft *Feldgröße* gibt die maximale Textlänge an. Wir würden hier mit weniger auskom-

men, allerdings wird durch einen größeren Wert auch kein unnötiger Speicherplatz verschwendet. Also belassen wir es bei diesem Wert.

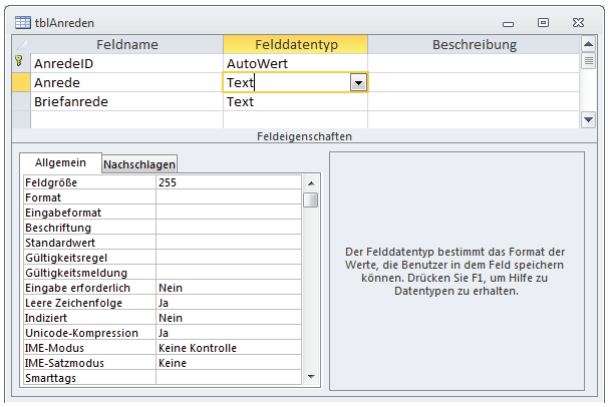


Abbildung 2.3: Entwurf der Tabelle *tblAnreden*

Speichern Sie die Tabelle anschließend unter dem Namen *tblAnreden*. Dies erledigen Sie bei aktivierter Tabelle beispielsweise mit der Tastenkombination *Strg + S* und anschließender Eingabe des Namens der Tabelle, hier also *tblAnreden*.

Primärschlüsselfelder

Jede Tabelle benötigt ein Primärschlüsselfeld. Das Primärschlüsselfeld enthält keinen Wert mehr als einmal und dient daher zur eindeutigen Identifizierung eines jeden Datensatzes einer Datenbank. Dies ist zwingend erforderlich, wenn Sie diese Datensätze den Datensätzen anderer Tabellen zuweisen möchten.

Die Anrede *Herr* erhält beispielsweise im Primärschlüsselfeld *AnredeID* den Wert *1*, *Frau* erhält den Wert *2*. So können Sie später in der Kundentabelle den Wert *1* oder *2* statt *Herr* oder *Frau* angeben und sind sicher, dass immer der richtige Wert der Tabelle *tblAnreden* zugeordnet wird. In der Regel versehen Sie ein Primärschlüsselfeld mit dem Access-Datentyp *Autowert*. Dadurch werden die Werte automatisch vergeben.

Tabelle tblAnreden füllen

Normalerweise sollten Sie die Daten von Tabellen nicht in der Datenblattansicht bearbeiten – auf gar keinen Fall aber dürfen die Benutzer der Datenbank dies tun! Eine Tabelle wie *tblAnreden*, die von Anfang an nur eine kleine Menge vordefinierter Daten aufnehmen soll, können Sie ruhig mal direkt füllen. Dazu wechseln Sie von der Entwurfsansicht mit dem Ribbon-Eintrag *Entwurf|Ansichten|Ansicht|Datenblattansicht* in die Datenblattansicht. Schneller geht das nur mit der Tastenkombination *Strg + Punkt (.)*. Zurück zur Entwurfsansicht gelangen Sie mit *Strg + Komma (,)*.

Tragen Sie zunächst die beiden Werte *Herr* und *Frau* ein, und zwar möglichst in dieser Reihenfolge. Access ergänzt das Feld *AnredeID* mit den beiden Werten *1* und *2* (siehe Abbildung 2.4). Im Feld *Briefanrede* tragen Sie die komplette erste Zeile eines Anschreibens ein. Natürlich kennen Sie den Namen des Adressaten noch nicht, aber das ist kein Problem: Wir legen einfach einen Platzhalter mit dem Feld, aus dem die Daten später geholt werden sollen, in eckigen Klammern an.

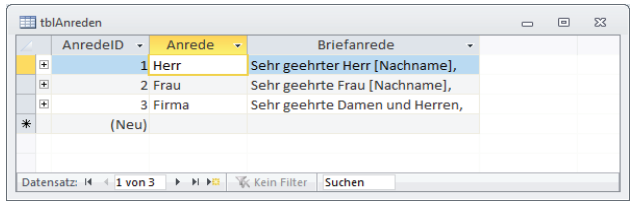


Abbildung 2.4: Hinzufügen der ersten Werte für die Tabelle *tblAnreden*

2.2.2 Die Tabelle tblKunden

Die Kundentabelle enthält die Liefer- und/oder Rechnungsdaten des Kunden sowie einige weitere Informationen. Genau wie der Tabelle *tblAnreden* fügen Sie auch *tblKunden* wieder ein Primärschlüsselfeld hinzu, das diesmal *KundeID* heißt (siehe Abbildung 2.5). Danach folgen die Adressdaten für die Rechnungsanschrift. Alle Felder haben gemeinsam, dass sie mit dem Präfix *Rechnung_* beginnen. Dies grenzt die Felder von den Feldern für die Lieferanschrift ab, die alle das Präfix *Liefer_* aufweisen.

Die Rechnungsanschrift enthält folgende Felder: *Rechnung_Firma*, *Rechnung_AnredeID*, *Rechnung_Vorname*, *Rechnung_Nachname*, *Rechnung_Strasse*, *Rechnung_PLZ*, *Rechnung_Ort* und *Rechnung_Land*. Alle Felder bis auf das Feld *Rechnung_AnredeID* sind als Textfelder mit der Feldgröße *255* ausgelegt. Das Feld *Rechnung_AnredeID* ist ein Fremdschlüsselfeld, über das die Beziehung zur Tabelle *tblAnreden* hergestellt werden soll. Auf das Feld *Rechnung_Anschrift* kommen wir später zu sprechen.

Beziehung zur Tabelle tblAnreden

Das Feld *Rechnung_AnredeID* soll einen der Zahlenwerte aus dem Feld *AnredeID* der Tabelle *tblAnreden* als Wert aufnehmen. Dadurch weisen Sie einer Rechnungsadresse eine der in der Tabelle *tblAnreden* gespeicherten Anreden zu. Wenn Sie später die Felder der Tabelle aus der Feldliste in ein Formular etwa zur Verwaltung der Kundendaten ziehen, soll das Steuerelement, welches das Feld *Rechnung_AnredeID* als Steuerelementinhalt verwendet, gleich als Kombinationsfeld ausgelegt werden. Um dies zu erreichen, legen Sie das Feld gleich als Nachschlagefeld mit den Datensätzen der Tabelle *tblAnreden* aus.

Dazu wählen Sie aus der Liste der Felddatentypen den Eintrag *Nachschlage-Assistent...* aus (siehe Abbildung 2.6).

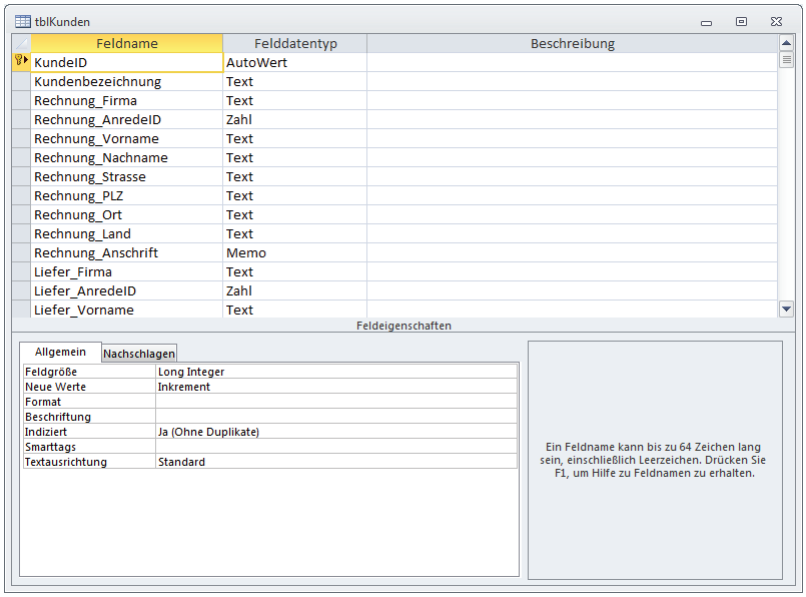


Abbildung 2.5: Entwurf der Tabelle *tblKunden*

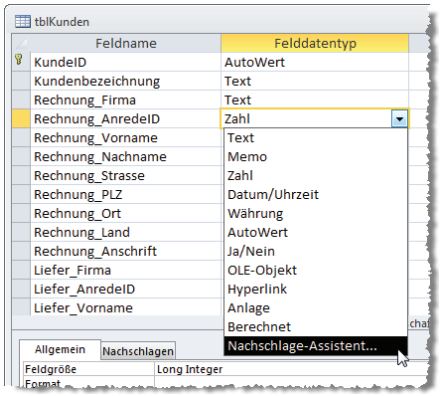


Abbildung 2.6: Erstellen eines Nachschlagefeldes

Im ersten Schritt des nun erscheinenden Assistenten legen Sie fest, dass die Werte aus einer Tabelle oder Abfrage stammen sollen. Legen Sie dann die Tabelle *tblAnreden* als Datenherkunft für das Nachschlagefeld fest und wählen Sie danach die beiden Felder *AnredeID* und *Anrede* als Spalten des Nachschlagefeldes aus. Die Festlegung einer Sortierung ist in diesem Fall nicht nötig – die Datensätze sollen schlicht in der Sortierung nach dem Primärschlüsselwert angezeigt werden. Schließlich teilen Sie dem Assistenten mit, dass die Schlüsselspalte ausgeblendet werden soll, und aktivieren, sofern Sie mit Access 2010 arbeiten, die Datenintegrität – unter Access

2007 ist noch ein zusätzlicher Schritt nötig, da der Nachschlage-Assistent diese Einstellung hier noch nicht vorsieht (siehe Abbildung 2.7).

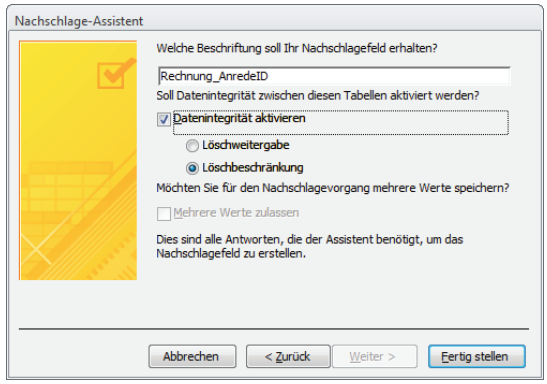


Abbildung 2.7: Letzter Schritt des Nachschlage-Assistenten

Was haben Sie nun erreicht? Der offensichtliche Effekt zeigt sich beim Wechsel in die Datenblattansicht der Tabelle *tblKunden* (Sie erinnern sich? Am schnellsten geht dies mit *Strg + Punkt(.)*). Dort öffnet ein Klick auf das Feld *Rechnung_AnredeID* ein Nachschlagefeld mit den beiden Einträgen der Tabelle *tblAnreden* (siehe Abbildung 2.8).

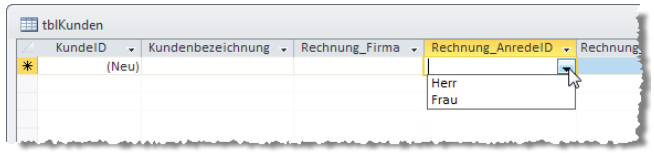


Abbildung 2.8: Erfolgreich angelegtes Nachschlagefeld

Wenn Sie dort einen Wert auswählen, trägt Access nun nicht etwa den Ausdruck *Herr* oder *Frau* in das Feld *Rechnung_AnredeID* der Tabelle *tblKunden* ein, sondern den entsprechenden Zahlenwert aus der Tabelle *tblAnreden*.

Welche Einstellungen Access hier vorgenommen hat, offenbart ein Blick auf die Registerkarte *Nachschlagen* der Eigenschaften des Feldes *Rechnung_AnredeID* (siehe Abbildung 2.9).

Das Feld soll kein Textfeld anzeigen, wie es ein Feld standardmäßig tut, sondern ein Kombinationsfeld. Dieses verwendet eine Tabelle oder Abfrage als Datensatzherkunft – in diesem Fall die Abfrage *SELECT [tblAnreden].[AnredeID], [tblAnreden].[Anrede] FROM tblAnreden;*. Warum nun zeigt das Nachschlagefeld die Anrede an, speichert aber nach der Auswahl den Wert des Feldes *AnredeID* des entsprechenden Datensatzes? Dafür sorgen die folgenden Eigenschaften: Zunächst wird die erste Spalte als gebundene Spalte festgelegt. Das bedeutet, dass der Wert der ersten Spalte der Datensatzherkunft im Feld *Rechnung_AnredeID* gespeichert wird.

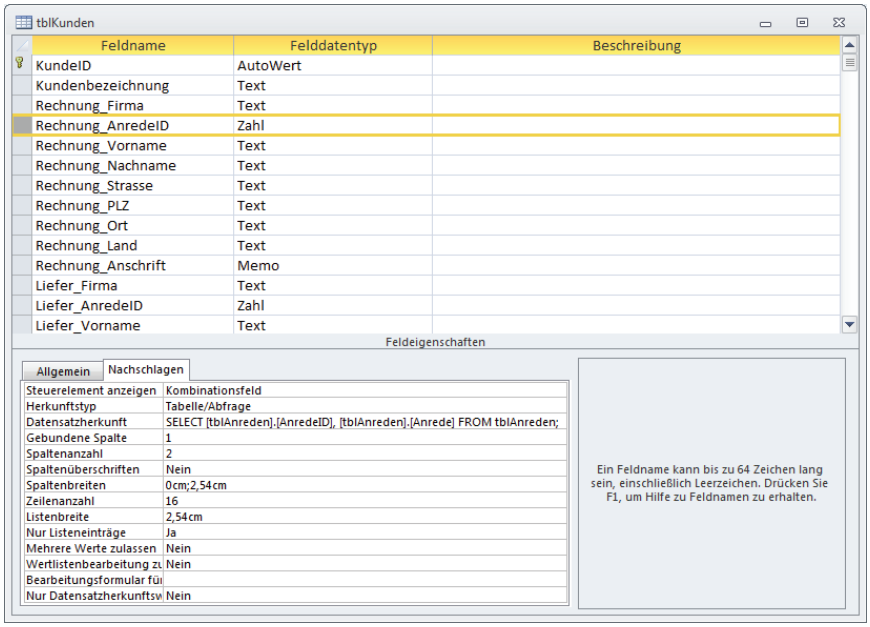


Abbildung 2.9: Eigenschaften eines Nachschlagefeldes

Das Feld *AnredeID* ist also nun offensichtlich vorhanden, warum wird es dann nicht angezeigt? Das liegt an der Einstellung der beiden Eigenschaften *Spaltenanzahl* und *Spaltenbreiten*. Die Eigenschaft *Spaltenanzahl* enthält den Wert *2*, also zeigt das Nachschlagefeld auch den Inhalt der beiden Felder der als Datensatzherkunft angegebenen Abfrage an.

Die Eigenschaft *Spaltenbreiten* legt jedoch mit dem Wert *0cm;2,54cm* die Breite der ersten Spalte auf *0cm* fest, was gleichbedeutend mit dem Ausblenden dieser Spalte ist.

Das Anlegen des Nachschlagefeldes hat jedoch noch weitreichendere Auswirkungen. Access hat nämlich auch noch eine Beziehung zwischen den Tabellen *tblKunden* und *tblAnreden* angelegt – und zwar eine 1:n-Beziehung.

Das heißt, dass Sie jedem Datensatz der ersten Tabelle genau einen Datensatz der zweiten Tabelle zuweisen können (die Tabelle, deren Primärschlüsselfeld an der Beziehung beteiligt ist, heißt übrigens Mastertabelle, die Tabelle mit dem Fremdschlüsselfeld Detailtabelle).

Diese finden Sie im Beziehungen-Fenster vor, das Sie über den Ribbon-Eintrag *Datenbanktools|Beziehungen|Beziehungen* öffnen. Das Beziehungen-Fenster zeigt die beiden Tabellen *tblKunden* und *tblAnreden* an und verbindet die beiden Felder *Rechnung_AnredeID* und *AnredeID* durch einen Beziehungspfeil.

Dieser besagt, dass Sie für das Feld *Rechnung_AnredeID* jeden Wert des Feldes *AnredeID* der Tabelle *tblAnreden* auswählen können (siehe Abbildung 2.10).

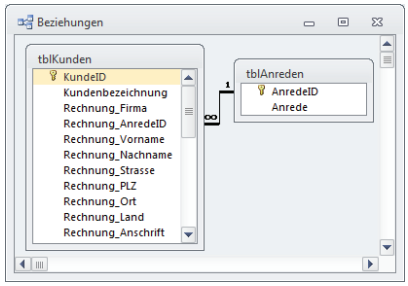


Abbildung 2.10: Beziehung zwischen den Tabellen *tblKunden* und *tblAnreden*

Schließlich haben Sie – zumindest unter Access 2010 – durch die Einstellung im letzten Schritt des Assistenten noch die Datenintegrität sichergestellt. Unter Access 2007 müssen Sie dies noch nachholen. Die entsprechende Eigenschaft finden Sie, wenn Sie doppelt auf den Beziehungspfeil klicken. Es erscheint der Dialog aus Abbildung 2.11, der weitere Eigenschaften der Beziehung anzeigt. Unter anderem hat Access hier in der Version 2010 bereits die Option *Mit referentieller Integrität* aktiviert – mit Access 2007 fügen Sie hier nun den entsprechenden Haken hinzu. Dadurch ist sichergestellt, dass Sie keine anderen Werte als die im Feld *AnredeID* der Tabelle *tblAnreden* enthaltenen Werte in das Feld *Rechnung_AnredeID* eingeben können.

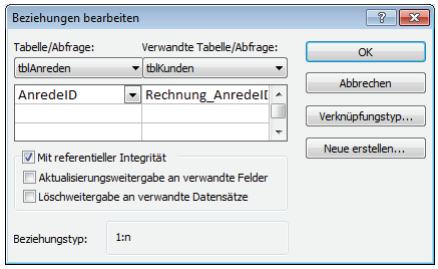


Abbildung 2.11: Bearbeiten der Beziehung

All diese Einstellungen können Sie auch von Hand vornehmen – der Nachschlage-Assistent nimmt einem diese Aufgabe jedoch gern ab.

Damit Sie erkennen, welche Arbeit der Nachschlage-Assistent Ihnen erspart, legen wir für das Feld *Liefer_AnredeID* noch kein Nachschlagefeld ein. Stattdessen definieren Sie dort nur die Beziehung. Öffnen Sie dazu das Beziehungen-Fenster (*Datenbanktools|Beziehungen|Beziehungen*). Klicken Sie mit der rechten Maustaste in das Beziehungen-Fenster und wählen Sie den Kontextmenü-Eintrag *Tabelle anzeigen* aus. Im nun erscheinenden Dialog *Tabelle anzeigen* klicken Sie doppelt auf die Tabelle *tblAnreden*, um diese ein weiteres Mal zum Beziehungen-Fenster hinzuzufügen – dieses wird dann *tblAnreden_1* benannt, weil der Name *tblAnreden* bereits vergeben ist. Ziehen Sie danach das Feld *AnredeID* der Tabelle *tblAnreden_1* auf das Feld *Liefer_AnredeID* der Tabelle *tblKunden*, um die Beziehung zwischen den beiden Tabellen

herzustellen. Klicken Sie dann doppelt auf den Beziehungspfeil und aktivieren Sie im Dialog *Beziehungen bearbeiten* die Option *Mit referentieller Integrität* (siehe Abbildung 2.12).

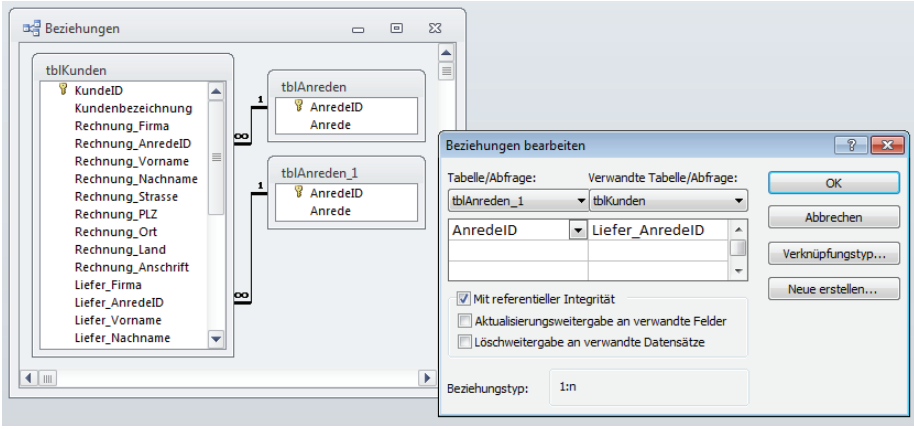


Abbildung 2.12: Herstellen einer weiteren Beziehung zwischen den Tabelle *tblKunden* und *tblAnreden*

Damit haben Sie nun zumindest sichergestellt, dass der Benutzer nur die Werte *1* oder *2* in das Feld *Liefer_AnredeID* eingeben kann.

Wie sich das Anlegen (und das Auslassen) des Nachschlagefeldes bei der späteren Entwicklung auswirkt, erfahren Sie etwa unter »Kombinationsfeld aus Nachschlagefeld« (Seite 90).

Beschriftungen definieren

Der Tabellentwurf bietet eine Eigenschaft namens *Beschriftung*. Damit Sie später weniger Arbeit beim Erstellen der Formulare haben, können Sie gleich an Ort und Stelle Beschriftungen für die Tabellenfelder festlegen.

Der Hintergrund ist, dass Sie Felder direkt aus der Feldliste per Drag and Drop zu einem Formular hinzufügen können. Dabei wird dann ein Textfeld angelegt, dem Access standardmäßig ein zusätzliches Bezeichnungsfeld mit dem Feldnamen als Beschriftung hinzufügt.

Eine Beschriftung wie *Rechnung_Firma* macht sich allerdings nicht besonders gut, daher werden Sie diese wahrscheinlich einfach durch den Ausdruck *Firma* ersetzen. Noch einfacher gelingt dies, wenn Sie die gewünschte Bezeichnung gleich für die Eigenschaft *Beschriftung* des Tabellenfeldes eintragen (siehe Abbildung 2.13). Access übernimmt diese Beschriftung dann beim Hinzufügen eines Textfeldes auf Basis dieses Tabellenfeldes in den Formularentwurf.

Auf die gleiche Weise fügen Sie auch den übrigen Feldern eine entsprechende Beschriftung hinzu. Sonderfälle sind etwa die Felder *Rechnung_Strasse* und *Liefer_Strasse*, wo nicht nur das Präfix wegfällt, sondern auch noch das doppelte *s* durch *ß* ersetzt wird, das Feld *EMail* (wird zu *E-Mail*) und das Feld *UStIdNr* (*Ust.-IdNr.*).

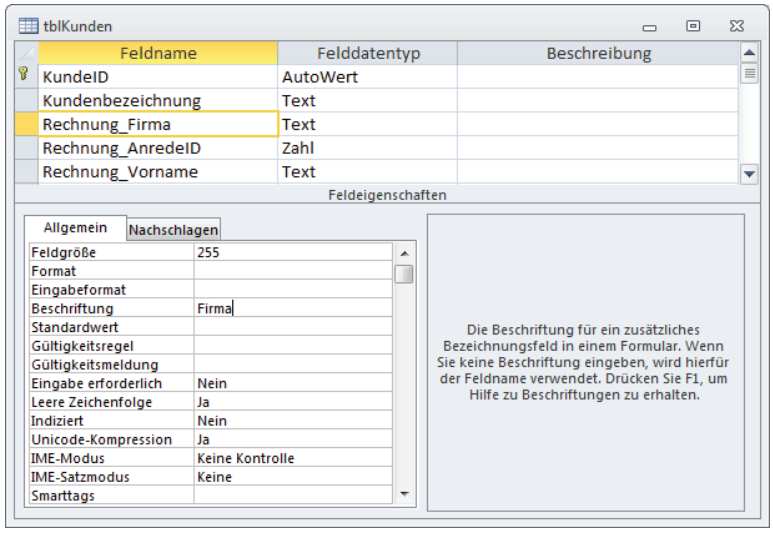


Abbildung 2.13: Festlegen einer Beschriftung für ein Tabellenfeld

2.3 Artikel verwalten

Während die Variationsmöglichkeiten beim Erstellen von Tabellen zum Speichern der Kundendaten bereits sehr umfangreich sind, gilt das Gleiche auch für die Verwaltung von Artikeln. Die einfachste Version einer Artikeltabelle dürfte sich mit Feldern wie *ArtikelID*, *Artikelname*, *Einzelpreis* und *Mehrwertsteuersatz* zufriedengeben. Hinzu kommt dann gegebenenfalls eine Artikelkategorie, die ihre Kategorien aus einer einfachen oder verschachtelten Kategorien-Struktur bezieht.

Für die Lagerhaltung sind Zahlen wie der aktuelle Bestand, der Bestand, bei dem nachgeordert werden soll, und gegebenenfalls ein Feld zur Kennzeichnung von Auslaufartikeln wichtig. Und wie viele Artikel sollen eigentlich nachbestellt werden, wenn der Vorrat zur Neige geht? Wird überhaupt auf Vorrat bestellt oder nur nach Eingang einer Bestellung? Letzteres darf sich beispielsweise ein Onlinehändler heutzutage wohl nur erlauben, wenn er ein Produkt exklusiv vertreibt.

Interessant ist außerdem der Verlauf der Preisgestaltung: Je nach Produktart ändern sich die Einkaufs- und Verkaufspreise von Produkten mehr oder weniger oft. Und wer weiß, wie sich die Mehrwertsteuersätze demnächst gestalten – bleibt es bei 7% und 19%? Für einfache Artikelverwaltungen können Sie sich den Aufwand zum Mitschreiben einer Preis- und MwSt.-Historie jedoch sparen: Der einzige, der sich für einen Preis zu einem bestimmten Zeitraum interessieren dürfte, ist wahrscheinlich der Kunde – und die Preise für bereits verkaufte Artikel finden sich ja in den Tabellen zum Speichern von Bestelldaten.

Das nächste interessante Thema sind Rabatte: Bei welcher Menge gibt es welchen Rabatt, wie wird dieser für Stammkunden nochmals angepasst?

Sie sehen: Es gibt hier eine Menge Dinge, die wir besprechen könnten. Ich fürchte nur, dass erstens eine Behandlung aller Varianten den Rahmen dieses Buches sprengen würde und das zweitens jede tiefer gehende Behandlung eines der zuvor genannten Aspekte nur für eine kleine Teilmenge der Leser interessant wäre. Also halten wir die Artikelverwaltung so einfach wie möglich.

Wie bereits weiter oben erwähnt, wollen wir die Artikeltabelle relativ einfach halten – also gerade so, dass die enthaltenen Daten als Grundlage für die Erfassung von Bestellungen und die Erstellung von Rechnungen ausreichen. In diesem Fall sollen die folgenden Informationen in der Tabelle gespeichert werden:

» Artikelnummer

» Artikelname

» Warengruppe

» Einzelpreis

» Mehrwertsteuersatz

» Beschreibung

» Einheit

Gleich bei der Artikelnummer wird es interessant: Ist es okay, wenn das Primärschlüsselfeld als Artikelnummer herhält? Oder benötigen Sie eine individuelle Artikelnummer? Wenn das Primärschlüsselfeld die Artikelnummer liefert, soll dieses dennoch als Autowert definiert werden oder möchten Sie die Werte selbst vergeben?

Die Entscheidung hängt wohl davon ab, ob Sie die Datenbank neu aufbauen oder ob Sie Artikel mit entsprechenden Artikelnummern aus einer bestehenden Anwendung übernehmen möchten. In der vorliegenden Tabelle halten wir es einfach: Das Feld mit der Artikelnummer soll gleichzeitig als Primärschlüsselfeld und als Autowertfeld ausgelegt sein.

Die Warengruppe soll eine Kategorisierung der Artikel ermöglichen, die einzelnen Warengruppen landen in einer Lookup-Tabelle – dazu später mehr.

Interessant wird es beim Mehrwertsteuersatz. Aktuell gibt es einen Standardsatz von 19% sowie einen ermäßigten Satz von 7% etwa für Lebensmittel. Von Zeit zu Zeit werden diese Steuersätze jedoch erstens grundsätzlich geändert und zweitens kann es sein, dass sich der Steuersatz für bestimmte Artikel ändert. Welche Möglichkeiten gibt es da zum Speichern des Mehrwertsteuersatzes? Die einfachste Variante wäre es, den Steuersatz einfach als Zahlenfeld in der Tabelle zu speichern. Das ist aber recht unkomfortabel, da man den Steuersatz jeweils manuell eintippen müsste. Ein Nachschlagefeld wäre schon schick. Nun gibt es mehrere

Möglichkeiten: Zunächst können Sie eine Wertliste anlegen, die innerhalb des Tabellenentwurfs gespeichert wird und die beiden Mehrwertsteuersätze enthält. Alternativ erstellen Sie eine kleine Tabelle mit den Steuersätzen und verknüpfen die Artikeltabelle über ein Nachschlagefeld mit dieser Tabelle. Letzteres würde den Vorteil mit sich bringen, dass der Benutzer der Anwendung über ein geeignetes Formular weitere Steuersätze angeben oder die bestehenden Steuersätze anpassen kann. Wenn Sie die Steuersätze im Tabellenentwurf in Form einer Wertliste anlegen, müssen Sie für eine Änderung der bestehenden Mehrwertsteuersätze den Tabellenentwurf ändern.

Also entscheiden wir uns, die Mehrwertsteuer in einer Lookup-Tabelle zu speichern und diese von der Artikeltabelle aus zu verknüpfen. Eine weitere Alternative soll zumindest noch erwähnt werden: Dabei legen Sie den Mehrwertsteuersatz nicht für die Artikel selbst, sondern für die Warengruppe fest, der die Artikel angehören. Dann würden Sie die Festlegung der Mehrwertsteuer zweistufig durchführen, indem Sie erst die Mehrwertsteuer für die Warengruppe festlegen und diese dann über die Auswahl der Warengruppe dem Artikel zuweisen. Schließlich sollen zu den Artikeln Einheiten angegeben werden können, wobei der Benutzer diese flexibel festlegen kann (Stück, Stunde, Kilo).

2.3.1 Die Tabelle tblArtikel

Die Artikeltabelle enthält ein Primärschlüsselfeld namens *ArtikelID*, das als Autowert definiert wird und gleichzeitig als Artikelnummer dient. Die Felder *WarengruppeID*, *MehrwertsteuersatzID* und *EinheitID* sind noch einfache Felder vom Typ *Zahl*, sollen aber gleich zu Nachschlagefeldern werden – allein die Tabellen, in denen nachgeschlagen werden soll, fehlen noch. Das Feld Einzelpreis erhält den Datentyp *Währung* (siehe Abbildung 2.14). Damit später beim Hinzufügen der Felder zu Formularen und Berichten gleich wieder passende Beschriftungsfelder erstellt werden (also etwa *Warengruppe* statt *WarengruppeID* oder *MwSt.-Satz* statt *Mehrwertsteuersatz*), stellen Sie die Eigenschaft *Beschriftung* für die betroffenen Felder gleich auf die gewünschten Texte ein.

2.3.2 Die Tabelle tblMehrwertsteuersaetze

Diese Tabelle enthält lediglich zwei Felder und dient als Nachschlage- oder Lookup-Tabelle für das Feld *MehrwertsteuersatzID* der Tabelle *tblArtikel*. Das Feld *MehrwertsteuersatzID* der Tabelle *tblMehrwertsteuersätze* dient als Primärschlüsselfeld. Das Feld *Mehrwertsteuersatz* soll den Zahlenwert aufnehmen, der dem Mehrwertsteuersatz entspricht. Hier wird es interessant: Gelegentlich findet man in Beispieldatenbanken solche Felder, die den Zahlenwert 7 oder 19 enthalten, die um das Prozentzeichen ergänzt werden. Das ist beim Berechnen der Mehrwertsteuerbeträge unpraktisch, weil man alles noch durch 100 teilen muss. In unserer Anwendung soll das Feld *Mehrwertsteuersatz* daher die Werte *0,07* und *0,19* aufnehmen. Dann kann man den Betrag einfach mit dem Mehrwertsteuersatz multiplizieren, um den Mehrwertsteuerbetrag zu erhalten.

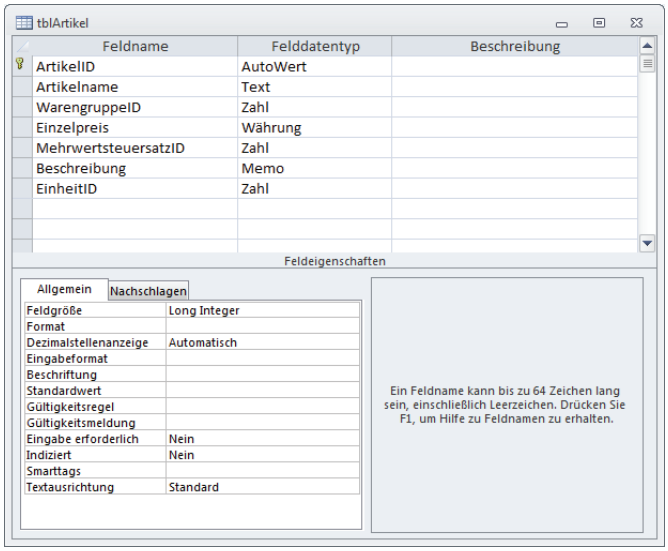


Abbildung 2.14: Entwurf der Tabelle *tblArtikel*

Nun kommt es noch darauf an, welchen Datentyp Sie für die Prozentzahl verwenden. Gleitkommazahlen liefern mitunter Rundungsfehler, weshalb gerade für Zahlen, die mit Währungsbeträgen multipliziert werden, ein Festkommazahlen-Datentyp verwendet werden sollte. Access macht es uns an dieser Stelle einfach: Wir verwenden einfach den Datentyp *Währung* und stellen als Format die Eigenschaft *Prozentzahl* ein (siehe Abbildung 2.15).

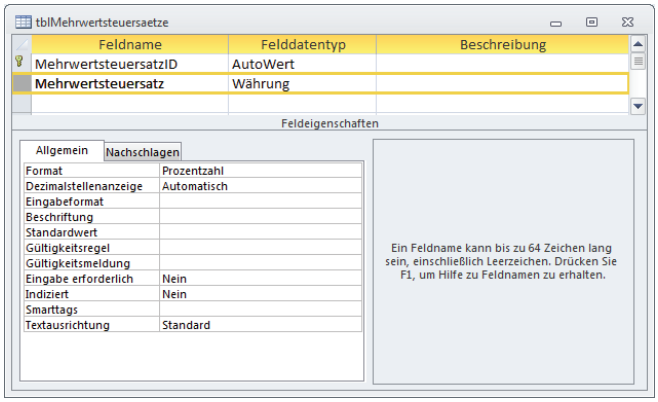


Abbildung 2.15: Entwurf der Tabelle *tblMehrwertsteuersaetze*

Das Interessante dabei ist, wie Access nun Eingaben in dieses Feld verarbeitet. Dazu wechseln Sie in die Datenblattansicht der Tabelle und geben die beiden Zahlenwerte *7* und *19* ein. Die Tabelle zeigt die Werte wie erwartet als *7%* und *19%* an (siehe Abbildung 2.16).

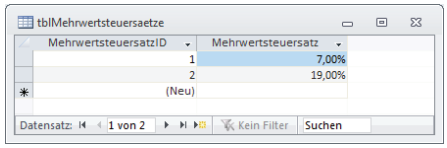


Abbildung 2.16: Eingabe von Prozentzahlen in eine Tabelle

Aber welche Werte sind nun in der Tabelle gespeichert? Dies bekommen Sie leicht heraus, indem Sie eine Abfrage auf Basis der Tabelle *tblMehrwertsteuersaetze* erstellen und die Eigenschaft *Format* für das Feld *Mehrwertsteuersatz* auf *Allgemeine Zahl* einstellen (siehe Abbildung 2.17).

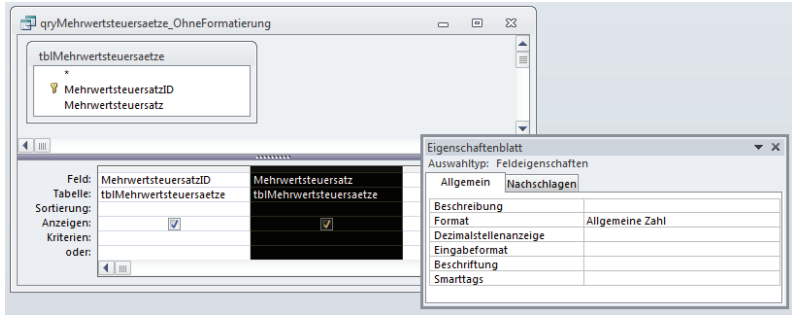


Abbildung 2.17: Diese Abfrage soll die wahren Werte hinter den Prozentzahlen sichtbar machen.

Das Ergebnis ist erfreulich: Hinter den Prozentzahlen verbergen sich in Wirklichkeit Festkommazahlen, welche die Prozentzahlen in Hundertsteln ausdrücken (siehe Abbildung 2.18).

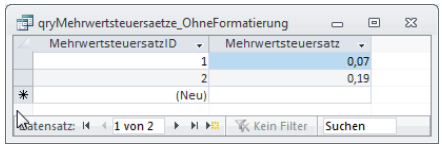


Abbildung 2.18: Die wahren Werte hinter Prozentzahlen

Nachdem die Tabelle fertiggestellt ist, können Sie auch das Zahlenfeld *MehrwertsteuersatzID* der Tabelle *tblArtikel* in ein Nachschlagefeld umwandeln. Dazu schließen Sie zunächst die Tabelle *tblMehrwertsteuersaetze* und öffnen dann die Tabelle *tblArtikel* in der Entwurfsansicht.

Weiter oben haben Sie am Beispiel des Feldes *AnredeID* der Tabelle *tblKunden* bereits erfahren, wie Sie eine Beziehung samt Nachschlagefeld per Assistent erstellen, nun exerzieren wir das einmal in Handarbeit durch. Klicken Sie im Entwurf der Tabelle *tblArtikel* auf das Feld *MehrwertsteuersatzID* und zeigen Sie die Registerseite *Nachschlagen* im Bereich *Feldeigenschaften* an. Dort wählen Sie zunächst den Wert *Kombinationsfeld* für die Eigenschaft *Steuerelement anzeigen* aus. Als Datensatzherkunft legen Sie den folgenden Ausdruck fest:

```
SELECT MehrwertsteuersatzID, Mehrwertsteuersatz FROM tblMehrwertsteuersaetze
ORDER BY Mehrwertsteuersatz;
```

Diese Abfrage liefert die Felder *MehrwertsteuersatzID* und *Mehrwertsteuersatz* für alle Datensätze der Tabelle *tblMehrwertsteuersaetze* zurück, und zwar in alphabetischer Reihenfolge für die Werte des Feldes *Mehrwertsteuersatz*. Sie brauchen diese Abfrage nicht manuell einzugeben, sondern können auch auf die Schaltfläche mit den drei Punkten rechts klicken und die nun erscheinende leere Abfrage mit der Tabelle *tblMehrwertsteuersaetze* füllen und das Entwurfsraster wie in Abbildung 2.19 ausstatten.

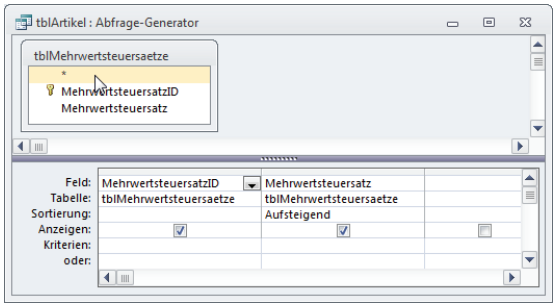


Abbildung 2.19: Datensatzherkunft für ein Nachschlagefeld in der Entwurfsansicht

Die Einstellung der beiden Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *2* und *0cm* sorgt dafür, dass zwar beide Felder im Nachschlagefeld angezeigt werden, das erste aber mit der Spaltenbreite *0cm* und somit quasi unsichtbar ist. Die zweite Spalte nimmt, da keine explizite Breite angegeben wurde, den Rest der Breite des Nachschlagefeldes ein.

Das waren schon alle Einstellungen, die Sie im Tabellenentwurf vornehmen müssen. Der Entwurf sieht nun wie in Abbildung 2.20 aus.

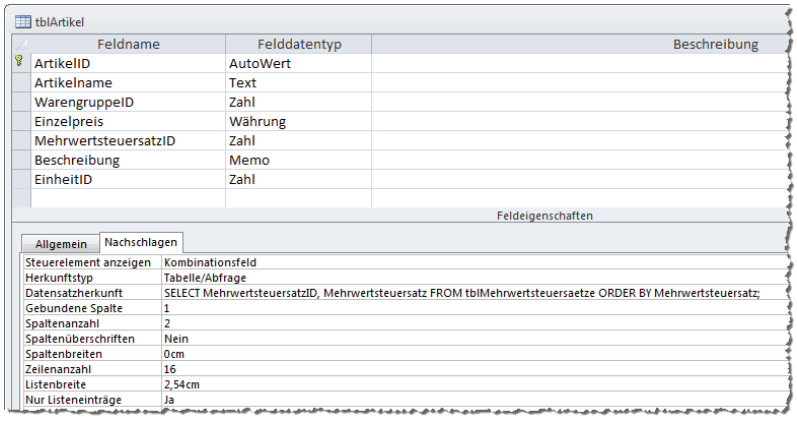


Abbildung 2.20: Manuelles Anlegen eines Nachschlagefeldes

Nun fehlt noch das Anlegen der Beziehung. Schließen Sie die Tabelle *tblArtikel* und öffnen Sie mit dem Ribbon-Befehl *Datenbanktools|Beziehungen|Beziehungen* das Beziehungsfenster.

Betätigen Sie die rechte Maustaste und rufen Sie den Eintrag *Tabelle anzeigen...* des Kontextmenüs auf. Fügen Sie damit die beiden Tabellen *tblArtikel* und *tblMehrwertsteuersaetze* zum Beziehungsfenster hinzu.

Ziehen Sie dann das Feld *MehrwertsteuersatzID* von der Tabelle *tblMehrwertsteuersaetze* auf das gleichnamige Feld der Tabelle *tblArtikel*. Es erscheint automatisch der Dialog *Beziehungseigenschaften*, mit dem Sie nur die oberste Option namens *Mit referentieller Integrität* aktivieren (siehe Abbildung 2.21).

Die Eigenschaft *Löschweitergabe an verwandte Datensätze* dürfen Sie auf gar keinen Fall aktivieren: Dies würde bedeuten, dass beim Versuch, einen der Mehrwertsteuersätze zu löschen, auch alle Artikel gelöscht werden, denen dieser Mehrwertsteuersatz zugewiesen wurde. Wenn Sie diese Option nicht aktivieren, führt der Versuch, einen bereits verwendeten Mehrwertsteuersatz zu löschen, zu einer Fehlermeldung.

Die Eigenschaft *Aktualisierungsweitergabe an verwandte Felder* bedeutet, dass beim Ändern der Primärschlüsselwerte auch die damit verknüpften Fremdschlüsselwerte aktualisiert werden. Dies sollte allerdings nötig sein.

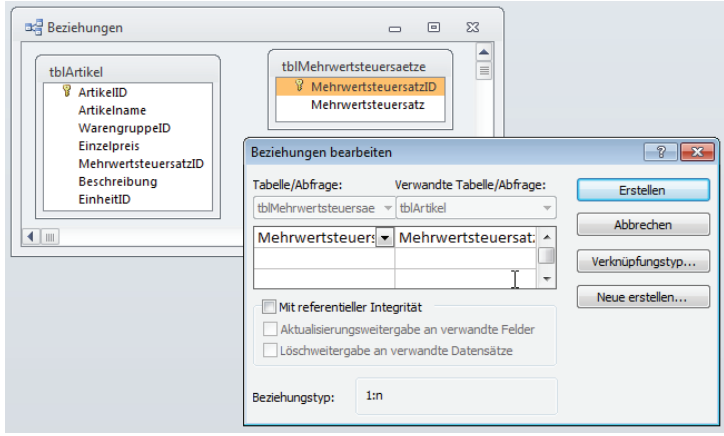


Abbildung 2.21: Einstellen der Eigenschaften einer frisch erstellten Beziehung

Nach dem Schließen der Beziehungs-Eigenschaften zeigt das Beziehungsfenster die neue Beziehung an, und zwar mit den Symbolen, die eine 1:n-Beziehung mit referentieller Integrität markieren (siehe Abbildung 2.22).

Fehlt nur noch die Probe aufs Exempel. Also schließen Sie den *Beziehungen*-Dialog und öffnen die Tabelle *tblArtikel* in der Datenblattansicht. Und die Auswahl des Mehrwertsteuersatzes funktioniert, wie Abbildung 2.23 zeigt.

3 Kunden verwalten

Die Verwaltung von Kunden ist einer der häufigsten Anwendungsfälle in Access-Datenbanken. In diesem Kapitel erfahren Sie, wie Sie die Daten der Tabelle *tblKunden* verwalten. Da es das erste Kapitel dieses Buchs ist, das die Erstellung von Formularen beschreibt, finden Sie hier einige grundlegende Aspekte; in den übrigen Kapiteln etwa zu den Themen *Artikelverwaltung* oder *Bestellverwaltung* gehen wir dann etwas mehr ans Eingemachte.

Am Ende des vorliegenden Kapitels werden Sie ein Übersichtsformular mit einem Unterformular zur Anzeige aller Kunden erstellt haben. Von diesem aus können Sie neue Kunden anlegen und vorhandene Kunden bearbeiten oder löschen. Das Anlegen und Bearbeiten erfolgt in einem Detailformular, das bereits einige technische Feinheiten enthalten wird.

3.1 Detailformular erstellen

Um überhaupt manuell Kunden anlegen oder bearbeiten zu können, legen Sie ein Formular an, das alle relevanten Kundendaten anzeigt. Erstellen Sie über den Ribbon-Eintrag *Erstellen|Formulare|Entwurfsansicht* ein neues, leeres Formular. Damit das Formular die Daten der Tabelle *tblKunden* anzeigt, fügen Sie diese als Wert der Eigenschaft *Datenherkunft* hinzu (siehe Abbildung 3.1).

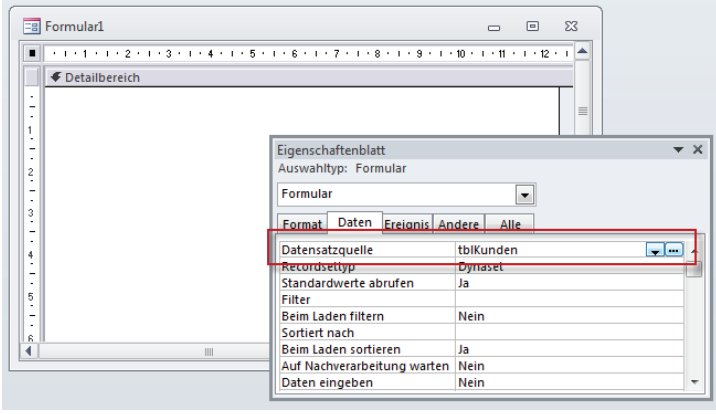


Abbildung 3.1: Festlegen der Datensatzquelle des Formulars

Anschließend finden Sie die Felder der Tabelle *tblKunden* in der Feldliste des Formulars vor. Diese blenden Sie über den Ribbon-Eintrag *Entwurf|Tools|Vorhandene Felder hinzufügen* ein. Die Liste zeigt alle Felder der Datensatzquelle an, in diesem Fall also alle Felder der Tabelle *tblKunden* (siehe Abbildung 3.2).

Abbildung 3.2: Liste der Felder der Datensatzquelle des Formulars

Sie können nun per Drag and Drop die benötigten Felder in den Detailbereich des Formularentwurfs ziehen. Dazu markieren Sie zunächst alle gewünschten Felder und platzieren diese dann an der gewünschten Stelle. Als Erstes legen Sie einen Block mit den Rechnungsinformationen an. Markieren Sie also alle Felder, die mit *Rechnung_* beginnen, und ziehen Sie diese dann en bloc in das Formular.

Kombinationsfeld aus Nachschlagefeld

Das Ergebnis sieht wie in Abbildung 3.3 aus. Wie Sie sehen, hat sich die Vorarbeit beim Entwurf der Tabelle *tblKunden* gelohnt: Die Bezeichnungsfelder werden mit korrekten Beschriftungen ausgestattet (also beispielsweise *Firma:* statt *Rechnung_Firma:*) und das Feld *Rechnung_AnredeID* wird als Kombinationsfeld ausgeführt, weil es bereits im Tabellenentwurf als Nachschlagefeld definiert wurde. Und auch die Schriftgröße wird auf den in der Normal-Formularvorlage vorgegebenen Wert eingestellt.

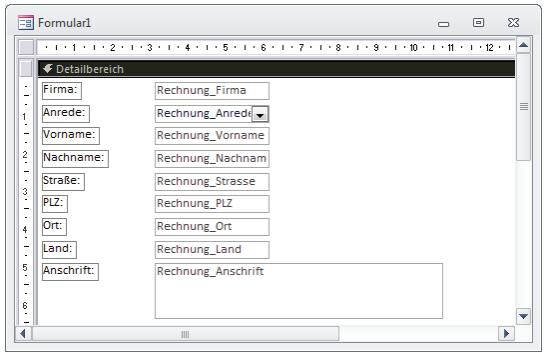


Abbildung 3.3: Frisch eingefügte gebundene Steuerelemente

Auf die gleiche Weise fügen Sie nun die Felder hinzu, deren Name mit *Liefer_* beginnt. Und wie soll man nun die Rechnungs- und die Lieferadresse voneinander unterscheiden? Dazu fügen Sie zwei Rahmen mit entsprechenden Überschriften hinzu. Klicken Sie im Ribbon auf die

Schaltfläche *Entwurf|Steuerelemente|Optionsgruppe* und ziehen Sie damit einen Rahmen in der gewünschten Größe auf (siehe Abbildung 3.4). Markieren Sie auf das Bezeichnungsfeld, das mit dem Rahmen angelegt wird, und klicken Sie erneut darauf, um die Beschriftung in *Rechnungs-/Lieferadresse* zu ändern.

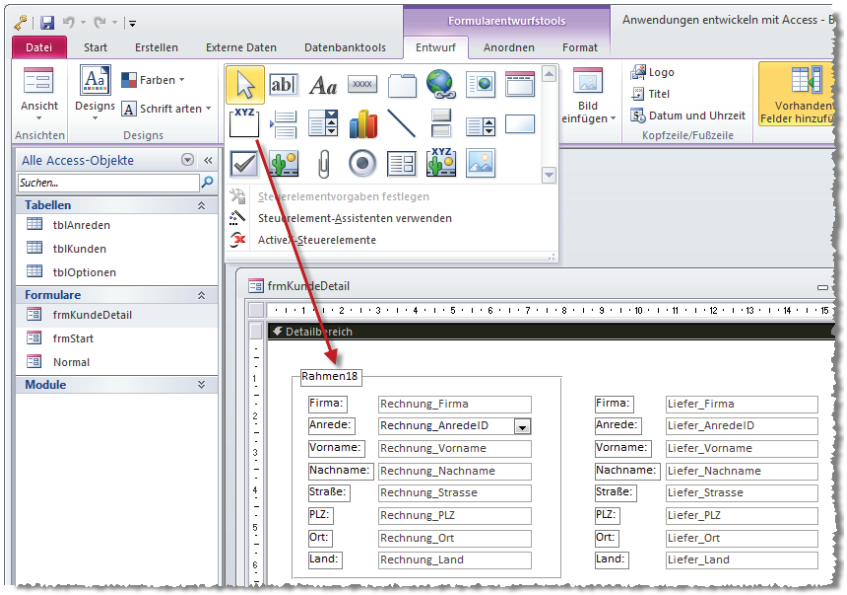


Abbildung 3.4: Rahmen zur optischen Gliederung aufziehen

Fügen Sie danach die Adressfelder für die Lieferanschrift zum Formular hinzu und legen Sie einen weiteren Rahmen an. Das Bezeichnungsfeld dieses Rahmens erhält die Beschriftung *Lieferanschrift*.

3.1.1 Felder anpassen und ausrichten

Nun kümmern wir uns um die Optik der bislang angelegten Steuerelemente. Am besten gelingt dies, wenn die zugrunde liegenden Tabellen bereits einige Datensätze enthalten. Wenn Sie das Buch von vorn gelesen haben, haben Sie die Tabelle *tblKunden* aber sicher bereits mit einigen durch den Beispieldaten-Assistenten generierten Daten gefüllt. Grundsätzlich sollten Sie zunächst einmal die horizontale Ausrichtung vornehmen. Als Erstes erledigen wir dies für die Bezeichnungsfelder der Rechnungsanschrift. Die Bezeichnungsfelder sollen linksbündig am Bezeichnungsfeld des Rahmens ausgerichtet werden. Dazu markieren Sie zunächst alle betroffenen Felder, am einfachsten durch Aufziehen eines vertikalen Rahmens, der alle Bezeichnungsfelder schneidet. Damit haben Sie gegebenfalls auch den Rahmen selbst markiert, der natürlich nicht mit ausgerichtet werden soll. Um den Rahmen aus der Auswahl zu entfernen, halten Sie die Umschalt-Taste gedrückt und klicken gezielt auf den Rahmen.

Danach klicken Sie mit der rechten Maustaste auf eines der markierten Elemente und wählen aus dem Kontextmenü den Eintrag *Ausrichten|Linksbündig* aus (siehe Abbildung 3.5).

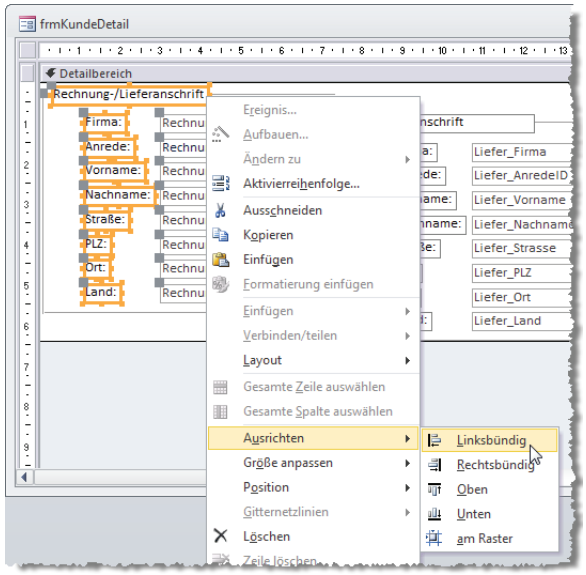


Abbildung 3.5: Ausrichten der Bezeichnungsfelder

Dies richtet die Bezeichnungsfelder linksbündig aus. Nun passen Sie die Breite und Position der Textfelder an. Da Sie alle Textfelder en bloc aus der Feldliste in den Detailbereich gezogen haben, dürften diese zumindest alle den gleichen Abstand zu den Bezeichnungsfeldern haben. Die Textfelder aus Abbildung 3.6 sollen nun erstens näher an die Bezeichnungsfelder heran verschoben und zweitens vergrößert werden.

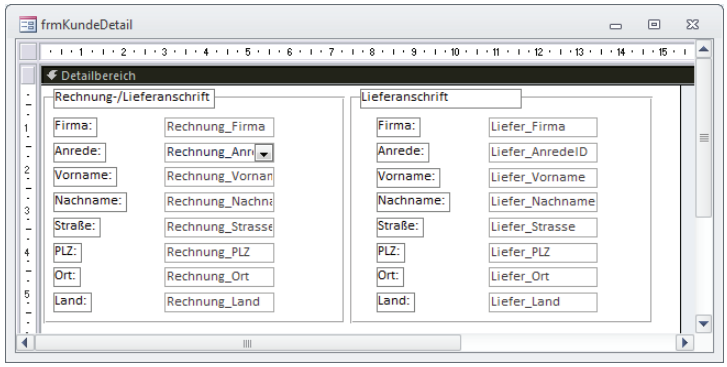


Abbildung 3.6: Die Textfelder sind zu weit rechts und zu schmal.

Dazu führen Sie zwei Schritte durch:

» Markieren Sie alle Textfelder und ziehen Sie den linken Rand eines der Textfelder näher an die Bezeichnungsfelder heran.

» Vergrößern Sie die Textfelder auf die gleiche Weise nach rechts.

Anschließend können Sie einzelne Felder, die für den Inhalt zu groß sind, noch verkleinern. Änderungen an einzelnen Feldern nehmen Sie am besten in der sogenannten Layout-Ansicht vor, die gleichzeitig die enthaltenen Daten anzeigt und das Anpassen bestimmter Eigenschaften der Steuerelemente erlaubt. Die Layout-Ansicht aktivieren Sie entweder über den Ribbon-Eintrag *Entwurf|Ansichten|Ansicht|Layout-Ansicht* oder über den Eintrag *Layoutansicht* des Kontextmenüs der Titelleiste des Formulars (siehe Abbildung 3.7).

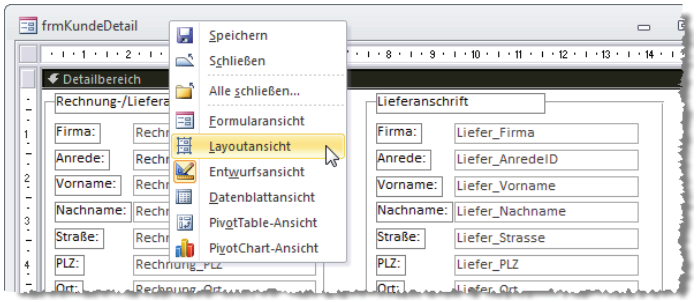


Abbildung 3.7: Aktivieren der Layoutansicht per Kontextmenü

In der Layoutansicht können Sie einzelne Steuerelemente anklicken und deren Position oder Größe einstellen. Änderungen wirken sich dabei jeweils auf das orange markierte Steuerelement aus (siehe Abbildung 3.8). Wenn Sie ein Steuerelement nicht durch Anklicken markieren können, liegt vermutlich ein anderes Steuerelement darüber, weil Sie es später hinzugefügt haben – im vorliegenden Formular könnte das passieren, wenn Sie die Rahmen nach den Textfeldern hinzugefügt haben. Dann gelangen Sie entweder per Tabulator-Taste zum gewünschten Steuerelement oder Sie wechseln zurück in die Entwurfsansicht, markieren das überlappende Steuerelement und verschieben es mit dem Kontextmenü-Eintrag *Position|In den Hintergrund* nach hinten.

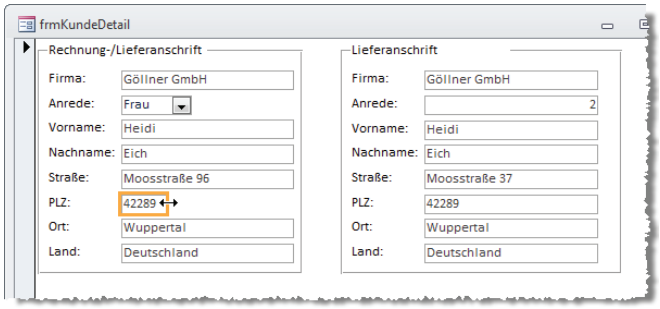


Abbildung 3.8: Anpassen der Steuerelementbreite in der Layoutansicht

3.1.2 Nachschlagefeld nachträglich einrichten

In der Abbildung erkennen Sie übrigens auch, dass das Feld *Liefer_AnredeID* nicht als Kombinationsfeld ausgelegt wurde. Das liegt daran, dass wir dieses Feld beim Definieren der Tabelle *tblKunden* nicht als Nachschlagefeld definiert haben.

Um das als Textfeld angelegte Feld *Liefer_AnredeID* in ein Kombinationsfeld umzuwandeln, wählen Sie zunächst den Eintrag *Ändern zu|Kombinationsfeld* aus dem Kontextmenü des Textfeldes aus (siehe Abbildung 3.9).

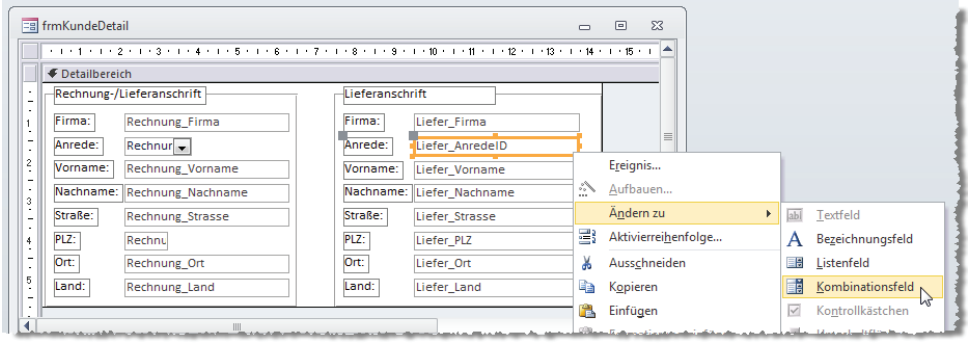


Abbildung 3.9: Umwandeln eines Textfeldes in ein Kombinationsfeld

Danach stellen Sie folgende Eigenschaften ein, damit das Kombinationsfeld die gewünschten Daten anzeigt:

» *Datensatzherkunft*: *SELECT AnredeID, Anrede FROM tblAnreden*

» *Spaltenanzahl*: *2*

» *Spaltenbreiten*: *0cm*

Warum wird die Spaltenbreite für die zweite Spalte hier nicht eingetragen? Weil Access das Fehlen dieser Angabe so interpretiert, dass die zweite Spalte den Rest der Breite des Kombinationsfeldes einnimmt. Fertig – das zweite Kombinationsfeld ist hergerichtet.

3.1.3 Aktivierreihenfolge

Wenn Sie mit der Tabulator-Taste durch die Steuerelemente navigieren, werden Sie feststellen, dass diese möglicherweise nicht in der gewünschten Reihenfolge durchlaufen werden. Diese Reihenfolge können Sie jedoch einstellen.

Dazu wechseln Sie in die Entwurfsansicht des Formulars und klicken mit der rechten Maustaste auf eines der Steuerelemente. Öffnen Sie mit dem Eintrag *Aktivierreihenfolge* des Kontextmenüs den Dialog *Reihenfolge* (siehe Abbildung 3.10). Dieser Dialog bietet zwei Funktionen:

» Mit der Schaltfläche *Automatisch* sortieren Sie die Aktivierreihenfolge der Steuerelemente nach einem bestimmten Schema, was meistens gute Ergebnisse liefert.

» Um einen Eintrag manuell in der Reihenfolge zu verschieben, klicken Sie zunächst auf den zu verschiebenden Eintrag. Dann klicken Sie auf den grauen Bereich links und verschieben den Eintrag per Drag and Drop an die gewünschte Stelle.

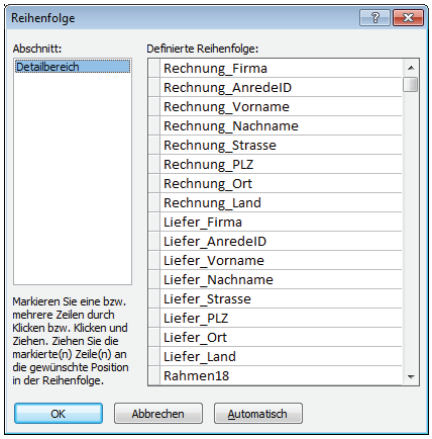


Abbildung 3.10: Mit diesem Dialog stellen Sie die Aktivierreihenfolge der Steuerelemente eines Formulars ein.

Sie können Steuerelemente auch komplett aus der Aktivierreihenfolge herausnehmen. Dazu markieren Sie das Steuerelement in der Entwurfsansicht und wechseln im Eigenschaftsfenster auf die Registerkarte *Andere*. Dort findet sich nicht nur die Eigenschaft *Reihenfolgenposition*, die den numerischen Wert für die Reihenfolgenposition angibt, sondern auch die Eigenschaft *In Reihenfolge* (siehe Abbildung 3.11). Stellen Sie diese Eigenschaft auf den Wert *Nein* ein, damit das Steuerelement beim Durchlaufen mit der Tabulator-Taste nicht mehr berücksichtigt wird.

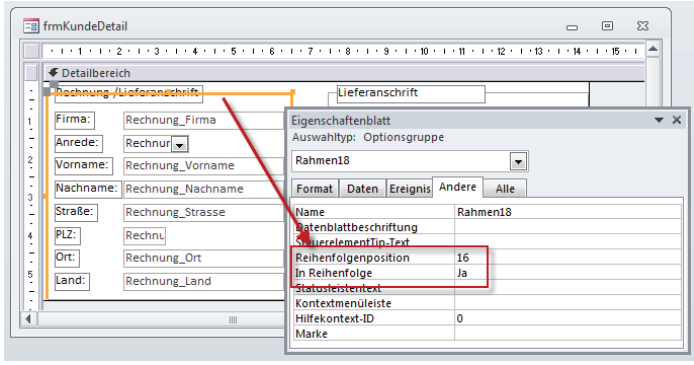


Abbildung 3.11: Einstellen der Aktivierreihenfolge per Eigenschaftsfenster

3.1.4 Steuerelementnamen mit Präfixen versehen

Wenn Sie ein Feld wie etwa *KundeID* aus der Feldliste in den Formularentwurf ziehen, erhält das Steuerelement den gleichen Namen wie das Feld, in diesem Fall also auch *KundeID*. Dies ist in manchen Fällen ungünstig. Wenn Sie etwa den Standardwert eines Feldes im Formular per VBA festlegen möchten, müssen Sie nämlich das Steuerelement referenzieren und nicht das Feld.

Was hilft in diesem Fall? Sie können das Steuerelement umbenennen, sodass Sie es vom gleichnamigen Feld unterscheiden können. Dazu stellen Sie dem Steuerelement ein dem Steuerelementnamen entsprechendes Präfix voran, also beispielsweise *txt* für *Textbox*, *cbo* für *Combobox* oder *chk* für *Checkbox*. Wenn Sie später Schaltflächen zum Formular hinzufügen, würden Sie diese etwa mit dem Präfix *cmd* versehen (zum Beispiel *cmdOK*).

Nun aber kümmern wir uns zunächst darum, dass Sie die gebundenen Steuerelemente von den entsprechenden Feldern unterscheiden können. Auf manuellem Wege würden Sie dazu in den Entwurf des Formulars wechseln, das Steuerelement markieren und die Eigenschaft *Name* des Steuerelements anpassen. Da ich grundsätzlich faul bin, habe ich auch dafür ein Add-In programmiert – den *Control-Renamer*. Informationen zum Download und zur Installation erhalten Sie weiter vorne unter »Control-Renamer« (Seite 25).

Den *Control-Renamer* starten Sie über den Ribbon-Eintrag *Datenbanktools|Add-Ins|Control-renamer*. Das Tool zeigt direkt das erste verfügbare Formular an, das an eine Tabelle oder Abfrage gebunden ist und das noch gebundene Steuerelemente enthält, deren Name mit dem Namen des zugrunde liegenden Feldes übereinstimmt. Abbildung 3.12 zeigt dies beispielsweise für die soeben erstellte Tabelle *frmKundenDetail*. Die linke Spalte liefert die aktuellen Steuerelementnamen, die rechte die Vorschläge des Tools *Control-Renamer*. Klicken Sie links auf *Alle auswählen* und dann auf *Bezeichnungen ändern*, um die vorgeschlagenen Bezeichnungen zu übernehmen.

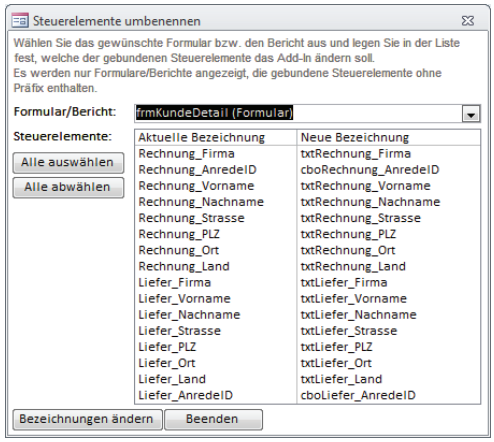


Abbildung 3.12: Der Control-Renamer in Aktion

Diesen Vorgang wiederholen Sie immer, wenn Sie neue gebundene Steuerelemente zu einem Formular hinzugefügt haben, aber spätestens, bevor Sie Ereignisprozeduren für ein solches Steuerelement anlegen. Der Hintergrund ist, dass eine Ereignisprozedur eines Steuerelements immer den Namen des Steuerelements im Prozedurnamen enthält.

Wenn Sie erst eine solche Ereignisprozedur anlegen (zum Beispiel *AnredeID_AfterUpdate*) und dann den Steuerelementnamen ändern, wird die Ereignisprozedur nicht mehr ausgeführt, weil der Name des Steuerelements (nun etwa *cboAnredeID*) nicht mehr mit dem für die Ereignisprozedur angegebenen Namen übereinstimmt.

3.1.5 Weitere Felder hinzufügen

Anschließend fügen Sie weitere Felder hinzu. Ein Rahmen soll die Überschrift *Kommunikation* tragen und die vier Felder *Telefon*, *Telefax*, *E-Mail* und *Mobil* aufnehmen. Schließlich folgt noch ein Rahmen mit der Beschriftung *Rechnungsdaten*, dem Sie die Felder *Kreditinstitut*, *BLZ*, *Kontonummer*, *IBAN*, *BAC* und *UStIDNr* hinzufügen.

Fehlen noch die beiden Felder *KundeID* und *Kundenbezeichnung*. Diese sollen ganz oben im Formular Platz finden, also markieren Sie alle übrigen Elemente und verschieben diese nach unten. Am genauesten gelingt dies, indem Sie die markierten Objekte mit den Cursor-Tasten positionieren. Wenn Sie dies pixelgenau erledigen wollen, halten Sie dabei die *Strg*-Taste gedrückt! Und wo wir gerade bei den Cursor-Tasten sind: Auch die Höhe und Breite von Steuerelementen lässt sich damit leicht anpassen. Eine Anpassung in groben Schritten erfolgt bei gedrückter *Umschalt*-Taste, eine pixelgenaue Anpassung mit *Strg* + *Umschalt* + *Nach oben*/*Nach unten*/*Nach links* und *Nach rechts*.

Wenn Sie nun noch die wichtigsten Felder, also Kundennummer/Kundenbezeichnung und die Überschriften der Kästchen, fett hervorheben (durch einzelnes Markieren der Steuerelemente bei gedrückter Umschalt-Taste), sieht das Formular wie in Abbildung 3.13 aus.

3.1.6 Formulareinstellungen für das Detailformular

Bislang haben wir ja lediglich ein Detailformular zum Bearbeiten von Kunden erstellt. Die Frage ist: Wollen Sie selbst bei einigen hundert Kundendatensätzen immer durch alle Datensätze durchblättern, um die Daten zu einem bestimmten Kunden anzuzeigen? Nein! Die Frage ist nur: Wie ermöglichen wir es, dass der Benutzer der Anwendung schnell die Daten des gesuchten Kunden auffindet? Dazu gibt es mehrere Möglichkeiten:

» Sie fügen diesem Formular ein Unterformular in der Datenblattansicht oder ein Listenfeld hinzu, das alle Kunden anzeigt und die Auswahl eines Kunden etwa per Mausklick auf den entsprechenden Listeneintrag erlaubt.

» Sie fügen dem Detailformular eine Suchfunktion hinzu, welche die Eingabe einzelner Buchstaben erlaubt, um damit in festgelegten Feldern der Kundendaten nach dem gewünschten

Kunden zu suchen. Dies würde im Gegensatz zur Kundenliste nur einen Bruchteil des Platzes in Anspruch nehmen.

» Sie können auch zusätzlich ein Übersichtsformular zur Anzeige einer Kundenliste erstellen, mit dem Sie einen Kunden auswählen und diesen dann im Detailformular anzeigen.

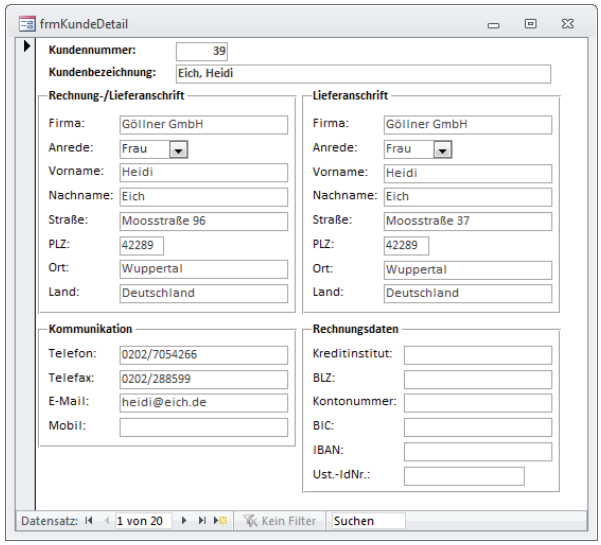


Abbildung 3.13: Kundendetails, Zwischenstand

Fürs Erste werden wir im Anschluss ein Übersichtsformular erstellen, das alle Kunden anzeigt und das Öffnen des Formulars *frmKundeDetail* mit einem bestimmten Kunden erlaubt. Das bedeutet, dass wir im Formular *frmKundeDetail* nicht mehr durch die Kundendatensätze navigieren müssen.

Also nehmen Sie einige Einstellungen vor, die das Formular von einigen unnötigen Steuerelementen befreien. Dazu stellen Sie die folgenden Eigenschaften auf die nachfolgend angegebenen Werte ein:

» *Navigationsschaltflächen*: *Nein*

» *Datensatzmarkierer*: *Nein*

» *Bildlaufleiste*: *Nein*

» *Trennlinien*: *Nein* (dürfte bereits auf diesen Wert eingestellt sein)

Außerdem soll das Formular beim Öffnen jeweils zentriert eingeblendet werden, was Sie durch die Einstellung der Eigenschaft *Automatisch zentrieren* auf *Ja* erreichen (siehe Abbildung 3.14). Schließlich erhält die Eigenschaft *Beschriftung* des Formulars noch den Wert *Kunde - Detailansicht* – auf diese Weise erscheint in der Titelleiste nicht der Name des Formulars.

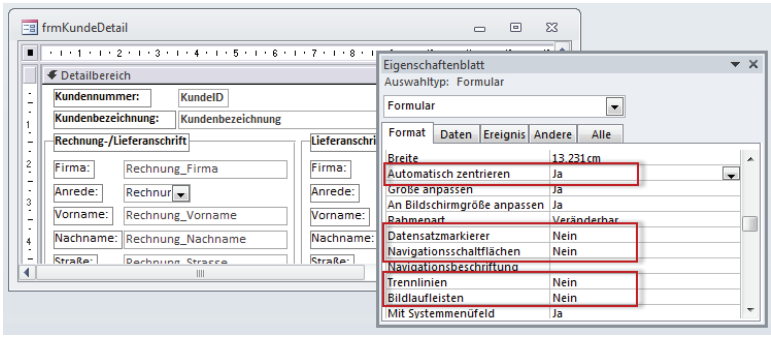


Abbildung 3.14: Einstellen einiger Formulareigenschaften für ein Detailformular

Das Ergebnis liefert bereits ein wesentlich aufgeräumteres Bild als zuvor (siehe Abbildung 3.15).

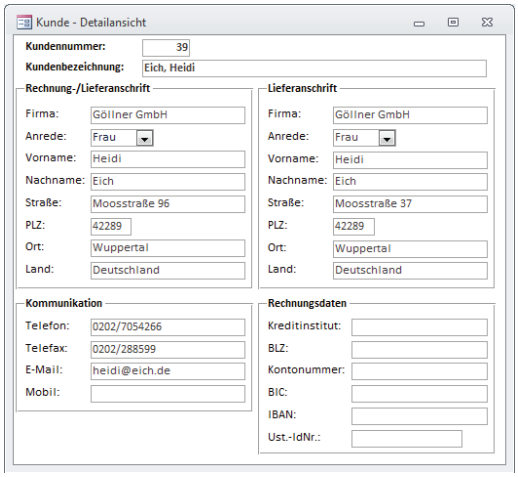


Abbildung 3.15: Das Formular ohne unnötigen Ballast

Damit verabschieden wir uns vorerst vom Kunden-Detailformular und wenden uns dem Übersichtsformular zu. Später kommen wir natürlich noch das eine oder andere Mal zu diesem Kundenformular zurück, denn es wird die Schaltzentrale für alle Aktionen rund um den Kunden werden – hier werden Bestellungen, Kommunikation und mehr verwaltet.

3.2 Übersichtsformular erstellen

Wenn Sie von einem Objekt wie beispielsweise einem Kunden mehrere Exemplare in einer Datenbank verwalten und sich nicht alle Daten dieses Objekts in einer Datenblattansicht erfassen lassen, erstellen Sie – wie oben beschrieben – ein entsprechendes Detailformular. Damit

entledigen Sie sich natürlich nicht der Aufgabe, ein Übersichtsformular zur Auswahl der Kunden zu erstellen. Dies erledigen wir in den folgenden Abschnitten.

Die grundlegende Überlegung dabei ist nicht, wie die Kunden dargestellt werden – dies geschieht natürlich in Listenform. Sie müssen sich vielmehr darüber Gedanken machen, welche Anforderungen Sie an diese Kundenliste stellen. Soll die Liste die Kunden nur anzeigen und etwa durch einen Doppelklick einen Kunden im Detailformular anzeigen? Wollen Sie grundlegende Kundendaten gleich in der Übersicht bearbeiten können? Wie sieht es mit dem Filtern und Sortieren der Daten in der Übersicht aus – soll dies auch möglich sein?

Technisch bieten die Bordmittel von Access folgende Möglichkeiten:

» Listenfeld

» Unterformular in der Datenblattansicht

» Endlosansicht

» ListView-Steuerelement

Das ListView-Steuerelement bietet keine Möglichkeit, eine Datenherkunft wie etwa die Kundentabelle einfach einer Eigenschaft zuzuweisen, daher ist es für den Beginn zu kompliziert. Ein Listenfeld zeigt beliebige Daten an und erlaubt keine direkte Bearbeitung. Der Benutzer kann keinerlei Einstellung vornehmen. Dies ist bei der Datenblattansicht ganz anders: Hier kann der Benutzer die Spalten anordnen und ihre Breite anpassen, die Datensätze sortieren und filtern und sogar Daten bearbeiten. Bei einem Formular in der Endlosansicht kann er Daten sortieren, filtern und bearbeiten.

Bei den genannten Eigenschaften sollten wir außerdem berücksichtigen, dass alle Möglichkeiten natürlich auch eingeschränkt werden können. Sie können also für ein Unterformular in der Datenblattansicht auch festlegen, dass der Benutzer darin keine Daten bearbeiten kann.

In einem Grundlagenbuch würde ich nun alle Methoden einzeln vorstellen und deren Vor- und Nachteile detailliert beschreiben, aber wir wollen hier eine Anwendung entwickeln und arbeiten daher etwas zielgerichteter. Das Übersichtsformular wird mit einem Unterformular in der Datenblattansicht gestaltet, weil es die größte Flexibilität bietet.

3.2.1 Hauptformular und Unterformular

Bei den übrigen drei Lösungen Listenfeld, ListView-Steuerelement und Endlosformular würden wir mit einem einzigen Formular auskommen. Bei Formularen in der Datenblattansicht ist das jedoch etwas anders: Wenn Sie diese Ansicht wählen, sieht das Formular genau wie die Datenblattansicht einer Tabelle aus – zusätzliche Steuerelemente etwa im Kopf oder im Fuß des Formulars werden einfach ausgeblendet. Außerdem können Sie in der Datenblattansicht auch nur drei Steuerelemente einsetzen: das Textfeld, das Kombinationsfeld und das Kontrollkästchen.

Dummerweise kommen wir bei der Kundenübersicht nicht ohne weitere Steuerelemente aus: Wir benötigen zumindest Schaltflächen zum Erstellen von neuen Datensätzen, zum Löschen bestehender Kunden und zum Öffnen eines Datensatzes in der Detailansicht. Nun gibt es zwei Möglichkeiten: Sie könnten tatsächlich mit einem Formular in der Datenblattansicht auskommen, wenn Sie dieses nach dem Öffnen im Access-Fenster maximieren und die Schaltflächen zum Erstellen, Löschen und Öffnen von Kundendatensätzen im Ribbon unterbringen. Dazu ist natürlich auch Interaktion zwischen Ribbon und Formular nötig – im Gegensatz zur Interaktion zwischen einem Formular und einem Unterformular ein recht kompliziertes Verfahren. Also wählen wir die Variante mit einem Hauptformular, das die Steuerelemente zum Erstellen, Anzeigen und Löschen der Kundendaten liefert, und einem Unterformular, dast die Kundenliste in der Datenblattansicht anzeigt.

3.2.2 Unterformular mit Kundenliste erstellen

Erstellen Sie ein neues Formular und speichern Sie es unter dem Namen *sfmKundenuebersicht*. Das Präfix *sfm* steht hier für Subform – im Gegensatz zur Bezeichnung *frm* für das Hauptformular. Weisen Sie der Eigenschaft *Datenherkunft*/*Datensatzquelle* des Formulars zunächst einfach die Tabelle *tblKunden* zu. Ziehen Sie dann alle Felder aus der Feldliste in den Detailbereich des Formulars. Das geht am einfachsten, indem Sie den ersten Eintrag der Liste auswählen und dann bei gedrückter Umschalt-Taste den letzten Eintrag – dies sollte alle Einträge der Liste markieren.

Ziehen Sie dann die markierten Einträge per Drag and Drop in den Detailbereich der Entwurfsansicht des Formulars (siehe Abbildung 3.16).

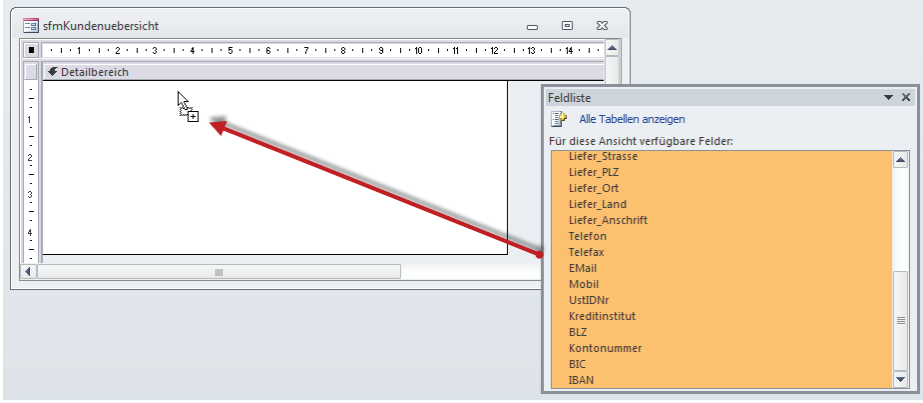


Abbildung 3.16: Hinzufügen aller Felder der Tabelle *tblKunden* zum Formular *sfmKundenuebersicht*

Das Formular enthält dann alle Felder samt Bezeichnungsfeldern. Wenn Sie nun noch die Eigenschaft *Standardansicht* auf *Datenblatt* einstellen, sind die Arbeiten am Unterformular fast beendet (siehe Abbildung 3.17).

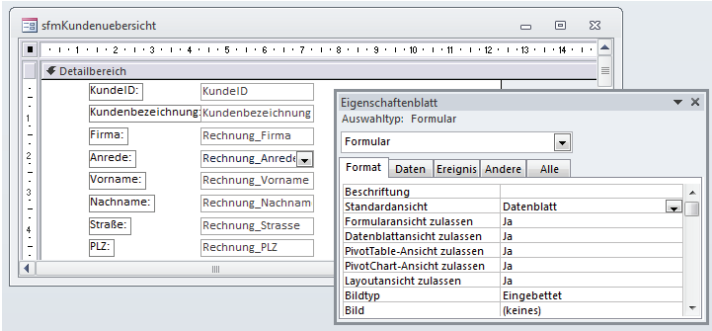


Abbildung 3.17: Finetuning ist bei Formularen in der Datenblattansicht zunächst nicht nötig.

Sie sollten lediglich noch die Namen der hinzugefügten Steuerelemente mit entsprechenden Präfixen ausstatten, also beispielsweise *txtKundeID* statt *KundeID* und *cboAnredeID* statt *AnredeID*. Dabei hilft Ihnen, wie bereits oben erwähnt, der *Control-Renamer* –siehe »Control-Renamer« (Seite 25).

Es lohnt sich ohnehin, dieses Tool während der Entwicklung hin und wieder anzuwerfen – immerhin zeigt es Formulare, die noch gebundene Steuerelemente mit dem Originalnamen enthalten, automatisch an.

Das Einzige, was wir gleich nach einem Wechsel in die Datenblattansicht des Unterformulars erledigen, ist die Einstellung der Schriftgröße. Dazu wechseln Sie zum Ribbon-Tab *Start* und stellen die Schriftart im Bereich *Textformatierung* ein (siehe Abbildung 3.18).

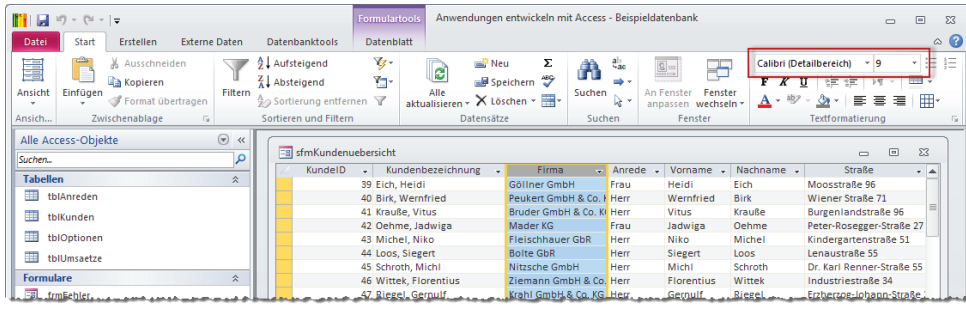


Abbildung 3.18: Einstellen der Schriftart für ein Formular in der Datenblattansicht

Hauptformular erstellen

Das Hauptformular zu diesem Formular soll *frmKundenuebersicht* heißen. Seine Standardansicht können Sie auf *Einzelnes Formular* belassen. Da es selbst keine Daten anzeigt, sondern nur das Formular *sfmKundenuebersicht* als Unterformular aufnehmen soll, können Sie auch hier die Eigenschaften *Navigationsschaltflächen*, *Datensatzmarkierer*, *Bildlaufleisten* und *Trennlinien*

auf den Wert *Nein* einstellen. Die Eigenschaft *Automatisch zentrieren* erhält wiederum den Wert *Ja*, die Eigenschaft *Name* den Wert *Kundenübersicht*. Formularkopf- und Formularfuß können Sie beibehalten.

Ziehen Sie das Formular auf eine entsprechende Größe auf – es soll ja immerhin eine Liste der Kunden anzeigen und dabei möglichst viele Felder liefern. Dann kommt der entscheidende Schritt: Sie integrieren das Unterformular in das Hauptformular.

Unterformular einbauen

Die einfachste Möglichkeit, ein Unterformular zu einem Hauptformular hinzuzufügen, ist folgender:

» Öffnen Sie das Hauptformular in der Entwurfsansicht.

» Ziehen Sie das Unterformular aus dem Navigationsbereich in das Hauptformular.

» Löschen Sie das Bezeichnungsfeld mit der Beschriftung *sfmKundenuebersicht* und ziehen Sie das Unterformular auf die gewünschte Größe. Es darf den gesamten Detailbereich einnehmen (siehe Abbildung 3.19).

Fertig! Damit haben Sie sich ein paar Schritte gespart. Wenn Sie noch nicht in die Programmierung von Unterformularen eingeweiht sind, haben Sie allerdings auch ein paar Schritte ausgelassen, die für das Verständnis notwendig sind.

Daher nun in aller Kürze: Access hat nun für Sie ein Unterformular-Steuerelement zum Formular hinzugefügt. Dieses fügen Sie normalerweise wie die übrigen Steuerelemente über das Ribbon hinzu. Das Unterformular-Steuerelement hat die Aufgabe, Formulare innerhalb anderer Formulare anzuzeigen (seit Access 2010 kann es sogar Berichte anzeigen).

Um dem Unterformular mitzuteilen, welches Objekt es anzeigen soll, stellen Sie die Eigenschaft *Herkunftsobjekt* auf das Formular oder den Bericht ein. Auf die übrigen Eigenschaften, die vor allem für die Programmierung mit VBA wichtig sind, gehen wir später ein.

Wichtig ist an dieser Stelle vor allem die Information, dass das Unterformular-Steuerelement und das Unterformular selbst nicht das Gleiche sind! Das Unterformular-Steuerelement ist schlicht ein Container, das beliebige Formulare in ein anderes Formular integrieren kann. Das Unterformular selbst ist nur das Formular, das innerhalb des Unterformulars angezeigt wird.

Access befeuert die weitverbreitete Idee, dass Unterformular-Steuerelement und Unterformular identisch sind, noch: Es benennt das Unterformular-Steuerelement bei der oben genannten Methode zum Hinzufügen des Unterformulars nämlich genau nach dem hinzugefügten Formular (hier also *sfmKundenuebersicht*).

Um Missverständnisse zu vermeiden, sollten Sie dies nachträglich ändern. Wenn das Formular, wie in diesem Fall, nur ein Unterformular enthält, ändern Sie den Namen des Unterformular-Steuerelements schlicht auf *sfm*. Am einfachsten geht dies, indem Sie im Eigenschaftsfenster

den Eintrag *sfmKundenuebersicht* auswählen und der Eigenschaft *Name* den Wert *sfm* zuweisen (siehe Abbildung 3.20).

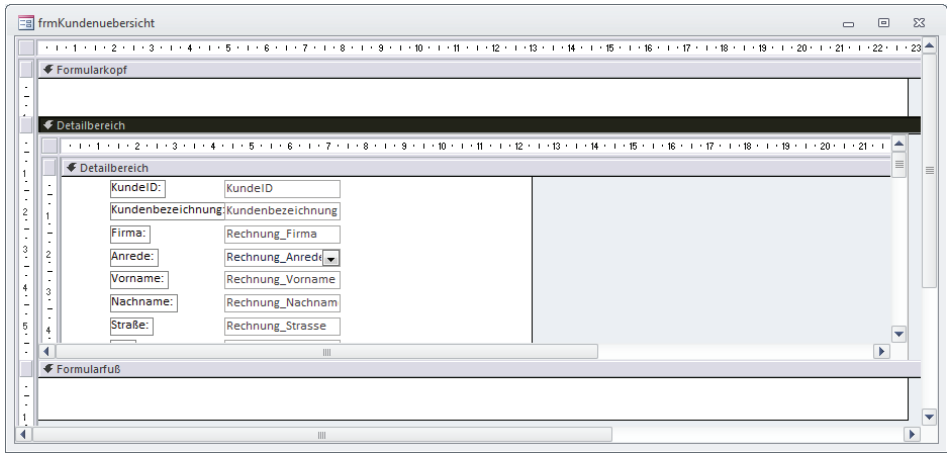


Abbildung 3.19: Das in das Hauptformular *frmKundenuebersicht* integrierte Unterformular *sfmKundenuebersicht*

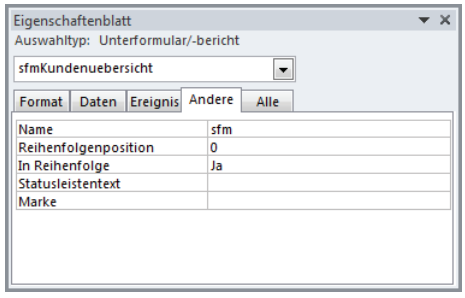


Abbildung 3.20: Ändern des Namens eines Unterformular-Steuerelements

Unterformulargröße dynamisch anpassen

Der Benutzer soll die Größe des Formulars *frmKundenuebersicht* je nach seinen Gegebenheiten anpassen können. Dazu brauchen Sie keine weiteren Einstellungen vorzunehmen. Wichtig wäre es allerdings, dass auch das Unterformular seine Größe entsprechend der Größe des Hauptformulars variiert. Seit Access 2007 ist das kein Problem mehr: Stellen Sie einfach die beiden Eigenschaften *Horizontaler Anker* und *Vertikaler Anker* in den Eigenschaften des Unterformular-Steuerelements auf den Wert *Beide* ein.

Von nun an brauchen Sie sich um die Größenanpassung des Unterformulars keine Sorgen mehr zu machen: Es passt immer komplett in den Detailbereich, egal auf welche Größe der Benutzer das Formular zieht (siehe Abbildung 3.21).

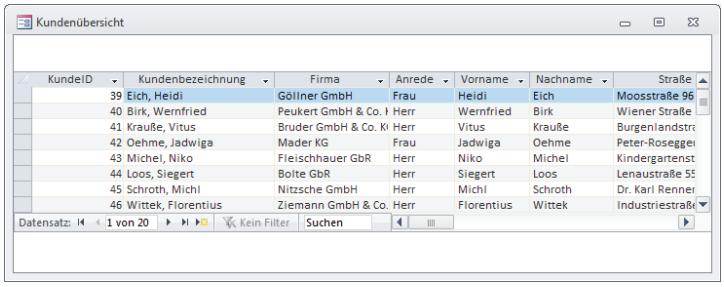


Abbildung 3.21: Das Unterformular ändert seine Größe mit dem Hauptformular.

3.2.3 Kunden anlegen, öffnen und löschen

Nun folgt der interessante Teil: Wir fügen dem Formular Schaltflächen zum Anlegen, Öffnen und Löschen von Kundendaten hinzu. Außerdem erhält das Formular eine Schaltfläche zum Schließen.

Hatte ich erwähnt, dass sich die Anschaffung hübscher Icons für die optische Aufbereitung von Formularen lohnt? Sie können natürlich auch einfach nur Schaltflächen mit Texten versehen, wie wir es nun erstmal tun. Später fügen wir allerdings noch einige Icons hinzu. Das Formular sieht nun im Entwurf wie in Abbildung 3.22 aus.

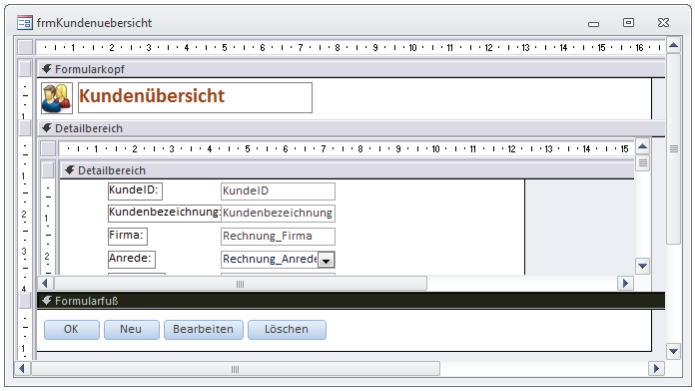


Abbildung 3.22: Die Kundenübersicht in der Entwurfsansicht

Im Formularkopf haben wir noch ein Icon und eine Überschrift hinzugefügt. Wozu diese Überschrift, wenn die Titelzeile des Formulars doch bereits eine Überschrift enthält? Ganz einfach: Weil die Titelleiste beim Maximieren eines Formulars verschwindet. Wie Sie ein solches Icon hinzufügen, erfahren Sie unter »Bilder im Bildsteuerelement (ungebunden)« (Seite 487).

Die Schaltflächen versehen Sie nun noch mit entsprechenden Namen. Dies hat zwei Gründe: Erstens wollen Sie vielleicht später einmal per VBA auf diese Steuerelemente zugreifen, um sie

beispielsweise ein- oder auszublenden oder zu aktivieren oder deaktivieren. Außerdem legen wir gleich VBA-Prozeduren an, die durch einen Mausklick auf die entsprechenden Schaltflächen ausgelöst werden, und diese enthalten den Namen des Steuerelements im Prozedurnamen. Daher sollten Sie aussagekräftige Namen vergeben. Die Schaltflächen erhalten die folgenden Namen:

» *cmdOK*

» *cmdNeu*

» *cmdBearbeiten*

» *cmdLoeschen*

Neben Objekt- und Feldnamen erhalten übrigens auch Formulare und Steuerelemente immer Namen, die ausschließlich aus den Buchstaben *A* bis *Z* und *a* bis *z*, den Zahlen *0* bis *9* und dem Unterstrich (_) bestehen. Weil es am einfachsten ist, legen Sie nun zunächst die Prozedur an, die das Formular beim Anklicken der *OK*-Schaltfläche schließt. Klicken Sie im Entwurf auf die Schaltfläche und wechseln Sie im Eigenschaftsfenster zur Registerseite *Ereignis*.

Klicken Sie dort auf die Eigenschaft *Beim Klicken*, wählen Sie den Wert *[Ereignisprozedur]* aus und klicken Sie dann auf die Schaltfläche mit den drei Punkten (...). Wenn Sie, wie weiter oben beschrieben, die Option *Immer Ereignisprozeduren verwenden* in den Access-Optionen unter *Objekt-Designer|Entwurfsansicht für Formulare/Berichte* eingestellt haben, reicht ein Klick auf die Schaltfläche mit den drei Punkten (siehe Abbildung 3.23).

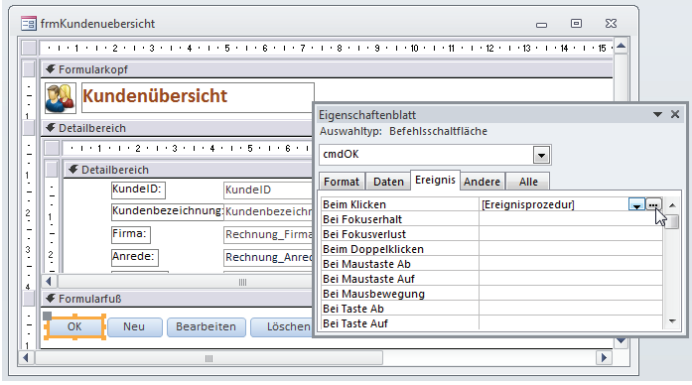


Abbildung 3.23: Hinzufügen einer Ereignisprozedur für das Ereignis *Beim Klicken* einer Schaltfläche

Es öffnet sich der VBA-Editor, der gleich ein Klassenmodul für das Formular *frmKundenuebersicht* erstellt und die ersten Zeilen der gewünschten Ereignisprozedur angelegt hat (siehe Abbildung 3.24). Das Klassenmodul nimmt alle durch die Ereignisse des Formulars und seiner Steuerelemente ausgelösten Ereignisprozeduren auf. Außerdem können Sie darin relativ einfach auf das Formular und die Steuerelemente Bezug nehmen, da das Formular ganz einfach mit dem Schlüsselwort *Me* referenziert werden kann – dazu später mehr.

4 Artikel verwalten

Ich glaube, dass ich es bereits vor einigen Kapiteln erwähnt habe: Faule Access-Entwickler kommen grundsätzlich schneller zum Ziel als andere. So werden wir es uns beim Gestalten der Formulare für die Verwaltung der Artikel erstmal richtig leicht machen.

Wie das aussieht? Nun: Wir benötigen genau wie bei der Verwaltung von Kunden ein Übersichtsformular, das die Artikel in einem Unterformular anzeigt. Außerdem soll ein Detailformular nach Wunsch die Details zu einem neuen oder vorhandenen Artikel anzeigen.

Sollten Sie das Kapitel zur Erstellung der Formulare für die Kundenverwaltung bereits durchgearbeitet haben, kommt Ihnen der Aufbau bekannt vor. Die folgenden Abschnitte stellen dar, wie Sie Ihre Artikelformulare schnellstmöglich zusammenstellen und dabei – zumindest teilweise – von bereits erledigten Arbeiten profitieren.

4.1 Übersichtsformular erstellen

Das Übersichtsformular zur Anzeige der Artikel erstellen wir auf Basis des Formulars zur Anzeige der Kunden. Das heißt, dass Sie das Hauptformular *frmKundenuebersicht* einfach kopieren und anpassen.

Dazu markieren Sie das Formular *frmKundenuebersicht* im Navigationsbereich und betätigen die Tastenkombinationen *Strg + C* (Kopieren) und *Strg + V* (Einfügen). Als Name des neuen Formulars geben Sie *frmArtikeluebersicht* an (siehe Abbildung 4.1).

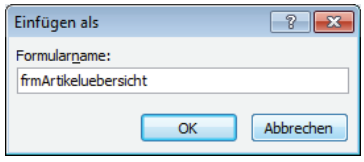


Abbildung 4.1: Kopieren und einfügen eines Formulars

Damit das Formular gleich Artikeldaten im Unterformular anzeigt, erstellen Sie zunächst ein solches – in diesem Fall von Grund auf. Das Formular erhält als Datenherkunft beziehungsweise Datensatzquelle die Tabelle *tblArtikel*. Nachdem Sie alle Felder der Tabelle aus der Feldliste in den Detailbereich der Entwurfsansicht gezogen haben, sieht dieser wie in Abbildung 4.2 aus. Sie erkennen hier noch das Feld *Beschreibung*, dieses haben wir jedoch entfernt – es reicht, wenn der Inhalt im Detailformular ersichtlich ist.

Stellen Sie die Eigenschaft *Standardansicht* auf *Datenblatt* ein, damit die Artikel in Listenform angezeigt werden.

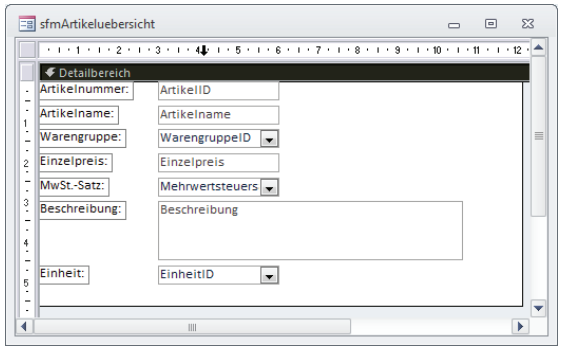


Abbildung 4.2: Das Unterformular *sfmArtikeluebersicht* in der Entwurfsansicht

Nach dem Speichern und Schließen des Formulars *sfmArtikeluebersicht* fügen Sie es dem Formular *frmArtikeluebersicht*, das ja noch vollständig wie das Formular zur Kundenverwaltung aussieht, als Unterformular hinzu. Da das Unterformularsteuerelement bereits vorhanden ist, brauchen Sie nur noch sein Herkunftsobjekt anzupassen.

Dies erledigen Sie mit der gleichnamigen Eigenschaft des Unterformular-Steuerelements in der Entwurfsansicht des Hauptformulars. Wählen Sie im Eigenschaftsfenster oben den Eintrag *sfm* aus (also das Unterformular-Steuerelement) und stellen Sie die Eigenschaft *Herkunftsobjekt* auf *Formulare.sfmArtikeluebersicht* ein (siehe Abbildung 4.3). Danach sieht zumindest das Unterformular schon einmal nach einem Artikelformular aus.

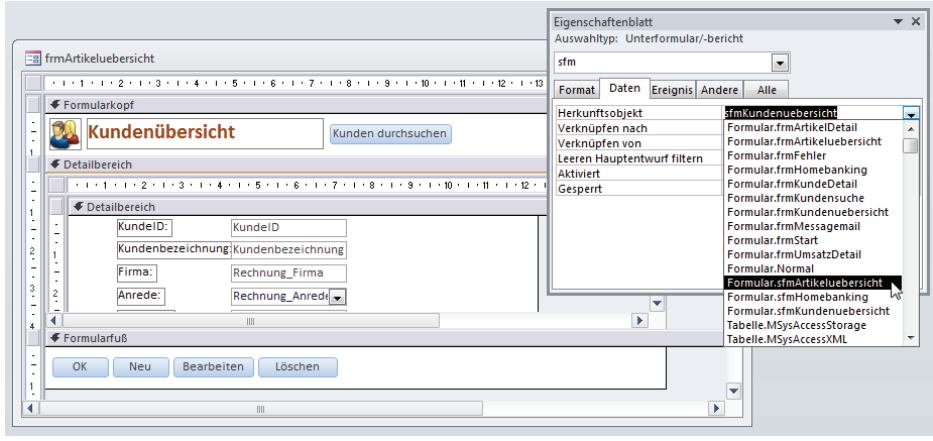


Abbildung 4.3: Einstellen eines anderen Unterformulars

Anschließend ändern Sie die übrigen Elemente, die den Eindruck erwecken, Sie würden hier mit einem Kundenformular arbeiten – also vor allem die Eigenschaft *Beschriftung* sowie die Elemente im Formularkopf und die Beschriftung der Schaltfläche *cmdSuche*.

Der Rest sieht oberflächlich erstmal gut aus – aber natürlich funktioniert der Code nicht, da er sich auf die Tabelle *tblKunden* und das Formular *frmKundeDetail* bezieht und nicht auf *tblArtikel* und *frmArtikelDetail*.

Sie könnten nun einfach alle relevanten Ausdrücke ändern – also beispielsweise die Zeile

```
strSQL = "DELETE FROM tblKunden WHERE KundeID = " & lngKundeID
```

in diese Zeile ändern:

```
strSQL = "DELETE FROM tblArtikel WHERE ArtikelID = " & lngArtikelID
```

Dazu müssten Sie an etwa zehn Stellen eingreifen. Davon ausgehend, dass Sie ein solches Übersichtsformular nicht nur für Kunden und Artikel, sondern noch für andere Objekte in dieser oder anderen Anwendungen einsetzen möchten, machen wir das Formular noch ein wenig flexibler.

Alle Elemente, die sich in Form einer Zeichenkette angeben lassen können, speichern wir in Konstanten:

```
Const strTabelle As String = "tblArtikel"
Const strPKFeld As String = "ArtikelID"
Const strDetailformular As String = "frmArtikelDetail"
Const strSuchformular As String = "frmArtikelsuche"
Const strMeldungKeineAuswahl As String = "Kein Artikel ausgewählt."
```

Das *strTabelle* nimmt die Tabelle auf, aus der das Formular seine Daten bezieht, *strPKFeld* den Namen des Primärschlüsselfeldes der Tabelle, *strDetailformular* das Formular, das zum Anzeigen der Details geöffnet werden soll, *strSuchformular* entsprechend das Suchformular und *strMeldungKeineAuswahl* soll als Meldungsfenster angezeigt werden, wenn der Benutzer ohne ausgewählten Datensatz einen Eintrag im Detailformular öffnen oder löschen möchte.

Die Konstanten sind im Code des gesamten Klassenmoduls des Formulars verfügbar.

Nun passen Sie die Ereignisprozeduren so an, dass diese auch für andere Daten als die der Tabelle *tblArtikel* eingesetzt werden können. Die Schaltfläche zum Anzeigen des Suchfensters etwa soll nicht mehr *"frmArtikelsuche"* öffnen, sondern das in der Konstanten *strSuchformular* angegebene Formular (das hier auch *frmArtikelsuche* heißt):

```
Private Sub cmdSuche_Click()
    DoCmd.OpenForm strSuchformular
End Sub
```

Ähnlich sieht es in der Prozedur aus, die das Detailformular zum Anlegen eines neuen Datensatzes öffnen soll:

```
Private Sub cmdNeu_Click()
    DoCmd.OpenForm strDetailformular, DataMode:=acFormAdd, WindowMode:=acDialog
```

```
    Me!sfm.Form.Requery
End Sub
```

Einige Änderungen mehr finden Sie in der Prozedur, die durch die Schaltfläche *Bearbeiten* ausgelöst wird. Die geänderten Elemente beziehungsweise die betroffenen Zeilen sind kursiv gedruckt. Den Anfang macht die Variable, welche den Wert des Primärschlüsselfeldes für den zu bearbeitenden Datensatz speichern soll. Statt *lngKundeID* oder *lngArtikelID* heißt diese einfach nur *lngID*.

Der Wert dieser Variablen wird nun nicht mehr mit *Me!sfm.Form!KundeID* ermittelt, stattdessen kommt eine andere Variante für den Zugriff auf die Felder eines Formulars zum Zuge. Dabei wird der Name des Feldes hinter der Formularreferenz (*Me!sfm.Form*) in Klammern angegeben. Hier könnten Sie *Me!sfm.Form("ArtikelID")* verwenden, aber *"ArtikelID"* ist ja in *strPKFeld* gespeichert – also kommt *Me!sfm.Form(strPKFeld)* zum Einsatz.

Auch die Meldung, die bei fehlender Auswahl eines Datensatzes angezeigt wird (hier *Kein Artikel ausgewählt*), wird in einer Konstanten gespeichert, in diesem Fall *strMeldungKeineAuswahl*.

Und die *WhereCondition*, die beim Aufruf des Formulars übergeben wird, stellt die Prozedur bis auf das Gleichzeitszeichen komplett aus Konstanten und Variablen zusammen:

```
Private Sub cmdBearbeiten_Click()
    Dim lngID As Long
    Dim strWhereCondition As String
    lngID = Nz(Me!sfm.Form(strPKFeld),0)
    If lngID = 0 Then
        MsgBox strMeldungKeineAuswahl
        Exit Sub
    End If
    strWhereCondition = strPKFeld & " = " & lngID
    DoCmd.OpenForm strDetailformular, DataMode:=acFormEdit, WindowMode:=acDialog, _
        WhereCondition:=strWhereCondition
    Me!sfm.Form.Requery
End Sub
```

Fehlt noch die Prozedur für die *Löschen*-Schaltfläche, die allerdings fast genauso aufgebaut ist wie die zum Öffnen des Detailformulars:

```
Private Sub cmdLoeschen_Click()
    Dim db As DAO.Database
    Dim lngID As Long
    Dim strSQL As String
    Set db = CurrentDb
    lngID = Nz(Me!sfm.Form(strPKFeld), 0)
    If lngID = 0 Then
```

```
            MsgBox strMeldungKeineAuswahl
            Exit Sub
        End If
        strSQL = "DELETE FROM " & strTabelle & " WHERE " & strPKFeld & " = " & lngID
        db.Execute strSQL, dbFailOnError
        Me!sfm.Form.Requery
        Set db = Nothing
    End Sub
```

Lediglich die Prozedur, die das Formular schließen soll, bleibt komplett erhalten:

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

Das Formular sieht schließlich wie in Abbildung 4.4 aus.

Abbildung 4.4: Das Formular *frmArtikeluebersicht* in der Formularansicht

4.2 Detailformular erstellen

Die Artikelübersicht enthält nun bereits einige Schaltflächen, die Formulare öffnen sollen, die noch nicht vorhanden sind – zum Beispiel das Detailformular. Im Prinzip können Sie auch hier wieder vom entsprechenden Formular zur Anzeige von Kunden abkupfern:

» Kopieren Sie das Formular *frmKundeDetail* nach *frmArtikelDetail*.

» Ändern Sie den Kopfbereich und die Beschriftung.

» Stellen Sie die Datenherkunft auf die Tabelle *tblArtikel* ein.

» Löschen Sie alle Steuerelemente bis auf das Textfeld zur Eingabe von Suchbegriffen und das dazu gehörende Listenfeld aus dem Detailbereich.

» Ziehen Sie alle Felder aus der Feldliste in den Entwurf und passen Sie Position und Größe nach Ihren Wünschen an.

Das Formular sieht nun im Entwurf wie in Abbildung 4.5 aus.

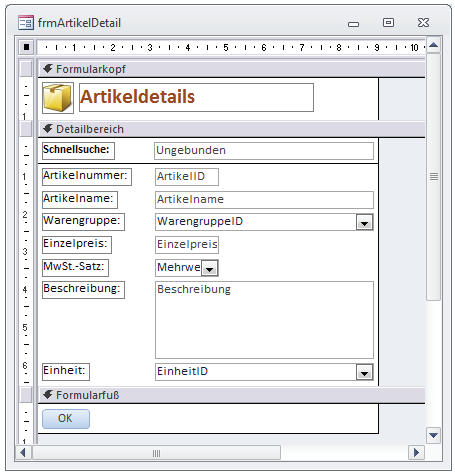


Abbildung 4.5: Das Formular *frmArtikelDetail* in der Entwurfsansicht

Das in der Abbildung ersichtliche Suchfeld wird später in »Schnellsuche für das Kundendetail-Formular« (Seite 359) zum Formular hinzugefügt. Den Code des Formulars brauchen Sie, sofern Sie das Formular *frmKundeDetail* in dem Zustand nach Kapitel »Detailformular erstellen« (Seite 89) kopiert haben, nicht zu ändern. Erweiterungen, die in weiteren Kapiteln vorgestellt werden, bedürfen allerdings einiger Anpassungen. Das Formular sieht in der Formularansicht wie in Abbildung 4.6 aus.

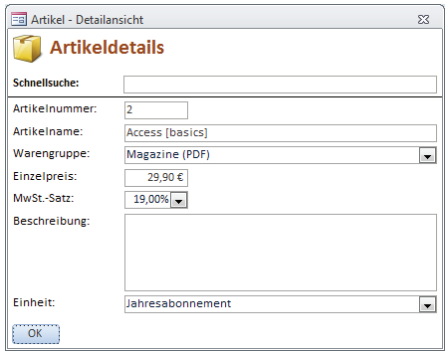


Abbildung 4.6: Die Artikeldetails in der Formularansicht

4.2.1 Warengruppen bearbeiten

Die Warengruppen sollen in einem eigenen Formular bearbeitet werden können – genau genommen in zwei Formularen. Das erste heißt *frmWarengruppenUebersicht* und zeigt eine

Liste aller Warengruppen, eine Schaltfläche zum Löschen, eine zum Bearbeiten und eine zum Anlegen von Warengruppen. Ein weiteres Formular zeigt die Warengruppe an und erlaubt das Ändern der Bezeichnung der Warengruppe beziehungsweise das Eintragen der Bezeichnung einer neuen Warengruppe.

Warengruppen-Übersicht

Legen Sie ein neues Formular namens *frmWarengruppenUebersicht* an und fügen Sie ihm einige Steuerelemente wie in Abbildung 4.7 hinzu:

» ein Bild und eine Überschrift im Kopfbereich

» ein Listenfeld namens *lst* im Detailbereich

» vier Schaltflächen namens *cmdOK*, *cmdNeu*, *cmdBearbeiten* und *cmdLoeschen* im Fuß des Formulars

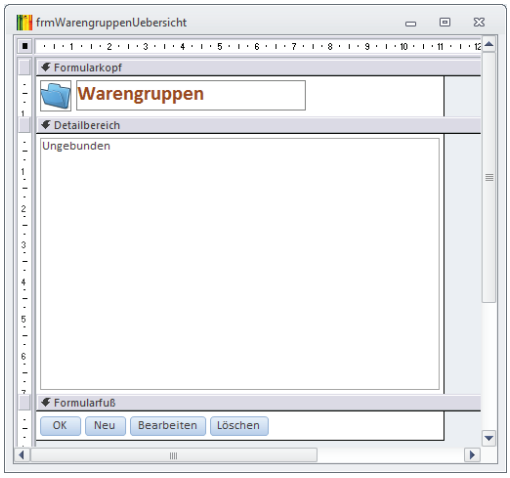


Abbildung 4.7: Entwurfsansicht des Formulars mit der Warengruppen-Übersicht

Das Listenfeld soll lediglich die Namen der Warengruppen anzeigen, und zwar in alphabetischer Reihenfolge. Die dazu nötige Datensatzherkunft fügen Sie durch einen Klick auf die Schaltfläche neben der Eigenschaft *Datensatzherkunft* im Eigenschaftenfenster hinzu. Im nun erscheinenden Abfrageentwurf wählen Sie die Tabelle *tblWarengruppen* als Datenquelle aus und fügen die beiden Felder der Tabelle zum Entwurfsraster der Abfrage hinzu. Stellen Sie die Sortierung für das Feld *Warengruppe* auf *Aufsteigend* ein (siehe Abbildung 4.8).

Vielleicht wollen Sie den Umweg über den Abfrage-Editor auch auslassen und direkt von Hand folgende Datensatzherkunft in das Eigenschaftsfenster eintragen:

```
SELECT WarengruppeID, Warengruppe FROM tblWarengruppen ORDER BY Warengruppe;
```

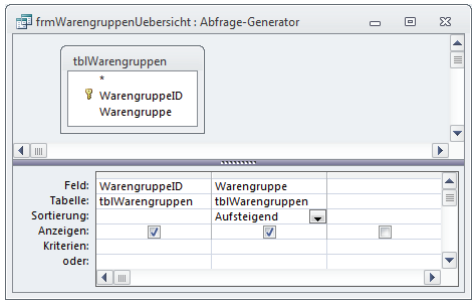


Abbildung 4.8: Datensatzherkunft des Listenfeldes *lst* des Formulars *frmWarengruppenUebersicht*

Damit das Listenfeld nur die Bezeichnungen der Warengruppen anzeigt und das Feld *WarengruppeID* ausblendet, stellen Sie die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *2* und *0cm* ein. Da das Formular selbst keine Daten anzeigt, in denen der Benutzer blättern könnte, brauchen Sie auch keine Navigationsschaltflächen oder einen Datensatzmarkierer. Überhaupt können Sie die Eigenschaften *Navigationsschaltflächen*, *Datensatzmarkierer*, *Trennlinien* und *Bildlaufleisten* auf den Wert *Nein* und *Automatisch zentrieren* auf *Ja* einstellen.

Das Formular sieht dann in der Formularansicht wie in Abbildung 4.9 aus.

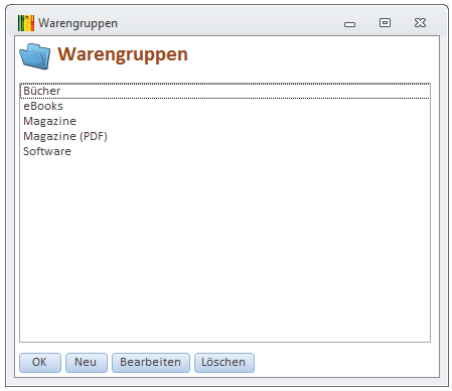


Abbildung 4.9: Formular zur Anzeige und Auswahl von Warengruppen

Warengruppen-Details bearbeiten

Bevor wir uns um die Funktionen des Listenfeldes und der Schaltflächen kümmern, legen wir noch schnell das Formular zum Anzeigen der Warengruppen-Details an. Dieses soll den Namen *frmWarengruppeDetail* erhalten und die Tabelle *tblWarengruppen* als *Datenherkunft*.

Dem Detailbereich fügen Sie aus der Feldliste des Formulars allein das Feld *Warengruppe* hinzu. Im Fußbereich legen Sie die beiden Schaltflächen *cmdOK* und *cmdAbbrechen* an (siehe Abbildung 4.10).

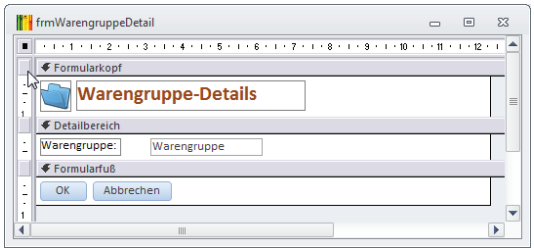


Abbildung 4.10: Detailansicht des Formulars *frmWarengruppeDetail*

Die Schaltfläche *cmdOK* soll das Formular einfach schließen. Dazu legen Sie die bereits bekannte Ereignisprozedur an:

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

Beim Abbrechen soll das Formular auch geschlossen werden, allerdings erst, nachdem die Änderungen am aktuellen Datensatz verworfen wurden:

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

Damit der Benutzer nur den aktuellen Datensatz bearbeiten und nicht zu anderen Datensätzen wechseln kann, stellen Sie die Eigenschaft *Zyklus* des Formulars *frmWarengruppeDetail* auf *Aktueller Datensatz* ein. Außerdem benötigen wir, da ohnehin immer nur ein einziger Datensatz gleichzeitig angelegt oder bearbeitet werden soll, noch einige weitere Einstellungen: *Navigationsschaltflächen, Datensatzmarkierer*, *Trennlinien* und *Bildlaufleisten* erhalten den Wert *Nein*, *Automatisch Zentrieren* den Wert *Ja*. Das Formular sieht anschließend wie in Abbildung 4.11 aus.

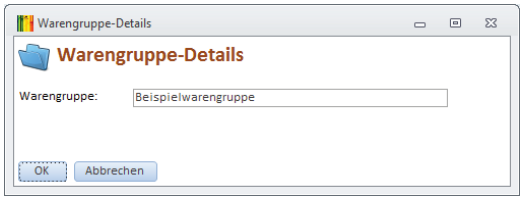


Abbildung 4.11: Formularansicht des Warengruppen-Detailformulars

Per Tabulator zwischen den Formularbereichen wechseln

Wenn Sie nun in die Formularansicht wechseln und nach der Eingabe einer neuen Warengruppe per Tabulator-Taste zur *OK*-Schaltfläche wechseln möchten, gelingt dies nicht. Der Grund ist,

dass dies standardmäßig nur innerhalb eines Bereichs möglich ist. Es gibt jedoch einen Trick: Sie legen im Detailbereich ein ungebundenes Textfeld an, nennen es *txtTabDetail*, stellen die Eigenschaft *Rahmenart* auf *Transparent* sowie *Höhe* und *Breite* jeweils auf *0cm* ein und legen dann eine Ereignisprozedur für das Ereignis *Beim Hingehen* an.

Diese füllen Sie mit folgender Anweisung:

```
Private Sub txtTabDetail_Enter()
    Me!cmdOK.SetFocus
End Sub
```

Sorgen Sie dafür, dass das Textfeld *txtTabDetail* in der Aktivierreihenfolge (Kontextmenü-Eintrag *Aktivierreihenfolge* eines der Steuerelemente des Formulars) direkt hinter dem Textfeld *txtWarengruppe* liegt (siehe Abbildung 4.12).

Wenn der Benutzer nun vom Textfeld *txtWarengruppe* aus die *Tabulator*-Taste betätigt, wird der Fokus zunächst auf das Textfeld *txtTabDetail* verschoben. Die obige Ereignisprozedur sorgt dann direkt dafür, dass der Fokus an die Schaltfläche *cmdOK* weitergereicht wird.

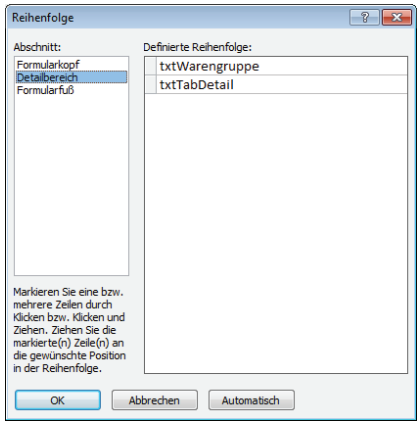


Abbildung 4.12: Einstellen der richtigen Aktivierreihenfolge

Nun soll der Fokus nach dem Verlassen der *Abbrechen*-Schaltfläche auch wieder beim ersten Steuerelement im Detailbereich landen, was wir durch ein weiteres Textfeld im Formularfuß mit transparentem Rahmen und dem Wert *0cm* für die Eigenschaften *Breite* und *Höhe* erreichen. Für dieses Textfeld namens *txtTabFooter* legen Sie folgende Ereignisprozedur an:

```
Private Sub txtTabFooter_Enter()
    Me!txtWarengruppe.SetFocus
End Sub
```

Auch hier müssen Sie die Aktivierreihenfolge beachten. Der Fokus wird dann wie in Abbildung 4.13 weitergereicht.

5 Bestellungen verwalten

Die Verwaltung der Bestellungen beinhaltet gleich eine ganze Reihe von Anforderungen. Die erste ist natürlich das eigentliche Aufnehmen einer Bestellung im klassischen Sinne, also direkt mit einem geeigneten Formular. Dieses Formular zeigt die Bestelldaten und die Bestellpositionen an. Mittlerweile hat sich neben der Bestellannahme durch Mitarbeiter am Telefon die Online-Bestellung breitgemacht. Es gibt kaum noch einen Hersteller oder Händler, der neben Ladenlokal und/oder Bestellung via Telefon oder Telefax noch keine Webseite mit integriertem Onlineshop besitzt.

Es wäre ja auch verrückt, sich dieser Möglichkeit zu verschließen – zumindest, wenn die Produkte oder Artikel für den Verkauf ohne persönliche oder telefonische Beratung geeignet sind. Neben dem klassischen Bestellformular ist natürlich auch der Import von Bestellungen aus externen Quellen möglich – darum können wir uns aus Platzgründen in diesem Buch leider nicht kümmern.

5.1 Bestellung aufnehmen

Zum Aufnehmen einer telefonischen oder schriftlich eingegangenen (aber nicht digital verarbeitbaren) Bestellung erstellen wir ein eigenes Bestellformular. Dieses bietet die Möglichkeit zum Auswählen (oder Anlegen) des Kunden, des Bestelldatums und weiterer Informationen wie natürlich der Bestellpositionen.

Das notwendige Formular sollte also dabei helfen, Daten für die Tabellen *tblBestellungen*, *tblBestellpositionen* und *tblKunden* zu erfassen – wobei die Kunden gegebenenfalls schon vorhanden sind und nur noch ausgewählt werden müssen. Allerdings sollte das Bestellformular dennoch die Möglichkeit bieten, schnell das Formular zum Erfassen eines Kunden zu öffnen – dazu später mehr.

Um den Aufbau des Formulars zu optimieren, stellen wir uns den Ablauf einer typischen Bestellung vor – egal, ob diese telefonisch oder schriftlich erfolgt. Als Erstes müssen Sie immer die Bestellung anlegen, also einen Datensatz in der Tabelle *tblBestellungen*. Danach ist interessant, wer bestellt – Sie wählen also entweder einen vorhandenen Kunden aus oder legen diesen neu an. Hier liegt schon ein gewisses Gewicht auf der Auswahl bestehender Kunden: Wenn der Kunde bereits vorhanden ist, sollte dieser möglichst schnell gefunden werden. In der Regel sollten eine oder mehrere der folgenden Informationen helfen: Kundennummer, Firma, Vorname, Nachname. Die Kundennummer könnte der Kunde einer vorhandenen Rechnung entnehmen, die übrigen Daten sollte er auswendig kennen ...

Neben der Auswahl des Kunden sollte die Bestellung ein Bestelldatum erhalten. Dieses können Sie beim Anlegen automatisch auf den heutigen Tag einstellen lassen und gegebenenfalls anpassen.

Schließlich folgen noch die wichtigsten Informationen, nämlich die gewünschten Artikel. Diese sollen in der Tabelle *tblBestellpositionen* gespeichert werden. Dabei muss der Artikel inklusive *Menge* und gegebenenfalls dem *Rabatt* erfasst werden.

Der *Einzelpreis* des Artikels und der *Mehrwertsteuersatz* sollen je nach gewähltem Artikel aus der Tabelle *tblArtikel* entnommen und in die Tabelle *tblBestellpositionen* übertragen werden.

Wie stellen wir dies nun dar – eine Bestellung mit einem Kunden und einer oder mehreren Bestellpositionen? Die Lösung ist der Einsatz eines Hauptformulars, das die Daten der Tabelle *tblBestellungen* anzeigt, mit einem Unterformular zur Eingabe der Bestellpositionen.

5.1.1 Das Hauptformular frmBestellungDetail

Als Grundgerüst für das Formular können Sie wieder das bewährte Detailformular verwenden – also beispielsweise das Formular *frmArtikelDetail*.

Kopieren Sie dieses, indem Sie es im Navigationsbereich markieren und dann die Tastenkombinationen *Strg + C* und *Strg + V* betätigen, und geben Sie den Namen *frmBestellungDetail* an.

Welche Elemente dieses Formulars können Sie weiter verwenden? Eigentlich alle bis auf die gebundenen Felder. Werfen Sie also alle Felder der Tabelle *tblArtikel* raus und stellen Sie die Eigenschaft *Datenherkunft* auf die Tabelle *tblBestellungen* ein.

Ändern Sie die Beschriftung in *Bestellung – Detailansicht* und das Bezeichnungsfeld im Kopfbereich in *Bestelldetails*. Fügen Sie dann die Felder der Tabelle *tblBestellungen* aus der Feldliste zum Detailbereich des Formulars hinzu. Das Suchfeld können wir später so umbauen, dass es sich zum Auffinden von Bestellungen nach Kunden-/Bestellnummer eignet.

Anschließend könnte das Formular etwa so wie in Abbildung 5.1 aussehen. Es ist etwas breiter geworden, weil ja unter den Feldern der Tabelle *tblBestellungen* ein Unterformular in der Datenblattansicht die Bestellpositionen anzeigen soll.

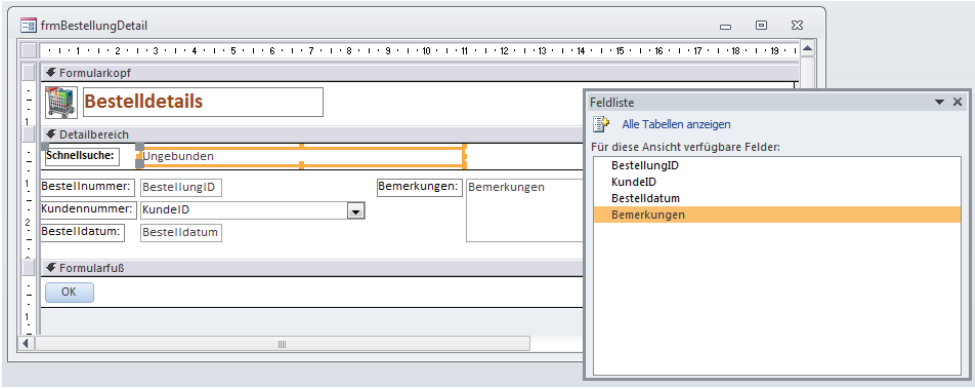


Abbildung 5.1: Das Formular *frmBestellungDetail* mit den Feldern der Tabelle *tblBestellungen*

5.1.2 Bestelldatum automatisch auf den heutigen Tag einstellen

Wenn das Feld *Bestelldatum* automatisch mit dem heutigen Datum gefüllt werden soll, gibt es verschiedene Möglichkeiten. Die erste ist, dass Sie festlegen, dass jeder Datensatz der Tabelle *tblBestellungen* standardmäßig beim Anlegen mit dem heutigen Datum versehen wird.

Dazu müssten Sie die Tabelle *tblBestellungen* öffnen und die Eigenschaft *Standardwert* des Feldes *Bestelldatum* wie in Abbildung 5.2 auf *Datum()* einstellen (oder englisch *Date()* – aber das wird in der deutschen Version ohnehin als *Datum()* angezeigt).

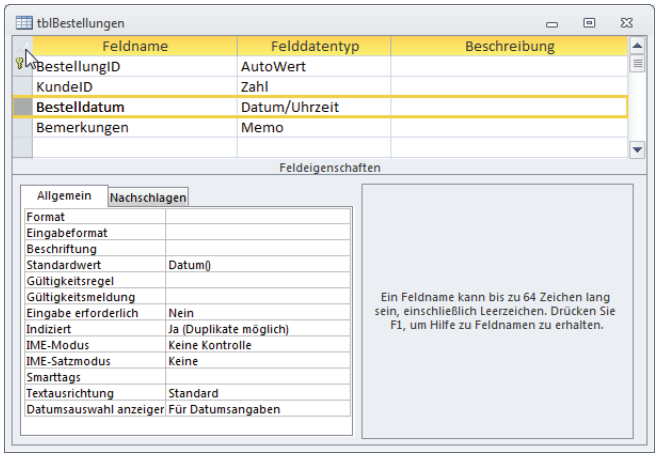


Abbildung 5.2: Einstellen des Standarddatums im Tabellenentwurf

Wenn Sie nun in die Datenblattansicht wechseln, zeigt Access für neue Datensätze gleich den entsprechenden Standardwert an (siehe Abbildung 5.3).

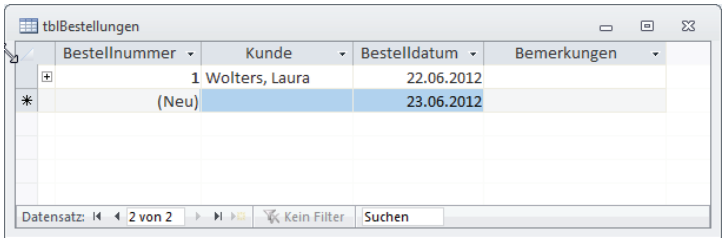


Abbildung 5.3: Standarddatum in der Datenblattansicht

Sie können das Datum auch nur innerhalb dieses Formulars mit einem Standardwert versehen. Dazu markieren Sie in der Entwurfsansicht des Formulars das Steuerelement, das an das Feld *Bestelldatum* gebunden ist, und stellen die Eigenschaft *Standardwert* auf den Wert *Datum()* ein (siehe Abbildung 5.4).

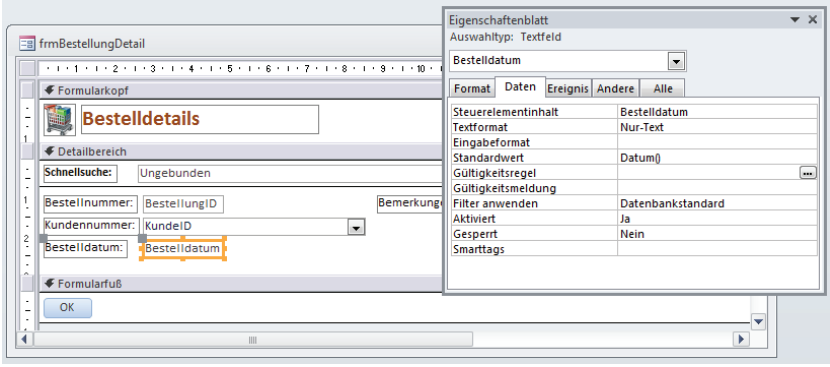


Abbildung 5.4: Einstellen des Standardwertes eines Datumsfeldes

5.1.3 Kundenauswahl

Grundsätzlich lassen sich mit dem Formular nun bereits die Daten der Tabelle *tblBestellungen* anlegen. Allerdings gelingt dies nur mit bestehenden Kunden, die Sie mit dem Kombinationsfeld *cboKundeID* auswählen können (wie gehabt, haben wir alle gebundenen Steuerelemente mit Präfixen wie *txt*, *cbo* oder *chk* versehen).

Was aber geschieht, wenn Sie eine Bestellung für einen neuen Kunden anlegen möchten? Neu beginnen und erstmal das Formular mit der Kundenübersicht öffnen, um von dort aus einen neuen Kunden anzulegen? Und dann nochmal mit der Aufnahme der Bestellung starten, jetzt, wo der Kunde vorliegt? Nein, das wäre deutlich zu unergonomisch. Also fügen wir dem Bestellformular zwei kleine Schaltflächen hinzu.

Mit der ersten öffnet der Benutzer die Kundenübersicht, um die dortigen erweiterten Möglichkeiten zur Kundensuche zu verwenden und schließlich den gewählten Kunden in das Auswahlfeld im Bestellformular zu übernehmen.

Die zweite öffnet direkt das Formular zum Anlegen eines neuen Kunden und übernimmt den neuen Kunden nach dem Schließen in das Bestellformular. Den Entwurf des Bestellformulars passen Sie dazu wie in Abbildung 5.5 an.

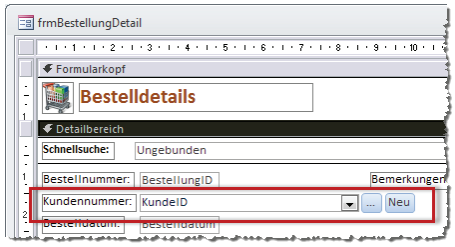


Abbildung 5.5: Schaltflächen zum Aufrufen der Kundenübersicht und zum Anlegen eines neuen Kunden

Bestehenden Kunden aus Kundenübersicht auswählen

Wenn Sie mit einem Klick auf die Schaltfläche *cmdKundenuebersicht* das Übersichtsformular *frmKundenuebersicht* öffnen, im dortigen Unterformular mit der Übersicht einen Kunden auswählen und diesen nach einem Klick auf die *OK*-Schaltfläche des Formulars übernehmen möchten, sieht das normalerweise so wie etwa im Abschnitt »Bearbeiten einer Warengruppe per Schaltfläche« (Seite 151) aus.

Das bedeutet, dass der Benutzer das Formular *frmKundenuebersicht* als modalen Dialog aufruft (*WindowMode:=acDialog*), der Benutzer den Kunden auswählt und dann auf die Schaltfläche *cmdOK* klickt, um die Kundenübersicht wieder zu schließen (Prozedur sieht unten).

Danach soll die aufrufende Prozedur den ausgewählten Eintrag im Unterformular einlesen und im Kombinationsfeld *cboKundeID* auswählen. Dabei gibt es allerdings ein Problem: Im Gegensatz zum Lookup-Formular *frmWarengruppen* aus dem oben genannten Kapitel wird das Formular *frmKundenuebersicht* direkt beim Anklicken der Schaltfläche *cmdOK* geschlossen und nicht etwa unsichtbar gemacht, sodass das aufrufende Formular die Auswahl nicht mehr auslesen kann. Nun gibt es zwei Möglichkeiten:

» Sie ändern den Code der Schaltfläche *cmdOK* so, dass die Prozedur das Formular mit *Me.Visible = False* einfach ausblendet. Die aufrufende Prozedur kann dann den ausgewählten Kunden auswählen und das Formular schließen.

» Sie referenzieren das Formular nach dem Öffnen mit einer Objektvariablen und implementieren die *Beim Entladen*-Ereignisprozedur im Klassenmodul des aufrufenden Formulars. So können Sie im Formular *frmBestellungDetail* einfach abwarten, bis der Benutzer das Formular *frmKundenuebersicht* schließt, und im *Beim Entladen*-Ereignis noch schnell den aktuellen Kunden auslesen.

Letztere Variante hört sich zwar etwas komplizierter an, ist aber letztlich die bessere Lösung. Sie brauchen den Code des Formulars *frmKundenuebersicht* nicht zu ändern, nur damit ein anderes Formular auf die enthaltenen Daten zugreifen kann. Noch besser: Das Formular *frmKundenuebersicht* braucht noch nicht einmal zu wissen, dass es von einem anderen Formular aufgerufen wurde.

Um dies zu realisieren, fügen Sie zunächst im Kopf des Klassenmoduls des Formulars *frmBestellungDetail* folgende Zeile ein:

```
Dim WithEvents objKundenuebersicht As Form
```

Diese deklariert eine Objektvariable, um auf das Formular *frmKundenuebersicht* zu verweisen.

Wenn der Benutzer nun auf die Schaltfläche *cmdKundenuebersicht* klickt, soll die folgende Prozedur ausgelöst werden:

```
Private Sub cmdKundenuebersicht_Click()
    DoCmd.OpenForm "frmKundenuebersicht", OpenArgs:=Nz(Me!cboKundeID)
```

```
    Set objKundenuebersicht = Forms!frmKundenuebersicht
    With objKundenuebersicht
        .OnUnload = "[Event Procedure]"
        .Modal = True
    End With
End Sub
```

Diese öffnet zunächst das Formular *frmKundenuebersicht*. Dann weist es der Eigenschaft *OnUnload* den Wert *[Event Procedure]* zu, was bedeutet, dass beim Eintreten des Ereignisses *Beim Entladen* auch in diesem Klassenmodul nach einer Implementierung der Ereignisprozedur gesucht werden soll. Schließlich wird die Eigenschaft *Modal* auf *True* eingestellt, wodurch das Formular als modaler Dialog angezeigt wird. Dies sorgt dafür, dass der Benutzer erst wieder mit anderen Elementen der Access-Benutzeroberfläche weiterarbeiten kann, wenn er dieses Formular geschlossen hat.

Warum haben wir das Formular nicht direkt durch Angabe des Parameters *WindowMode:=acDialog* als modalen Dialog geöffnet? Ganz einfach: Weil dann auch die aufrufende Prozedur angehalten wird. Das bedeutet, dass wir dem Formular nicht mitteilen können, dass im aufrufenden Klassenmodul noch eine Implementation des Ereignisses *Beim Entladen* vorliegt (*OnUnload = "[Event Procedure]"*).

Diese Prozedur legen wir nun wie folgt an (am schnellsten durch Auswahl von *objKundenuebersicht* im linken und *OnUnload* im rechten Kombinationsfeld des Codefensters):

```
Private Sub objKundenuebersicht_Unload(Cancel As Integer)
    If Not IsNull(objKundenuebersicht!sfm.Form!KundeID) Then
        Me!cboKundeID = objKundenuebersicht!sfm.Form!KundeID
    End If
End Sub
```

Die Prozedur erledigt nichts anderes, als den Wert des Feldes *KundeID* des aktuell im Unterformular des Formulars *frmKundenuebersicht* ausgewählten Datensatzes als Wert des Kombinationsfeldes *cboKundeID* festzulegen – allerdings nur, wenn auch ein Datensatz ausgewählt ist. Dadurch wird dann der entsprechende Kunde im Kombinationsfeld angezeigt.

Neuen Kunden anlegen und auswählen

Die Schaltfläche *cmdNeuerKunde* soll beim Anklicken das Formular *frmKundeDetail* öffnen und dem Benutzer die Möglichkeit bieten, einen neuen Kunden anzulegen. Nach dem Anlegen und Schließen des Formulars soll der Kunde direkt im Kombinationsfeld *cboKundeID* des aufrufenden Formulars *frmBestellungDetail* angezeigt werden.

Dies realisieren wir fast genauso wie bei der vorherigen Schaltfläche. Zunächst benötigen wir eine entsprechende Objektvariable, deren Ereignisprozeduren wir zum Abfangen des *Beim Entladen*-Ereignisses des Formulars *frmKundeDetail* nutzen können:

```
Dim WithEvents objKundeDetail As Form
```

Die Prozedur, die beim Anklicken der Schaltfläche *cmdNeuerKunde* ausgelöst wird, öffnet diesmal direkt das Formular zum Anlegen eines neuen Kunden, legt fest, dass eine Implementierung des Ereignisses *Form_Unload* vorliegen könnte, und stellt das Formular auf den modalen Modus um. Die *OpenForm*-Methode erhält diesmal noch den Parameter *DataMode:=acFormAdd*, damit sie gleich einen neuen Datensatz anzeigt:

```
Private Sub cmdNeuerKunde_Click()
    DoCmd.OpenForm "frmKundeDetail", DataMode:=acFormAdd
    Set objKundeDetail = Forms!frmKundeDetail
    With objKundeDetail
        .OnUnload = "[Event Procedure]"
        .Modal = True
    End With
End Sub
```

Die Prozedur, die beim Schließen des Formulars *frmKundeDetail* ausgelöst wird, sieht schließlich so aus:

```
Private Sub objKundeDetail_Unload(Cancel As Integer)
    If Not IsNull(objKundeDetail!KundeID) Then
        Me!cboKundeID = objKundeDetail!KundeID
    End If
End Sub
```

Auch diese Prozedur prüft zunächst, ob das Formular einen Datensatz mit einer Kundennummer anzeigt. Dies ist gegebenenfalls nicht der Fall, wenn der Benutzer das Formular schließt, bevor er überhaupt Daten eingegeben hat.

5.1.4 Das Unterformular sfmBestellungDetail

Wir benötigen nun noch das Unterformular zur Eingabe der Bestellpositionen. Dieses Formular legen Sie unter dem Namen *sfmBestellungDetail* neu an. Es benötigt weder Formularkopf noch -fuß, sondern nur einen Detailbereich.

Interessant ist bereits die Datenherkunft. Das Formular soll ja sowohl die Daten aus der Tabelle *tblBestellpositionen* anzeigen als auch zumindest den Namen des Artikels aus der Tabelle *tblArtikel*. Da liegt die Vermutung nahe, dass als Datenherkunft eine Abfrage zum Einsatz kommt, welche die beiden Tabellen *tblArtikel* und *tblBestellpositionen* enthält, die ja über das Feld *ArtikelID* verknüpft sind.

Ob dies so ist, beantwortet die Frage nach dem Ablauf beim Hinzufügen einer Bestellposition zur Bestellung. Typischerweise geben Sie zunächst die Kundendaten ein und fügen dann die Bestellpositionen hinzu. Welches ist das wichtigste Merkmal einer Bestellposition? Richtig: der

Artikel. Diesen sollte der Benutzer möglichst komfortabel auswählen können. Das gelingt am einfachsten mit einem Kombinationsfeld, das seine Werte auch noch alphabetisch sortiert anzeigt. Die Eingabe sollte also wie in Abbildung 5.6 aussehen.

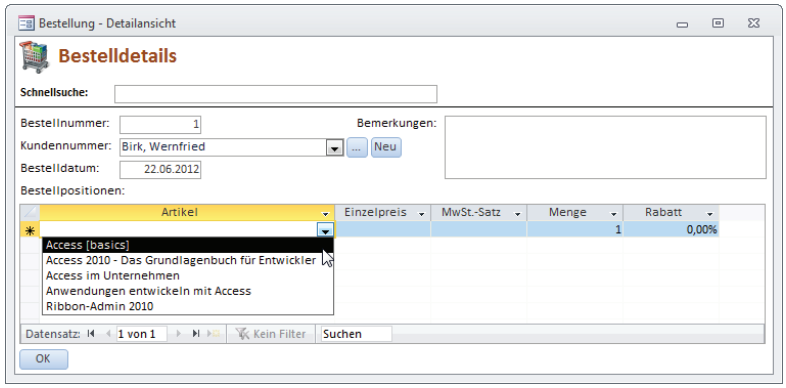


Abbildung 5.6: So soll das Hinzufügen eines Artikels zu einer Bestellung aussehen.

Nach der Auswahl soll das Formular automatisch die notwendigen Daten für den gewählten Artikel aus der Tabelle *tblArtikel* auslesen und in die Felder *Einzelpreis* und *Mehrwertsteuersatz* der frisch hinzugefügten Bestellposition eintragen.

Dem Benutzer bleibt dann noch die Aufgabe, die *Menge* und gegebenenfalls den *Rabatt* anzupassen – aufgrund der in der Tabelle *tblBestellpositionen* definierten Standardwerte werden diese ja auf *1* beziehungsweise *0,00%* eingestellt.

Schauen wir also einmal genau hin: Wir benötigen definitiv die Felder *Einzelpreis*, *Mehrwertsteuersatz*, *Menge* und *Anzahl* aus der Tabelle *tblBestellpositionen* (wobei die Letztgenannten aus der Tabelle *tblArtikel* gefüllt werden). Woher aber stammt das Feld zur Auswahl des Artikels? Ein Blick auf die Tabelle *tblBestellpositionen* liefert die Antwort (siehe Abbildung 5.7).

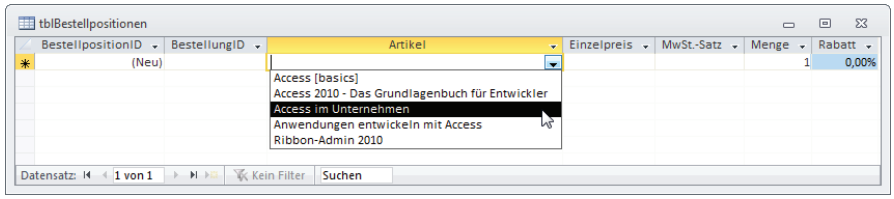


Abbildung 5.7: Die Tabelle *tblBestellpositionen* bietet mit dem Nachschlagefeld *ArtikelID* bereits die Möglichkeit, Artikel gleichzeitig auszuwählen und diese auch anzuzeigen.

Wir brauchen die Tabelle *tblArtikel* nämlich gar nicht: Alles, was wir benötigen, ist bereits im Nachschlagefeld *ArtikelID* der Tabelle *tblBestellpositionen* enthalten. Gespeichert werden soll ohnehin nur der Primärschlüsselwert der Tabelle *tblArtikel*, nämlich *ArtikelID*, und zwar im

gleichnamigen Fremdschlüsselfeld der Tabelle *tblBestellpositionen*. Es reicht also vorerst völlig aus, die Tabelle *tblBestellpositionen* als Datenherkunft des Unterformulars *sfmBestellungDetail* einzusetzen.

Anlegen des Unterformulars sfmBestellungDetail

Schreiten wir also zur Tat und legen ein neues Formular namens *sfmBestellungDetail* an. Fügen Sie die Tabelle *tblBestellpositionen* als *Datenherkunft* hinzu und schieben Sie die Felder *ArtikelID*, *Einzelpreis*, *Mehrwertsteuersatz*, *Menge* und *Rabatt* zum Detailbereich des Formulars hinzu (siehe Abbildung 5.8). Warum nur diese Felder und nicht *BestellpositionID* und *BestellungID*? Formulare in der Datenblattansicht zeigen normalerweise alle Felder an, die Sie zum Detailbereich hinzugefügt haben. Das Feld *BestellpositionID* brauchen wir dort gar nicht und das Feld *BestellungID* wird nur zum Synchronisieren der Datensätze des Unterformulars mit denen des Hauptformulars benötigt – und dazu braucht es nicht sichtbar zu sein.

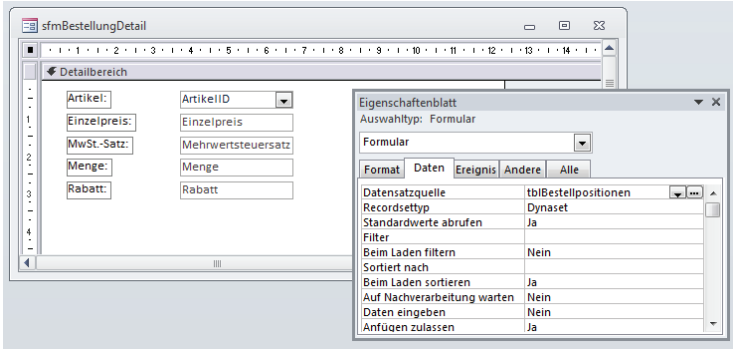


Abbildung 5.8: Datenherkunft und Aufbau des Unterformulars *sfmBestellungDetail*

Nachdem Sie die Felder zum Formular hinzugefügt haben, stellen Sie noch die Eigenschaft *Standardansicht* des Formulars auf *Datenblatt* ein. Sie können auch gleich an Ort und Stelle einmal in die Datenblattansicht wechseln und die Schriftgröße entsprechend Ihren Wünschen anpassen.

Anschließend fügen Sie das Unterformular zum Hauptformular *frmBestellungDetail* hinzu. Diesmal gehen wir etwas anders als sonst vor:

» Schließen Sie das Formular *sfmBestellungDetail*.

» Öffnen Sie das Formular *frmBestellungDetail* in der Entwurfsansicht.

» Fügen Sie im unteren Bereich ein Unterformular-Steuerelement hinzu, indem Sie es im Ribbon unter *Entwurf|Steuerelemente* markieren und dann in der gewünschten Größe aufziehen.

» Legen Sie den Namen *sfm* für das Unterformular-Steuerelement fest.

» Wählen Sie für die Eigenschaft *Herkunftsobjekt* den Eintrag *Formular.sfmBestellungDetail* aus (siehe Abbildung 5.9).

Das war es schon: Access erkennt nun automatisch, dass sich im Unterformular eine Tabelle befindet, deren Fremdschlüsselfeld *BestellungID* mit dem Primärschlüsselfeld der Datenherkunft des Hauptformulars verknüpft ist, und legt eine entsprechende Verknüpfung auch für das Unterformular-Steuerelement fest.

Dazu verwendet es die beiden Eigenschaften *Verknüpfen von* und *Verknüpfen nach* (siehe Abbildung 5.10).

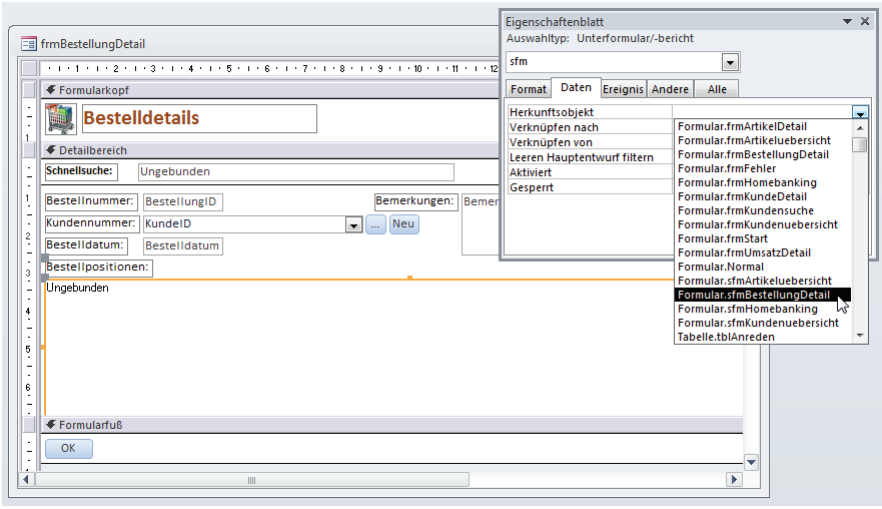


Abbildung 5.9: Haupt- und Unterformular

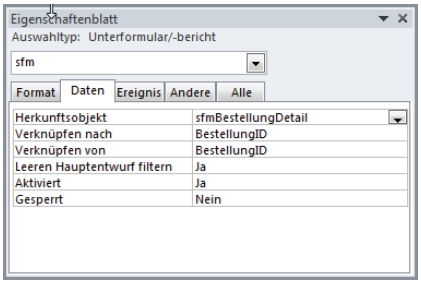


Abbildung 5.10: Angabe der Felder, über welche Haupt- und Unterformular synchronisiert werden sollen

Sie können nun bereits in die Formularansicht wechseln und Artikel zu Bestellungen hinzufügen. Dadurch, dass die Verknüpfung zwischen den Tabellen aus Haupt- und Unterformular auch für das Unterformular-Steuerelement eingetragen wurde, zeigt das Unterformular jeweils nur die zur Bestellung im Hauptformular gehörenden Bestellpositionen an.

Unterformular verankern

Damit sich das Unterformular beim Ändern der Größe des Formulars *frmBestellungDetails* in der Breite und in der Höhe anpasst, stellen Sie noch die beiden Eigenschaften *Horizontaler Anker* und *Vertikaler Anker* auf *Beide* ein. Dadurch wird das Unterformular nach rechts und/oder unten erweitert, wenn Sie das Formular vergrößern.

Achtung: Wenn Sie ein Steuerelement, an das ein Bezeichnungsfeld gebunden ist (wie etwa Textfelder, Listenfelder, Kombinationsfelder oder Unterformulare), auf diese Weise verankern, stellt Access merkwürdigerweise die Eigenschaft *Horizontaler Anker* des Bezeichnungsfeldes auf *Rechts* und *Vertikaler Anker* auf *Unten* ein. Diese Änderung müssen Sie noch rückgängig machen und die Eigenschaften auf *Links* und *Oben* einstellen.

Einzelpreis und Mehrwertsteuersatz automatisch einstellen

Natürlich möchten Sie für eine Bestellposition möglichst nur den Artikel auswählen und gegebenenfalls die Standardwerte für die Felder *Anzahl* und *Rabatt* anpassen. Der *Einzelpreis* und der *Mehrwertsteuersatz* sind ja in der Tabelle *tblArtikel* gespeichert und sollen von dort ausgelesen und in den neuen Datensatz der Tabelle *tblBestellpositionen* eingetragen werden.

Wie immer müssen wir ein Ereignis bestimmen, zu dem diese Aktion geschieht. Soll es beim Auswählen des Artikels passieren oder vielleicht sogar erst beim Speichern der Bestellposition? Nein, wir wollen gleich nach der Auswahl mit dem Kombinationsfeld den *Einzelpreis* und den *Mehrwertsteuersatz* im neuen Datensatz vorfinden.

Dazu schließen Sie das Formular *frmBestellungDetail* und öffnen das Unterformular *sfmBestellungDetail* in der Entwurfsansicht. Markieren Sie das Steuerelement *cboArtikelID* und legen Sie für die Eigenschaft *Nach Aktualisierung* den Wert *[Ereignisprozedur]* an (siehe Abbildung 5.11).

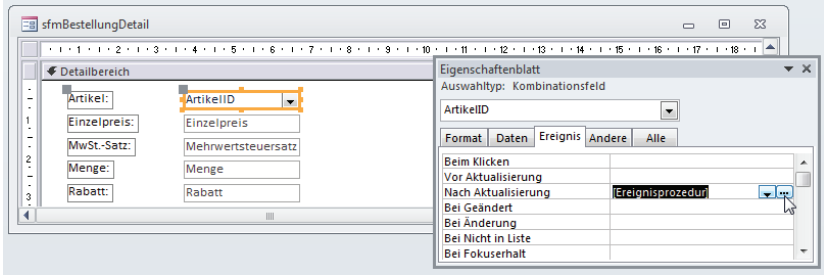


Abbildung 5.11: Anlegen des Ereignisses *Nach Aktualisierung*

Nach einem Mausklick auf die Schaltfläche mit den drei Punkten zeigt Access das Formularmodul mit der leeren Prozedur *cboArtikelID_AfterUpdate* an, die Sie wie folgt füllen:

```
Private Sub cboArtikelID_AfterUpdate()
    Dim lngArtikelID As Long
```

```
    Dim lngMehrwertsteuersatzID As Long
    lngArtikelID = Nz(Me!cboArtikelID, 0)
    If lngArtikelID > 0 Then
        Me!txtEinzelpreis = _
            DLookup("Einzelpreis", "tblArtikel", "ArtikelID = " & lngArtikelID)
        lngMehrwertsteuersatzID = _
            DLookup("MehrwertsteuersatzID", "tblArtikel", "ArtikelID = " & lngArtikelID)
        Me!txtMehrwertsteuersatz = DLookup("Mehrwertsteuersatz", _
            "tblMehrwertsteuersaetze", "MehrwertsteuersatzID = " & lngMehrwertsteuersatzID)
    End If
End Sub
```

Die Prozedur liest zunächst die Nummer des mit dem Kombinationsfeld ausgewählten Artikels aus und speichert diese in der Variablen *lngArtikelID*. Diese Variable kann auch den Wert *0* erhalten, wenn der Benutzer eine leere Zeichenkette in das Feld eingegeben hat.

In diesem Fall geschieht nichts weiter, denn die in der folgenden *If...Then*-Bedingung enthaltenen Anweisungen werden nur für Werte von *lngArtikelID* größer als *0* ausgeführt.

Dann folgen drei Anweisungen: Die erste liest mit der *DLookup*-Anweisung den Wert des Feldes *Einzelpreis* aus der Tabelle *tblArtikel* für den Datensatz aus, dessen Feld *ArtikelID* dem in *lngArtikelID* enthaltenen Wert entspricht, und fügt diesen Wert zum Feld *txtEinzelpreis* hinzu. Die zweite ermittelt den gleichen Wert für das Feld *MehrwertsteuersatzID* aus der Tabelle *tblArtikel* und speichert ihn in der Variablen *lngMehrwertsteuersatzID* zwischen.

Bedenken Sie: Der Mehrwertsteuersatz wird aus einer Lookup-Tabelle ausgelesen und nicht direkt in der Tabelle *tblArtikel* gespeichert. Den eigentlichen Mehrwertsteuersatz ermittelt erst die folgende Anweisung, die ebenfalls per *DLookup* auf den Wert *Mehrwertsteuersatz* der Tabelle *tblMehrwertsteuersaetze* zugreift, dessen Feld *MehrwertsteuersatzID* dem Wert der Variablen *lngMehrwertsteuersatzID* entspricht. Das Ergebnis der *DLookup*-Anweisung wird wiederum in das Textfeld *txtMehrwertsteuersatz* des Unterformulars *sfmBestellungDetail* eingetragen.

Somit landen Einzelpreis und Mehrwertsteuersatz einer Bestellposition nun automatisch im entsprechenden Datensatz (siehe Abbildung 5.12).

5.1.5 Berechnungen: Position und Bestellsumme

Gelegentlich möchte der Kunde bei einer telefonischen Bestellung direkt wissen, wie hoch der Rechnungsbetrag ausfallen wird. Dazu fügen wir dem Bestellformular noch die benötigten Steuerelemente mit den entsprechenden Berechnungsformeln hinzu:

» ein Textfeld im Unterformular, das den Betrag einer Rechnungsposition ausgibt, und

» einige Textfelder im Hauptformular, welche die Nettosumme, die Mehrwertsteuer und die Bruttosumme ausgeben.

6 Rechnungsbericht

In diesem Kapitel lernen Sie einen Bericht zum Anzeigen und Ausdrucken von Rechnungen kennen. Im Kapitel »Rechnungen verwalten« (Seite 225) fügen wir dann die Funktionen zur Anwendung hinzu, mit denen die Rechnung als PDF-Dokument gespeichert und vom entsprechenden Bestelldatensatz aus verlinkt wird. Schließlich ist neben dem Ausdruck zum anschließenden Versenden per Post auch der Versand per E-Mail vorgesehen.

6.1 Überlegungen und Vorarbeiten

Vor der Erstellung eines Rechnungsberichts ist ein wenig Planung nötig. Am besten nehmen Sie eine bestehende Rechnung im Papierformat, die alle nötigen Daten enthält, als Vorlage, skizzieren die notwendigen Anpassungen und begeben sich dann an die Umsetzung.

6.1.1 Bestellpositionen und Bestellsummen

Ich muss zugeben, dass ich eine Weile überlegt habe, wie ich den Bereich des Berichts gestalte, der die Rechnungspositionen enthält, und wie die Bestellsummen im unteren Teil des Berichts abgebildet werden sollen.

Landen Mehrwertsteuersatz und -betrag nur in den Bestellpositionen oder in den Bestellsummen? Und wie werden die unterschiedlichen Mehrwertsteuersätze dabei berücksichtigt? Zumindest die Mehrwertsteuer muss für die einzelnen Positionen ersichtlich sein (außer natürlich, alle Positionen enthalten den gleichen Mehrwertsteuersatz – aber das wäre zu einfach ...).

Außerdem muss die Summe der Mehrwertsteuer für alle Mehrwertsteuersätze der Rechnung zu entnehmen sein – entweder in den einzelnen Bestellpositionen oder als Summe. Und wie sollen die verschiedenen Berechnungen erfolgen? Beispiel: Eine Rechnungsposition enthält einen Artikel mit dem Preis *10 EUR*, der Menge *3*, einem Steuersatz von *19%* und einem Rabatt von *10%*.

Soll der Original-Einzelpreis abgebildet werden oder direkt der rabattierte? Soll der Einzelpreis überhaupt erscheinen oder gleich der Preis für die angegebene Anzahl? Ich drehte mich im Kreis und stellte fest: Egal, wie ich es mache – der Entwickler bekommt vom Auftraggeber sowieso individuelle Anforderungen, die mit hoher Wahrscheinlichkeit von meiner Umsetzung abweichen werden. Also soll jede Position die folgenden Daten enthalten:

» Position

» Artikelname

» Netto-Einzelpreis inklusive Rabatt

» Mehrwertsteuersatz

» Mehrwertsteuerbetrag

» Brutto-Gesamtpreis

Da dies bereits die Mehrwertsteuerbeträge für alle Positionen abbildet, soll unter den Rechnungspositionen nur noch die Rechnungssumme angezeigt werden. Zusätzlich werden wir dort einen Satz hinzufügen, der den Mehrwertsteuerbetrag für die verschiedenen Mehrwertsteuersätze ausgibt.

6.1.2 Änderungen am Datenmodell

Die für die Rechnung benötigten Daten liegen bereits vor – zumindest die Kunden- und Bestelldaten. Neben diesen Daten soll der Bericht auch noch einige Daten des Versenders erhalten. Hier stellt sich die Frage, ob man diese Daten fest im Bericht verankert oder diese gegebenenfalls in einer Stammdatentabelle speichert und von dort einliest.

Die erste Lösung ist einfacher, weil dazu keine weitere Tabelle und auch kein Formular zur Eingabe der Absenderdaten nötig ist. Wenn Sie jedoch planen, dem Kunden (oder sich selbst) eine Anwendung mit geringem Wartungsaufwand zu bescheren, sollten Sie die Daten des Rechnungsversenders nicht direkt im Bericht speichern.

Sobald sich auch nur eine Information ändert, sei es die Adresse, die Telefonnummer oder die Bankverbindung, müssen Sie in den Entwurf des Berichts eingreifen. Wenn Sie hingegen eine Tabelle mit den Stammdaten anlegen und dem Benutzer ein Formular zum Anpassen der Daten bereitstellen, kann dieser solche Änderungen selbst ganz einfach durchführen und braucht nicht auf den Entwickler zurückzugreifen.

Grundsätzlich können Sie einige Bausteine der Rechnung in einer solchen Tabelle unterbringen, zum Beispiel die Absenderzeile, die sich normalerweise über der Empfängeranschrift befindet.

Wir gehen später in diesem Kapitel auf dieses Thema ein. Im ersten Entwurf des Berichts erhält dieser feste Bezeichnungsfelder mit den Informationen, die für jede Rechnung gleich sein sollen. Später ersetzen wir diese Elemente durch Texte, die in einer Tabelle gespeichert und per Formular leicht geändert werden können, falls sich einmal die Absenderadresse oder andere Daten des Rechnungserstellers ändern.

6.2 Bericht erstellen

Einen neuen Bericht legen Sie mit dem Ribbon-Befehl *Erstellen|Berichte|Berichtsentwurf* an. Den leeren Bericht speichern Sie am besten erst einmal, und zwar unter dem Namen *rptRechnung*.

6.2.1 Datenherkunft definieren

Bevor wir nun Steuerelemente zum Bericht hinzufügen können, benötigen wir eine Datenherkunft – am besten in Form einer gespeicherten Abfrage, die wir im Folgenden zusammenstellen.

Ohne Datenherkunft könnten Sie bereits den Briefkopf layouten – aber darum kümmern wir uns später. Erstmal legen wir die Struktur des Rechnungsberichts fest, und dazu benötigen wir eine Datenherkunft.

Erstellen Sie also eine neue Abfrage und speichern Sie diese unter dem Namen *qryRechnung*. Welche Daten benötigen wir alle für den Bericht und aus welchen Tabellen stammen diese? Schauen wir uns die folgende Auflistung an:

» *tblKunden*: Die Rechnungsanschrift ist bereits vorbereitet und kann dem Feld *Rechnungsanschrift* entnommen werden. Außerdem benötigen wir die Kundennummer aus dem Feld *KundeID*. Da die Anrede dort bereits integriert ist, brauchen wir die Anrede zumindest nicht für die Anschrift. Allerdings soll im Betreff der Nachname des Kunden erscheinen, in diesem Fall der Nachname der Rechnungsadresse (*Rechnung_Nachname*).

» Für die Briefanrede benötigen wir die Tabelle *tblAnreden* nun doch: Daraus entnehmen wir die Werte des Feldes *Briefanrede*, zum Beispiel *Sehr geehrter Herr [Rechnung_Nachname]*. Den Platzhalter *[Rechnung_Nachname]* füllen wir zur Laufzeit.

» *tblBestellungen*: Diese Tabelle steuert die Felder *BestellungID*, *Bestelldatum* und *RechnungAm* bei.

» *tblBestellpositionen*: Aus dieser Tabelle entnehmen wir die Felder *Einzelpreis*, *Mehrwertsteuersatz* und *Menge*.

» *tblArtikel*: Fehlt noch der Name des Artikels, also das Feld *Artikelname*.

Wie nun fügen wir die Daten dieser vielen Tabellen zum Bericht hinzu? Nach dem, wie wir bisher die Formulare erstellt haben, ist das doch sicher irre viel Aufwand mit einigen Unterberichten et cetera.

Aber nein, es ist viel einfacher: Wir packen einfach alle benötigten Tabellen und Felder in eine einzige Abfrage! Wie wir die Felder nachher im Bericht platzieren, damit die Daten wie gewünscht angezeigt werden, erläutern wir im Anschluss.

Die resultierende Abfrage sieht wie in Abbildung 6.1 aus (die verwendeten Felder sind im oberen Bereich mit Pfeilen markiert).

Fügen Sie außerdem noch ein Feld zur Berechnung des rabattierten Einzelpreises hinzu:

```
Nettopreis: [tblBestellpositionen].[Einzelpreis]*(1-[Rabatt])
```

Beim Feld *Einzelpreis* müssen Sie zusätzlich die Tabelle angeben, da dieses Feld sowohl in der Tabelle *tblBestellpositionen* als auch *tblArtikel* vorkommt.

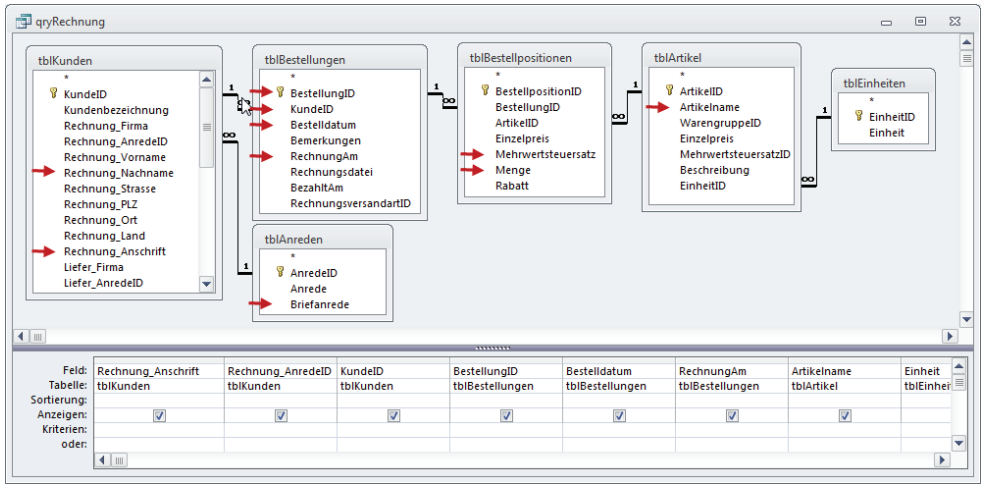


Abbildung 6.1: Datenherkunft des Berichts *rptRechnung*

Auch der Mehrwertsteuerbetrag soll gleich in der Abfrage berechnet werden. Dieser bezieht sich auf den Nettopreis (also den im vorherigen Feld berechneten, bereits rabattierten Einzelpreis), die Menge und dem Mehrwertsteuersatz:

```
Mehrwertsteuerbetrag: [Nettopreis]*[Menge]*[tblBestellpositionen].[Mehrwertsteuersatz]
```

Der Bruttopreis berechnet sich ähnlich, er enthält die Summe aus dem mit der Menge multiplizierten Nettopreis und der Mehrwertsteuer:

```
Bruttopreis: [Nettopreis]*[Menge]*(1+[tblBestellpositionen].[Mehrwertsteuersatz])
```

Schließlich soll die im Feld *Rechnungsanrede* gespeicherte Floskel (zum Beispiel *Sehr geehrter Herr [Rechnung_Nachname]* noch mit dem richtigen Nachnamen gefüllt werden.

Dies erledigt die *Replace*-Funktion, die als ersten Parameter den Originalausdruck, als zweiten den zu ersetzen Ausdruck und als dritten den Ersatzausdruck enthält:

```
Briefanrede_Rechnung: Ersetzen([Briefanrede];"[Rechnung_Nachname]";[Rechnung_Nachname])
```

Das Ergebnis der Abfrage sieht wie in Abbildung 6.2 aus.

Datenherkunft zuweisen

Zeigen Sie nun den Bericht *rptRechnung* in der Entwurfsansicht an. Fügen Sie der Eigenschaft *Datenherkunft* beziehungsweise *Datensatzquelle* den Wert *qryRechnung* hinzu. Aktivieren Sie mit *Entwurf|Tools|Vorhandene Felder* hinzufügen die Feldliste (siehe Abbildung 6.3).

Wie verteilen Sie nun die Felder auf die drei angezeigten Bereiche *Seitenkopf*, *Detailbereich* und *Seitenfuß*? Sind dies überhaupt die richtigen Bereiche oder benötigen wir noch weitere?

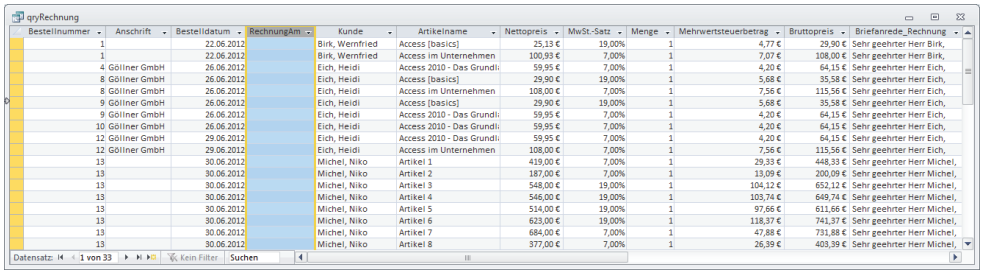


Abbildung 6.2: Datenblattansicht der Abfrage, die als Datenherkunft des Berichts dient

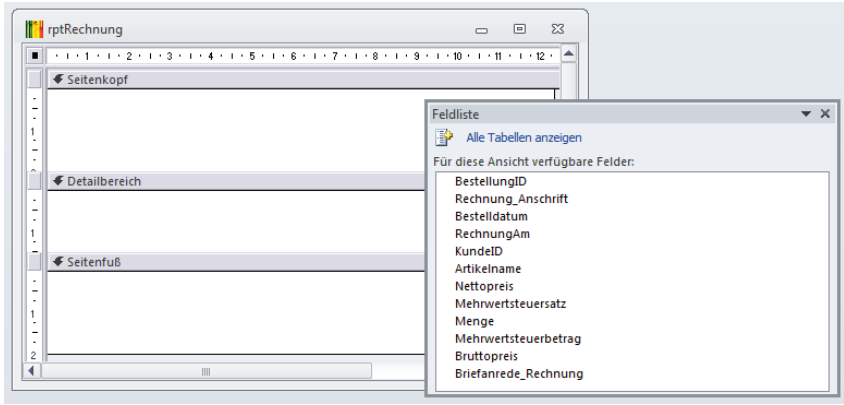


Abbildung 6.3: Der Bericht nach dem Anlegen und Hinzufügen der Datenherkunft

Rekapitulieren wir, welcher Bereich wie angezeigt werden kann. Der Seitenkopf erscheint oben auf jeder Seite, außer Sie ändern die Eigenschaft *Seitenkopf* des Berichts auf einen der Werte *Außer Berichtskopf*, *Außer Berichtsfuß* oder *Außer Berichtskopf/-fuß*.

Wenn der Seitenkopf beispielsweise nicht auf der ersten Seite erscheinen soll, stellen Sie die Eigenschaft also auf *Außer Berichtskopf* ein. Für den Bereich *Seitenfuß* gilt das Gleiche: Sie können ihn für die Seite mit dem Berichtskopf, mit dem Berichtsfuß oder beide ausblenden.

Interessant ist hierbei Folgendes: *Berichtskopf* wird synonym zu *Erste Seite* behandelt. Selbst wenn der Bericht gar keinen *Berichtskopf*-Bereich enthält, führt die Einstellung *Außer Berichtskopf* für die Eigenschaft *Seitenkopf* dazu, dass der Seitenkopf auf der ersten Seite nicht angezeigt wird. *Berichtsfuß* hingegen ist nicht mit *Letzte Seite* zu verwechseln: Der Seitenfuß wird auf der letzten Seite nur ausgeblendet, wenn der Bericht tatsächlich einen Berichtsfuß enthält.

Außerdem ist die Position wichtig. Der Berichtskopf befindet sich über dem Seitenkopf, der Berichtsfuß aber vor dem Seitenfuß – sofern diese auf einer Seite erscheinen. Dies hängt damit zusammen, dass der Seitenfuß zuverlässig immer ganz unten auf der Seite abgebildet wird (siehe Abbildung 6.4). Wenn Sie sehen möchten, wie Sie die Texte anordnen müssen, um diesen

Bericht zu erhalten, schauen Sie sich den Bericht *rptRechnung_Aufbau* an – er wird allerdings im weiteren Verlauf nicht mehr benötigt.

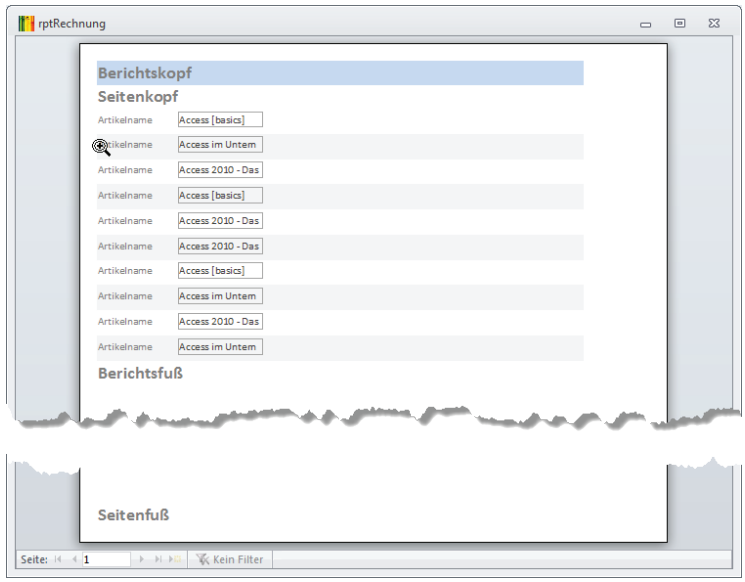


Abbildung 6.4: Struktur eines Berichts

Nicht zu vergessen: der Detailbereich. Er wird für jeden Datensatz der Datenherkunft je einmal abgebildet. Und welche Daten einer Rechnung sollen überhaupt wiederholt werden? Richtig: die Rechnungspositionen.

6.2.2 Bericht mit Steuerelementen füllen

Fügen wir also nun die Steuerelemente zum Bericht hinzu – erstmal ganz grob, um den grundsätzlichen Aufbau zu skizzieren. Zunächst einmal sollen die sich wiederholenden Felder, also alle Informationen rund um die Bestellposition, im Detailbereich landen.

Dazu ziehen Sie die entsprechenden Felder aus der Feldliste in diesen Bereich (siehe Abbildung 6.5).

So sollen die Rechnungspositionen allerdings nicht im Bericht erscheinen, sondern nebeneinander – mit den Überschriften über den Feldern. Jetzt kommt die erste wichtige Überlegung: Wenn die Überschriften nur einmal ausgegeben werden sollen, die Details der Rechnungspositionen jedoch für jeden Datensatz einmal – wo landen dann die Überschriften? Richtig: Sie landen zunächst einmal im Seitenkopf.

Seit Access 2007 ist das Arrangieren der Steuerelemente auf diese Art zum Glück sehr einfach geworden – vorher war es eine friemelige Handarbeit. Markieren Sie alle soeben hinzugefüg-

ten Textfelder und wählen Sie dann den Ribbon-Befehl *Anordnen|Tabelle|Tabelle* aus (siehe Abbildung 6.6). Dies ordnet die markierten Elemente wie in Abbildung 6.7 an.

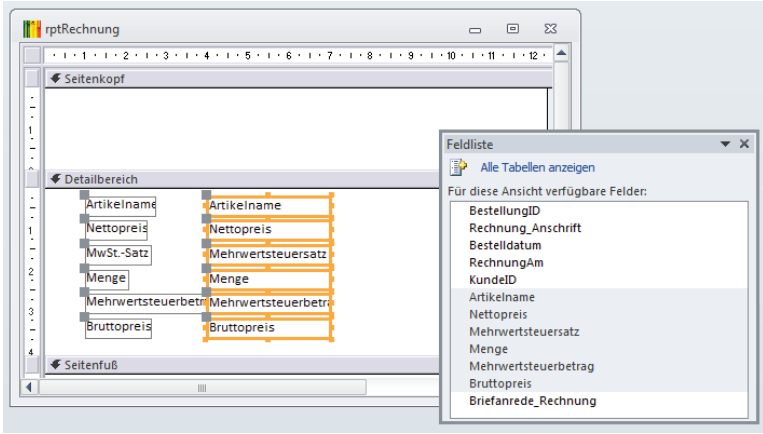


Abbildung 6.5: Hinzufügen der Informationen einer Rechnungsposition

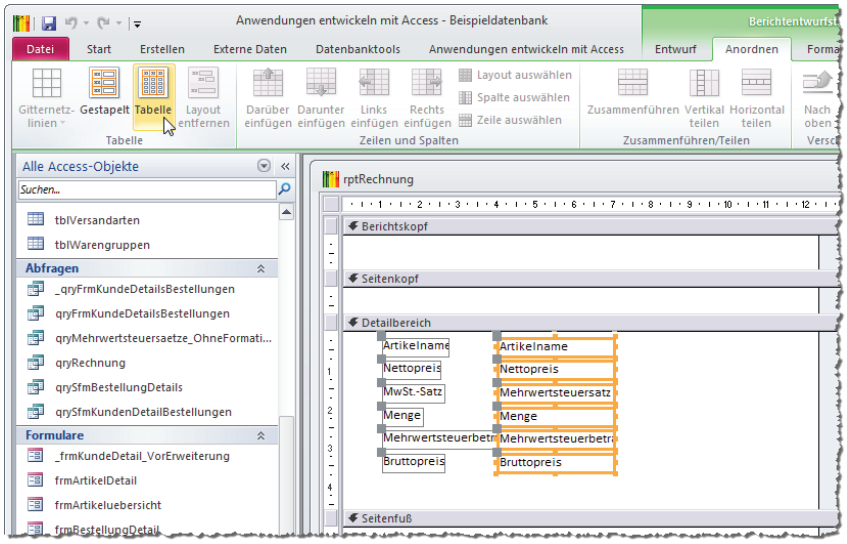


Abbildung 6.6: Markieren und Ausrichten der Bestellpositionen

Danach nutzen Sie ein weiteres Feature, das es in Access 2003 noch nicht gab – die Layoutansicht. Klicken Sie mit der rechten Maustaste auf die Titelleiste des Berichts und wählen Sie dort den Eintrag *Layoutansicht* aus. Dies öffnet den Bericht in einer weiteren Ansicht, die sowohl alle durch die Datenherkunft gelieferten Daten anzeigt als auch das Anpassen der Größe der Steuerelemente ermöglicht (siehe Abbildung 6.8).

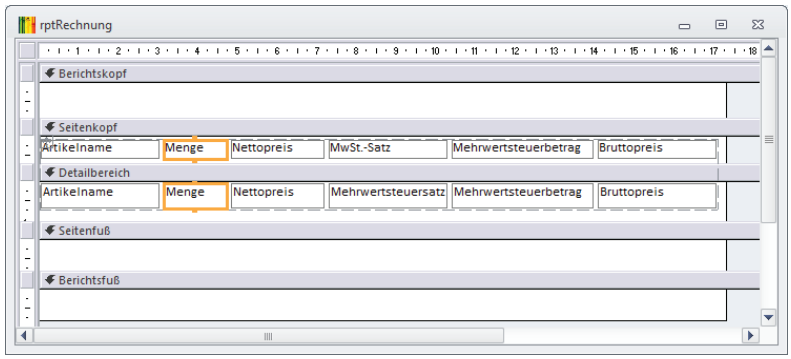


Abbildung 6.7: Ordentlich angeordnete Steuerelemente

Zu diesem Zeitpunkt sollten Sie bereits einige Datensätze angelegt haben, ansonsten holen Sie dies nach (die Beispieldatenbank enthält natürlich bereits einige Bestellungen).

Textfelder im Detailbereich anpassen

Nun können Sie die Breite der Spalten des Detailbereichs in der Layoutansicht so anpassen, dass die bisher verfügbaren Daten hineinpassen. Bei den meisten Steuerelementen lässt sich der benötigte Platz gut abschätzen – so sollten Sie bereits wissen, wie viele Zahlenstellen die Preise für Ihre Artikel einnehmen (ich hoffe, viele!). Auch der benötigte Platz für die Menge, der Steuersatz und der Bruttopreis sollten sich gut abschätzen lassen.

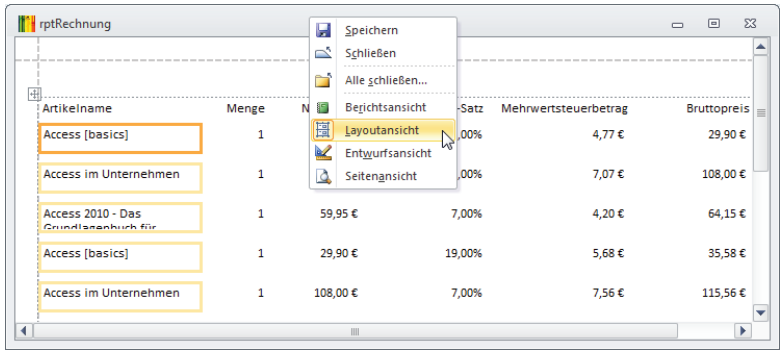


Abbildung 6.8: Anzeigen der Layoutansicht eines Berichts

Bringen Sie also die Spalten in die richtige Breite. Wenn Sie die Steuerelemente wie oben angegeben zu einem Layout zusammengefasst haben, stellt Access die Breite von Bezeichnungsfeld und Textfeld gleichzeitig ein, wenn Sie eines der beiden Elemente ändern.

Dazu klicken Sie entweder auf das Bezeichnungs- oder das Textfeld und ziehen das Feld am rechten oder unteren Rand kleiner oder größer.

Passen Sie gegebenenfalls die Spaltenüberschriften nach Wunsch an. Außerdem können Sie in einem Layout, wie Sie es angelegt haben, auch komplette Spalten verschieben, ohne das übrige Layout zu zerstören. Abbildung 6.9 zeigt, wie dies aussieht: Klicken Sie beispielsweise mitten in das Bezeichnungsfeld *Menge*, sodass eine Art Fadenkreuz erscheint. Halten Sie die *Umschalt*-Taste gedrückt und markieren Sie auch das *Textfeld* darunter. Beide sollen direkt hinter den Artikelnamen verschoben werden, also ziehen Sie die Markierung so weit nach links, bis zwischen den Feldern *Artikelname* und *Einzelpreis* eine Einfügemarke erscheint. Lassen Sie das Feld fallen und freuen Sie sich, dass Access die übrigen Felder einfach nach links verschiebt, ohne das Layout zu zerstören (in der Entwurfsansicht markieren Sie die Felder durch einen Klick auf den Steuerelement-Rahmen).

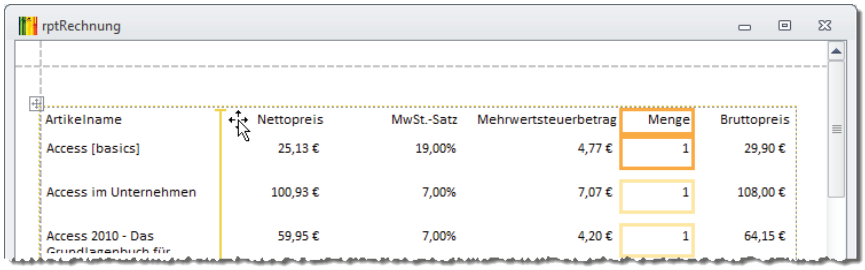


Abbildung 6.9: Verschieben eines Feldes in einem Tabellen-Layout

Die Layoutansicht ermöglicht es auch, die enthaltenen Steuerelemente auf die verfügbare Breite zu verteilen. Die gestrichelten Linien zeigen die Seitenränder an.

Tipp: Die Reihenfolge beim Erstellen eines Layouts richtet sich nach dem Wert der Eigenschaft *Aktivierreihenfolge* der Steuerelemente, die wiederum durch die Reihenfolge des Hinzufügens in den Bericht beeinflusst wird. Sie könnten also direkt in der Abfrage die gewünschte Reihenfolge der Spalten einstellen und brauchen die Felder dann nur noch von der Feldliste in den Entwurf des Berichts zu ziehen.

Mehrzeilige Artikelbezeichnungen

Das einzige Feld, das nun nach dem Hinzufügen neuer Artikel nicht ausreichend Platz bieten könnte, ist das Feld *Artikelname*. Damit zu lange Texte hier nicht abgeschnitten werden, können Sie die Inhalte auch mehrzeilig darstellen (siehe Abbildung 6.10). Legen Sie für die Eigenschaft *Vergrößerbar* des Textfeldes *Artikelname* den Wert *Ja* fest und stellen Sie sicher, dass die gleichnamige Eigenschaft des Detailbereichs ebenfalls diesen Wert enthält.

6.2.3 Überschriften verschieben

Sicher haben Sie bemerkt, dass das Anwenden des Layouts die Bezeichnungsfelder als Überschriften in den Bereich *Seitenkopf* verschoben hat. Das ist kein Zufall: Access verschiebt die Überschriften beim Layout *Tabelle* immer in den Bereich, der sich über den markierten Steuer-

elementen befindet – in diesem Fall war es der Bereich *Seitenkopf*. Wenn Sie den *Seitenkopf*-Bereich ausblenden, würde das Anwenden des Layouts *Tabelle* die Spaltenüberschriften in den Bereich *Berichtskopf* verschieben.

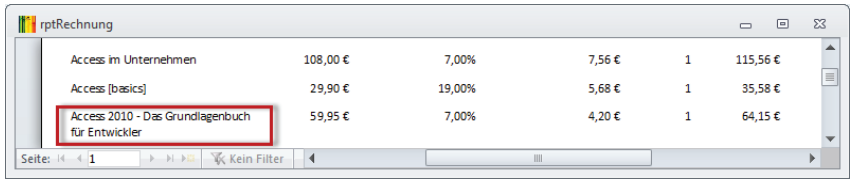


Abbildung 6.10: Mehrzeilige Felder im Bericht

Wohin mit den Überschriften?

In welchem Bereich aber sollen die Überschriften nun landen? Der Berichtskopf wird nur auf der ersten Seite angezeigt. Der Seitenkopf wird auf jeder Seite angezeigt, außer Sie legen fest, dass er nicht auf der Seite des Berichtskopfes oder des Berichtsfußes erscheinen soll. Wenn wir also festlegen, dass der Seitenkopf erst ab der zweiten Seite erscheint, könnten wir die Überschriften einmal im Berichtskopf unterbringen (für die erste Seite) und, falls die Rechnung viele Positionen enthält, im Seitenkopf.

Das heißt also, dass wir die kompletten Überschriften gleich in zwei Bereichen unterbringen – was bei Änderungen natürlich doppelten Aufwand bedeutet. Zum Glück gibt es jedoch noch weitere Bereiche, nämlich die Kopf- und Fußbereiche von Gruppierungsebenen. Mit einer Gruppierung fassen Sie mehrere Datensätze mit gleichen Eigenschaften zusammen.

Die Datenherkunft für den Bericht *rptRechnung*, also die Abfrage *qryRechnung*, liefert ja im Originalzustand alle Bestellungen für alle Kunden mit allen Bestellpositionen. Normalerweise wird dieser Bericht mit einem Parameter geöffnet, der nur die Datensätze für eine einzige Bestellung anzeigt und dazu einen Filter wie *BestellungID = 12* einstellt – beispielsweise so:

```
DoCmd.OpenReport "rptRechnung", View:=acViewPreview, WhereCondition:="BestellungID = 12"
```

Würden wir dies weglassen, würde der Bericht jedoch die Bestellpositionen für alle Bestellungen anzeigen. Hier könnten Sie nun eine neue Gruppierung nach dem Feld *BestellungID* anlegen und dieser einen Gruppenkopf hinzufügen (wie das im Detail funktioniert, erfahren Sie gleich). Wenn Sie dort das Feld *BestellungID* (und die Spaltenüberschriften) einfügen, sähe der Bericht in der Vorschau wie in Abbildung 6.11 aus. Der Clou für unsere aktuelle Aufgabe ist, dass der Gruppenkopfbereich (genau wie der Gruppenfußbereich) eine Eigenschaft namens *Bereich wiederholen* anbietet, die mit der Einstellung *Ja* dafür sorgt, dass der Bereich zu Beginn der folgenden Seite wiederholt wird, wenn auf der aktuellen Seite nicht alle Detailbereiche der Gruppe Platz finden. Das bedeutet, dass wir die Spaltenüberschriften nicht sowohl im Berichtskopf-Bereich als auch im Seitenkopf-Bereich unterbringen müssen, um Spaltenköpfe auf jeder Seite anzuzeigen – wir können dies auch mit dem Kopfbereich einer Gruppierung erreichen.

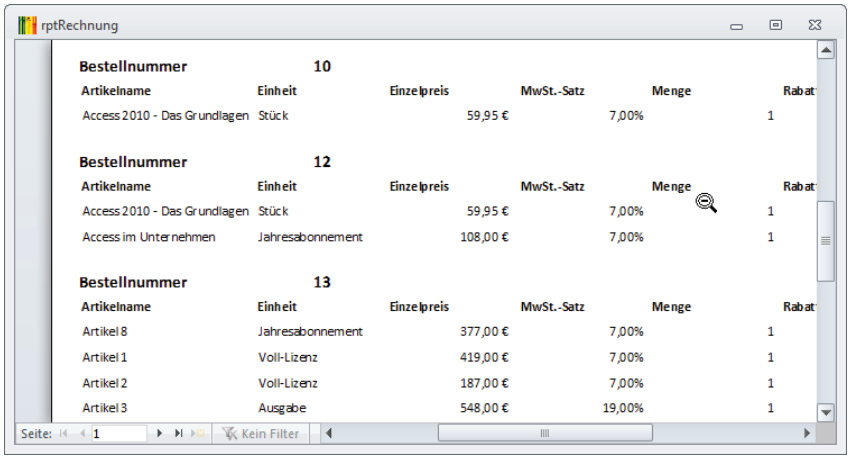


Abbildung 6.11: Effekte einer Gruppierung

Gruppierung nach *BestellungID* anlegen

Dazu legen wir nun eine Gruppierung für den Bericht an. Um den Bereich der Benutzeroberfläche anzuzeigen, der dies ermöglicht, wählen Sie den Eintrag *Entwurf|Gruppierung und Summen|Gruppieren und Sortieren* des Ribbons aus.

Im unteren Bereich erscheint der (leider nicht verschiebbare) Bereich *Gruppieren, Sortieren und Summe*. Klicken Sie hier auf *Gruppe hinzufügen* (siehe Abbildung 6.12).

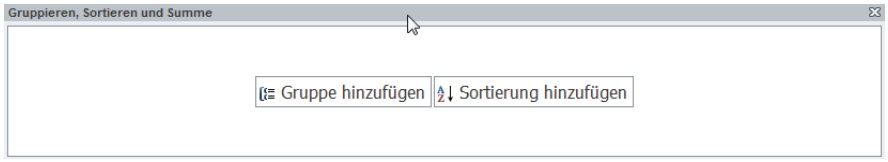


Abbildung 6.12: Anlegen einer Gruppierung

Wählen Sie unter *Gruppieren nach* das Feld *BestellungID* aus und klicken Sie dann auf *Mehr*, um die übrigen Eigenschaften zu sehen. Hier brauchen wir nichts mehr zu erledigen, da dort bereits die Option *mit Kopfzeilenbereich* ausgewählt ist (siehe Abbildung 6.13).

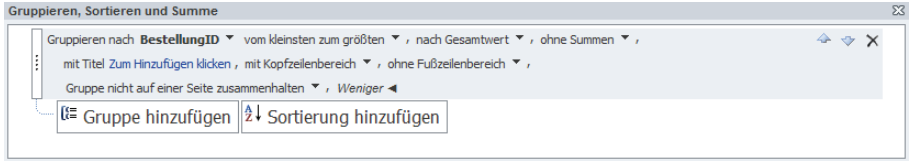


Abbildung 6.13: Eigenschaften einer Gruppierung

Nun haben Sie zwar einen Gruppenkopf erstellt, der die Spaltenüberschriften für die Rechnungspositionen aufnehmen soll, aber die benötigten Bezeichnungsfelder befinden sich noch im falschen Bereich (siehe Abbildung 6.14).

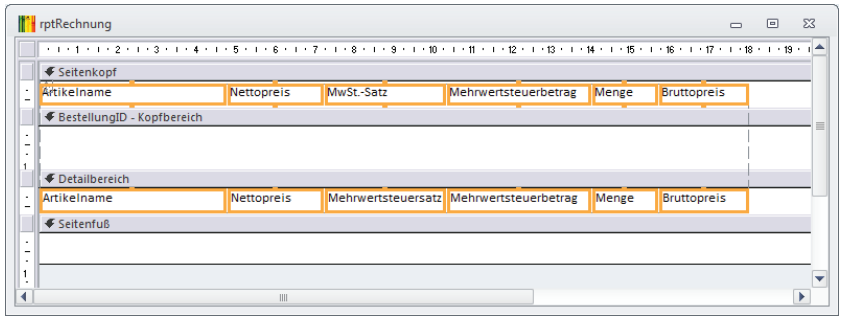


Abbildung 6.14: Rechnungsbericht mit Gruppenkopf für die Spaltenüberschriften

Um die Bezeichnungsfelder aus dem Seitenkopf in den Gruppenkopf zu verschieben, kommen Sie nicht umhin, das Layout zu zerstören – es gibt schlicht keinen Weg, die Bezeichnungsfelder in einem Rutsch zu verschieben. Also markieren Sie alle Spaltenüberschriften, betätigen die Tastenkombination *Strg + X*, um diese auszuschneiden, markieren dann den Bereichskopf der neu angelegten Gruppierung und drücken dann auf *Strg + C*. Sie können das Layout aber dann wieder herstellen, indem Sie alle Bezeichnungsfelder und Textfelder markieren und nacheinander die Kontextmenü-Befehle *Layout|Layout entfernen* und *Layout|Tabelle* auswählen. Allein einige Spaltenbreiten zerschießen Sie mit dieser Vorgehensweise, aber die sind schnell wiederhergestellt. Achten Sie darauf, die Höhe des Detailbereichs durch Verschieben des Seitenfußes nach oben an Ihre Bedürfnisse anzupassen – beim Bearbeiten des Detailbereichs wird dieser schnell zu groß und enthält in der Vorschau und beim Drucken unnötige Leerräume. Damit sind die Arbeiten am Detailbereich vorerst beendet – wenden wir uns nun den übrigen Bereichen zu.

6.2.4 Berichtskopf mit Briefkopf und Anschrift ausstatten

Der Briefkopf und die Anschrift sollen nur einmal je Rechnung angezeigt werden. Der Berichtskopf-Bereich ist genau der richtige Bereich für diesen Zweck. Also ziehen Sie den Berichtskopf auf eine entsprechende Höhe – wie hoch er wird, liegt an den Inhaltselementen, die Sie benötigen. Es sollte Platz sein für folgende Elemente:

» Briefkopf

» Empfängeranschrift (die sich in einem festen Bereich befinden muss, damit sie in den Sichtfenstern herkömmlicher Umschläge angezeigt werden kann)

» Ort/Datum

» Rechnungsnummer

» Rechnungstext

» gegebenenfalls ein grafisches Element wie etwa ein Firmen-Logo

Wie hoch der Bereich letztendlich wird, hängt von den gewählten Elementen ab. Unser Berichtskopf sieht wie in Abbildung 6.15 aus. Es gibt dort statische und dynamische Elemente, also Bezeichnungsfelder, die immer den gleichen Text anzeigen, und Textfelder, die erst beim Anzeigen mit den Texten aus der Datenherkunft gefüllt werden.

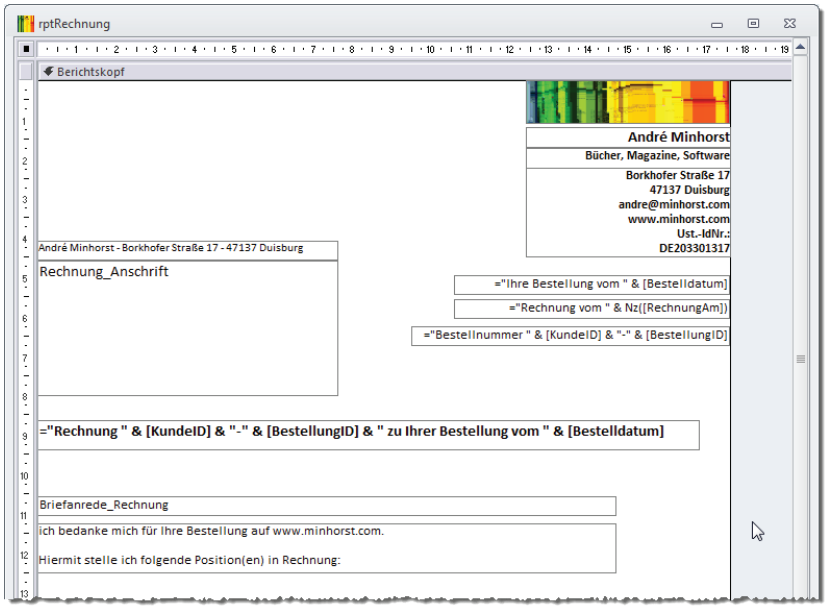


Abbildung 6.15: Entwurf des Berichtskopfes

Ein Blick auf die Entwurfsansicht verrät bereits, dass einige der Steuerelemente (es sind alles Textfelder) aus den Feldern der Datenherkunft gefüllt werden.

Dabei handelt es sich zunächst um die Anschrift, die ihre Daten aus dem Feld *Rechnung_Anschrift* der Abfrage *qryRechnung* bezieht. Dieses Feld hält die Anschrift ja direkt als Block aus mehreren Zeilen bereit, der im Formular *frmKundeDetail* auf Basis der Rechnungsadresse angepasst werden kann. Auch das Feld zur Anzeige des Bestelldatums ist an die Datenherkunft gebunden, allerdings nicht direkt.

Das heißt, dass die Eigenschaft *Steuerelementinhalt* nicht den Namen eines einzigen Feldes enthält (hier *Bestelldatum*), sondern dass stattdessen ein Ausdruck mit führendem Gleichheitszeichen angegeben wurde, der den Feldinhalt beinhaltet. Dieser Ausdruck setzt sich aus einem Literal und dem Feldnamen zusammen:

```
="Ihre Bestellung vom " & [Bestelldatum]
```

Ähnlich verhält es sich beim Steuerelement zur Anzeige des Rechnungsdatums, das diesen Ausdruck enthält:

```
="Rechnung vom " & Nz([RechnungAm])
```

Darunter befindet sich noch die Bestellnummer, die gleich zwei Felder der Datenherkunft und ein Literal zusammenführt:

```
="Bestellnummer " & [KundeID] & "-" & [BestellungID]
```

Wenn Sie hier noch etwa das Rechnungsjahr integrieren möchten, können Sie den Zusatz *Jahr(Bestelldatum)* mit den entsprechenden &-Zeichen an der gewünschten Stelle hinzufügen.

6.2.5 Berichtsfuß mit Bankverbindung und Co. versehen

Der Berichtsfuß soll Informationen wie beispielsweise die Bankverbindung für die Überweisung des Rechnungsbetrags und eine Grußformel enthalten. Diese fügen Sie in einfache Bezeichnungsfelder ein. Wenn Sie verschiedene Schriftarten benötigen, um beispielsweise eine Zeile mit dem Hinweis auf den zu vermerkenden Verwendungszweck hervorzuheben, verwenden Sie mehrere Bezeichnungsfelder mit unterschiedlichen Formatierungen (siehe Abbildung 6.16).

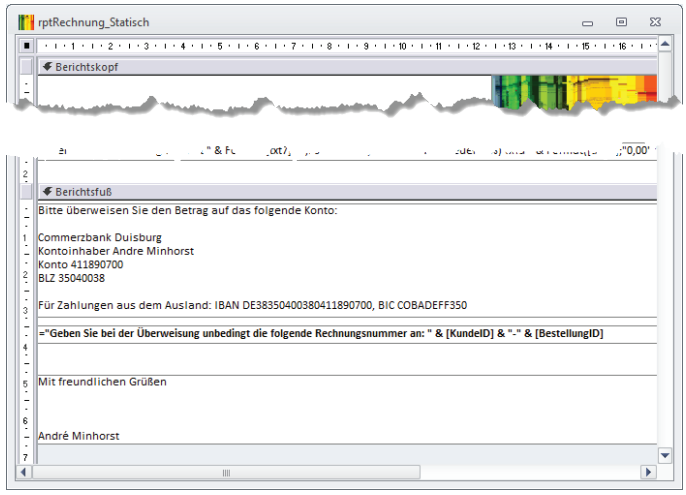


Abbildung 6.16: Statischer Gruppenfuß mit mehreren Bezeichnungsfeldern

6.2.6 Gruppenfuß mit Berichtssummen füllen

In der Beispieldatenbank gibt es verschiedene Steuersätze, die auch in einer Bestellung erfasst werden können. Folglich sollten diese auch in einer Rechnung landen. Interessant wird es hier bei der Summe der Umsatzsteuer: Muss man die Beträge für die verschiedenen Steuersätze aufführen? Das wäre doch etwas aufwendig, zumindest wenn man die Anwendung auch noch so

7 Rechnungen verwalten

Wie Sie einen Bericht zur Ausgabe von Rechnungen erstellen, haben Sie ja bereits in »Rechnungsbericht« (Seite 191) erfahren. Noch interessanter ist jedoch, zu welchem Zeitpunkt und von welchen Formularen aus die Rechnungen erstellt werden, wie sie zum Kunden gelangen (in Briefform oder als E-Mail mit angehängtem PDF), wie Sie den Eingang prüfen und wie Sie Erinnerungen oder Mahnungen versenden.

7.1 Grundsätzliche Überlegungen

Dementsprechend betrachten wir im ersten Teil dieses Kapitels grundlegend, wie der Rechnungsbericht in die Anwendung integriert wird und wie Sie die Rechnungen und Eingänge verwalten.

Dabei gibt es viele Varianten. Beginnen wir mit der Rechnung: Diese kann, wie es im Kapitel »Rechnungsbericht« (Seite 191) geschehen ist, einfach auf der Tabelle *tblBestellungen* basieren. Das heißt, dass es eine Rechnung je Bestellung gibt – unabhängig davon, ob alle Artikel lieferbar sind oder der Kunde Artikel retourniert. Eine weitere Variante ist, dass eine Rechnung nur einige der Bestellpositionen einer Bestellung aufnimmt und andere in einer zweiten Rechnung landen, weil nicht alle Artikel gleich lieferbar sind. Eine weitere Alternative ist es, mehrere Bestellungen in einer Rechnung zu erfassen. Auf diese Weise könnten Kunden mehrere Bestellungen tätigen und würden beispielsweise einmal im Monat eine Rechnung erhalten. Oder, um die Flexibilität auf die Spitze zu treiben: Sie fügen Bestellpositionen aus mehreren Bestellungen in einer Rechnung zusammen.

Die erstgenannte Methode ist die einzige, für die Sie keine eigene Rechnungen-Tabelle anlegen müssen. Dort werden die Rechnungsinformationen wie etwa das Rechnungsdatum oder der Zahlungseingang direkt in die Tabelle *tblBestellungen* eingetragen. Sobald Sie für eine Bestellung jedoch mehrere Rechnungen schreiben, benötigen Sie eine Rechnungen-Tabelle, welche jeweils einige Bestellpositionen in einer Rechnung erfasst.

7.1.1 Änderungen im Datenmodell

In der Beispielanwendung beschreiben wir die einfache Variante mit einer Rechnung pro Bestellung. Das bedeutet für das Datenmodell, dass wir der Tabelle *tblBestellungen* einige Felder hinzufügen müssen:

» *Rechnungsdatum* (*Datum/Uhrzeit*): Nimmt das Datum der Rechnungsstellung/des Rechnungsversands auf

» *Zahlungsziel* (*Datum/Uhrzeit*): Datum des Zahlungsziels

» *BezahltAm* (*Datum/Uhrzeit*): Zeitpunkt des Geldeingangs

» *RechnungsversandartID* (*Nachschlagefeld*): Fremdschlüsselfeld zur Auswahl einer Versandart aus der Tabelle *tblVersandarten* (siehe Abbildung 7.1)

» *Rechnungsdatei* (*Text*): Name der Rechnungsdatei

» *StorniertAm* (*Datum/Uhrzeit*): Zeitpunkt der Stornierung einer Rechnung

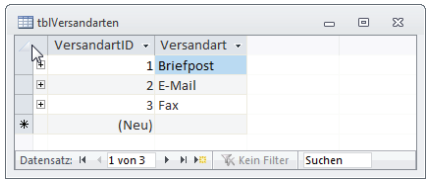


Abbildung 7.1: Die Tabelle *tblVersandarten*

7.1.2 Zeitpunkt und Ort der Rechnungserstellung

Die erste Frage ist: Wann soll die Rechnung überhaupt erstellt werden? Immerhin gehen wir davon aus, dass Bestellungen auf verschiedenen Wegen ankommen – telefonisch, per Fax, per E-Mail oder per Internet-Shop beziehungsweise Bestellformular. Bei der Internet-Variante nehmen wir an, dass die Shop-Software keine Rechnung erstellt, sondern nur die Bestellung entgegennimmt – alle Rechnungen sollen also zentral von unserer Anwendung aus erstellt werden.

Dort haben wir ein Formular, in das wir die Bestellungen manuell eingeben (*frmBestellungDetail*) und eine Übersicht der Bestellungen eines Kunden im Formular *frmKundeDetail*. Beide können wir zwar mit einer Schaltfläche zum Erstellen einer Rechnung bestücken, aber das Hauptwerkzeug zum Erstellen von Rechnungen und zum Verfolgen der Rechnungseingänge soll ein eigenes Formular sein, das alle Bestellungen inklusive Datum der Rechnungserstellung, Eingang des Rechnungsbetrages, versendete Mahnungen et cetera anzeigt.

Dieses Formular soll auch die Möglichkeit bieten, die Bestelldetails zu einer Bestellung oder den Kunden anzuzeigen, die Rechnung als PDF zu öffnen, nach verschiedenen Kriterien zu filtern (etwa nach offenen Rechnungen oder Rechnungserinnerungen) oder zu sortieren (nach Rechnungsdatum oder Geldeingang) und mehr.

7.1.3 Rechnung per Detailformular erstellen

Zunächst fügen wir jedoch dem Formular *frmBestellungDetail* zwei Schaltflächen hinzu, mit denen die Rechnung entweder gedruckt oder per E-Mail versendet werden kann. Außerdem soll eine weitere Schaltfläche einfach nur die Rechnung entsprechend den aktuellen Daten anzeigen (siehe Abbildung 7.2).

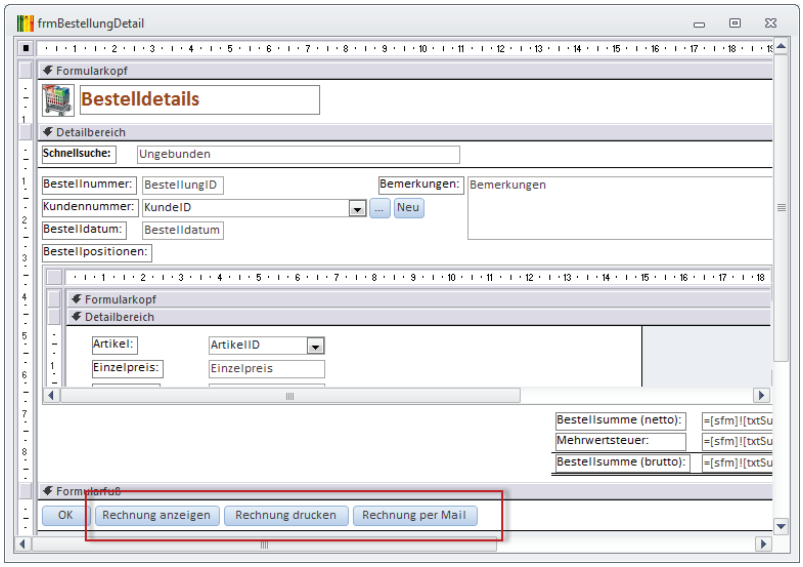


Abbildung 7.2: Schaltflächen zum Anzeigen, Drucken und Versenden der Rechnungen

Die erste Schaltfläche heißt *cmdRechnungAnzeigen*. Fügen Sie dieser Schaltfläche eine Ereignisprozedur hinzu, indem Sie die Eigenschaft *Beim Klicken* auf *[Ereignisprozedur]* einstellen und auf die Schaltfläche mit den drei Punkten klicken. Im VBA-Editor ergänzen Sie die dadurch erzeugte Prozedur wie folgt:

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

Die Prozedur zeigt den Rechnungsbericht für diese Bestellung an. Weiter passiert nichts – die Rechnung wird nicht gedruckt und auch nicht im PDF-Format gespeichert.

Dies soll erst geschehen, wenn der Benutzer auf eine der beiden anderen Schaltflächen klickt. Die Schaltfläche *cmdRechnungDrucken* etwa soll den Bericht zu Dokumentationszwecken speichern und diesen auch gleich drucken. Die Schaltfläche *cmdRechnungPerMail* speichert den Bericht ebenfalls, verschickt diesen dann aber als Anhang einer E-Mail.

7.1.4 Rechnung drucken

Egal, ob die Rechnung gedruckt und per Post versendet oder im PDF-Format per E-Mail verschickt werden soll: Sie muss auf jeden Fall im PDF-Format gespeichert werden. Außerdem soll das Feld *RechnungAm* mit dem Datum der Rechnungserstellung gefüllt werden und das Feld *Rechnungsdatei* soll den Dateinamen der Rechnung erhalten. Der Teil, den die Prozedur dazu

beiträgt, die durch einen Mauslick auf die Schaltfläche *cmdRechnungDrucken* ausgelöst wird, sieht so aus:

```
Private Sub cmdRechnungDrucken_Click()
    Me!Rechnungsdatei = RechnungErstellen(Me!BestellungID)
    Me!RechnungAm = Now
       DoCmd.OpenReport "rptRechnung", acViewNormal, _
           WhereCondition:="BestellungID = " & BestellungID
End Sub
```

7.1.5 Rechnung als PDF speichern

Hier verbirgt sich natürlich der Aufruf einer weiteren Funktion, nämlich *RechnungErstellen*. Diese bringen wir in einem neuen Standardmodul unter, das Sie im VBA-Editor mit dem Menübefehl *Einfügen|Standardmodul* anlegen. Speichern Sie dieses gleich unter dem Namen *mdlRechnungen*.

Warum soll die Prozedur in einem eigenen Modul landen und nicht im Klassenmodul des Formulars? Weil wir noch von anderen Stellen aus Rechnungen erstellen werden, zum Beispiel vom Formular mit der Rechnungsübersicht. Also fügen wir die notwendigen Anweisungen in eine eigene Prozedur ein und bringen diese an einer Stelle unter, auf welche die in den anderen Modulen befindlichen Prozeduren einfach zugreifen können.

Die Prozedur braucht nur eine Information, nämlich den Primärschlüsselwert des Bestelldatensatzes. Diese übergeben Sie mit dem Parameter *lngBestellungID*. Die Prozedur deklariert und füllt eine Variable namens *strBericht* mit dem Namen des Berichts. Das ist reine Faulheit, denn dieser Name wird im Folgenden gleich dreimal benutzt. Sollte sich der Bericht mal ändern, braucht man diesen so nur an einer statt an drei Stellen zu ändern.

Die zweite Variable *strDateipfad* wird mit dem Ergebnis der Funktion *DateipfadErmitteln* aus dem Modul *mdlTools* gefüllt. Dies ist eine Funktion, die wiederum von unterschiedlichen Stellen aus aufgerufen wird und sich von verschiedenen Orten das Verzeichnis und den Dateinamen zusammensucht, die enthaltenen Platzhalter ersetzt und daraus den Dateipfad erstellt – mehr dazu im Anschluss an die Rechnungserstellung. Mit dem Dateipfad im Gepäck folgen nun drei Schritte:

» Öffnen des Berichts für die angegebene Bestellung in der Vorschauansicht, allerdings im nicht sichtbaren Modus (*WindowMode:=acHidden*)

» Exportieren beziehungsweise Speichern des Berichts mit der *Output*-Methode des *DoCmd*-Objekts. Der erste Parameter gibt die Objektart an, der zweite den Namen des zu exportierenden Berichts, der dritte das Export-Format und der vierte den Namen der Zieldatei. Die Ermittlung der Pfad- und Dateiangaben in der Funktion basieren auf den Angaben in einigen Feldern der Tabelle *tblOptionen*, die weiter unten vorgestellt werden.

» Schließen des Berichts

Trotz des verborgenen Öffnens des Berichts flackert der Bildschirm bei diesem Vorgang ein wenig, aber das soll uns nicht weiter stören. Schließlich wird der Name der erstellten Datei als Rückgabewert der Funktion festgelegt:

```
Public Function RechnungErstellen(lngBestellungID As Long) As String
    Dim strDateipfad As String
    Dim strBericht As String
    strBericht = "rptRechnung"
    strDateipfad = DateipfadErmitteln("tblBestellungen", "BestellungID", _
        lngBestellungID, "VerzeichnisRechnungsdateien", "DateinameRechnungsdateien")
    DoCmd.OpenReport strBericht, WhereCondition:="BestellungID = " & lngBestellungID, _
        View:=acViewPreview, WindowMode:=acHidden
    DoCmd.OutputTo acOutputReport, strBericht, "PDF", strDateipfad
    DoCmd.Close acReport, strBericht
    RechnungErstellen = strDateipfad
End Function
```

Wenn Sie noch die nachfolgende beschriebene Funktion *DateipfadErmitteln* erstellt haben (siehe Beispieldatenbank, Modul *mdlTools*), war es das schon – die Rechnung befindet sich im angegebenen Verzeichnis (siehe Abbildung 7.3). Und auch die PDF-Dateien des Rechnungsberichts sehen genau wie erwartet aus (siehe Abbildung 7.4).

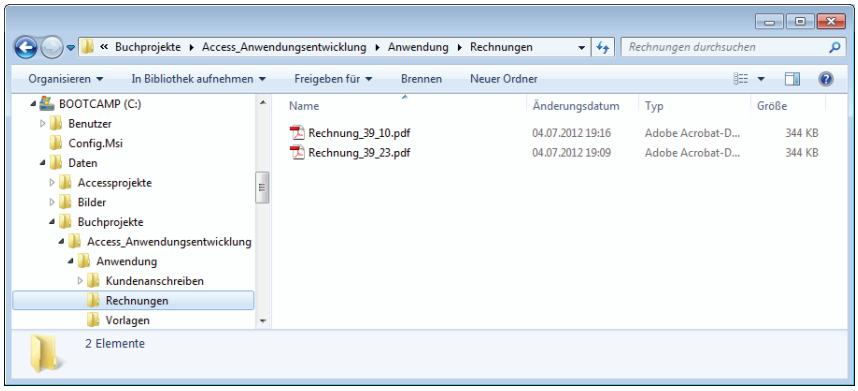


Abbildung 7.3: Verzeichnisstruktur mit Beispielrechnungen

Ermitteln des Dateipfades für die zu erstellende PDF-Datei

Für die Dateipfade, also für Verzeichnisse und Dateinamen, soll die Anwendung bestimmte Standardwerte bereithalten, die der Anwender nach Bedarf anpassen kann. Diese Einstellungen werden im Dialog *frmOptionen* gespeichert, der wie in Abbildung 7.5 aussieht. Das Formular enthält Daten aus der Tabelle *tblOptionen* (die Feldbeschreibungen finden Sie im Entwurf der Tabelle in der Beispieldatenbank).

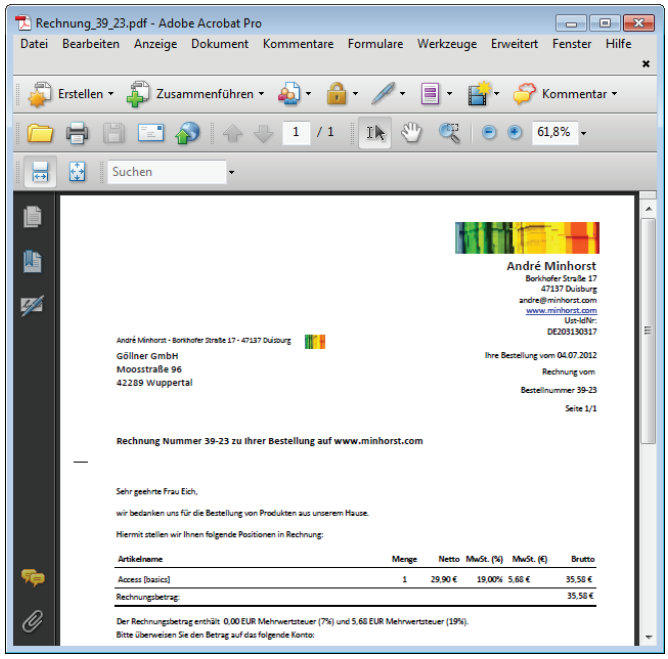


Abbildung 7.4: Ein Access-Bericht im PDF-Format

Der Standardwert für das Verzeichnis der Rechnungsdateien steht im Feld *VerzeichnisRechnungsdateien* der Tabelle *tblOptionen*, der Standardwert für den Dateinamen im Feld *DateinameRechnungsdateien*. Die in der Beispieldatenbank für die Felder hinterlegten Werte *[Backend]\Rechnungen* und *Rechnung_[KundeID]_[BestellungID].pdf* enthalten bereits einige Platzhalter.

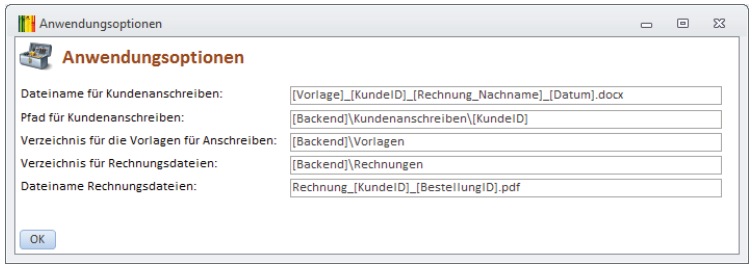


Abbildung 7.5: Angabe des Pfades für die Rechnungsdateien

Die Funktion *DateipfadErmitteln* aus dem Modul *mdlTools* macht aus diesen Vorlagen einen richtigen Dateipfad, indem sie die Werte der Tabelle *tblOptionen* einliest und die Platzhalter ersetzt. Nun gibt es nicht nur Platzhalter wie *[Backend]*, der durch das Verzeichnis des Backends ersetzt werden soll, sondern auch Feldnamen aus Tabellen. Um wie hier Verzeichnis und Dateiname

für die zu erstellende PDF-Datei zu ermitteln, übergibt die Prozedur *RechnungErstellen* folgende Parameter an die Funktion *DateipfadErmitteln*:

» *strDatenherkunft*: Enthält den Namen der Tabelle, aus der Werte für Platzhalter stammen (hier *tblBestellungen*)

» *strID*: Primärschlüsselfeld dieser Tabelle (hier *BestellungID*)

» *lngID*: Wert des Primärschlüsselfeldes für den Datensatz, der die Werte für die Platzhalter liefern soll

» *strVorlageVerzeichnis*: Feld der Tabelle *tblOptionen*, welches die Vorlage für das Verzeichnis liefert (hier das Feld *VerzeichnisRechnungsdateien*)

» *strVorlageDateiname*: Feld der Tabelle *tblOptionen*, aus der die Vorlage für den Dateinamen stammt (hier das Feld *DateinameRechnungsdateien*)

Damit ermittelt die Funktion zunächst per *DLookup*-Funktion die Vorlagen für Verzeichnis und Dateiname und speichert diese in *strVerzeichnis* und *strDateiname*. In *strVerzeichnis* werden, soweit vorhanden, die Platzhalter *[Backend]* und *[Frontend]* durch entsprechende Verzeichnisse ersetzt. Wenn *strVerzeichnis* nicht auf \ endet, hängt die Prozedur dieses Zeichen nachträglich an. Dann folgt der Teil, der Platzhalter durch Feldinhalte der mit *strDatenherkunft* übergebenen Tabelle oder Abfrage ersetzt. Dazu durchläuft die Prozedur alle Felder dieser Datenherkunft und ersetzt Platzhalter, die dem in eckige Klammern eingefassten Feldnamen entsprechen, durch den Wert des jeweiligen Feldes. So wird aus *Rechnung_[KundeID]_[BestellungID].pdf* beispielsweise *Rechnung_12_31.pdf*. Die Prozedur *VerzeichnisErstellen* legt die Verzeichnisstruktur an, falls noch nicht geschehen (diese Prozedur finden Sie im Modul *mdlTools*). Der aus *strVerzeichnis* und *strDateiname* zusammengesetzte Ausdruck wird schließlich als Funktionswert zurückgeliefert.

```
Public Function DateipfadErmitteln(strDatenherkunft As String, strID As String, _
        lngID As Long, strVorlageVerzeichnis As String, _
        strVorlageDateiname As String) As String
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field
    Dim strVerzeichnis As String
    Dim strDateiname As String
    strVerzeichnis = DLookup(strVorlageVerzeichnis, "tblOptionen")
    strDateiname = DLookup(strVorlageDateiname, "tblOptionen")
    strVerzeichnis = Replace(strVerzeichnis, "[Backend]", Backendverzeichnis)
    strVerzeichnis = Replace(strVerzeichnis, "[Frontend]", CurrentProject.Path)
    If Not Right(strVerzeichnis, 1) = "\" Then
        strVerzeichnis = strVerzeichnis & "\"
    End If
    Set db = CurrentDb
```

```
    Set rst = db.OpenRecordset("SELECT * FROM " & strDatenherkunft & " WHERE " _
        & strID & " = " & lngID)
    For Each fld In rst.Fields
        On Error Resume Next
        strVerzeichnis = Replace(strVerzeichnis, "[" & fld.Name & "]", fld.Value)
        strDateiname = Replace(strDateiname, "[" & fld.Name & "]", fld.Value)
        On Error GoTo 0
    Next fld
    VerzeichnisErstellen strVerzeichnis
    DateipfadErmitteln = strVerzeichnis & strDateiname
End Function
```

7.2 Übersichtsformular für Rechnungen

Das Übersichtsformular für Rechnungen soll zeigen, wie Sie in einem Hauptformular und in einem Unterformular Daten aus der gleichen Datenquelle anzeigen, in diesem Falle aus der Tabelle *tblBestellungen*. Der Fokus liegt dabei natürlich auf den rechnungsrelevanten Daten.

7.2.1 Rechnung als bezahlt markieren oder Abgleich mit den Umsätzen?

Die obige Vorgehensweise erwartet, dass der Benutzer etwa beim manuellen Abgleich mit dem Kontoauszug die einzelnen Rechnungen durch Eintragen eines Datums in das Feld *BezahltAm* als bezahlt markiert.

Später werden Sie im Kapitel »Onlinebanking« (Seite 301) Techniken kennenlernen, mit denen Sie die Umsätze auf einem Bankkonto in einer Tabelle namens *tblUmsaetze* erfassen können. Diese Tabelle enthält auch ein Feld namens Betrag, der den ein- oder ausgehenden Umsatz liefert.

Wäre es nicht eine praktische Idee, wenn man die Umsätze gegen die Rechnungsbeträge abgleichen könnte? Im Idealfall zahlt der Kunde genau den Betrag, der auch auf der Rechnung angegeben ist, aber gelegentlich kommt es vor, dass er sich vertippt oder dass er nicht den kompletten Betrag auf einmal zahlt.

Wenn Sie nur ein Feld namens *BezahltAm* vorhalten und der Kunde den Betrag in mehr als einem Schritt bezahlt, müssen Sie schon das Bemerkungen-Feld bemühen, um dies zu vermerken – die Rechnung kann ja zu diesem Zeitpunkt nicht als bezahlt markiert werden.

Also machen wir aus der Not eine Tugend und ermöglichen es, einer Bestellung beziehungsweise Rechnung beliebig viele Beträge aus der Tabelle *tblUmsaetze* zuzuweisen. Eigentlich reicht es aus, wenn Sie der Tabelle *tblUmsaetze* ein Feld namens *BestellungID* hinzufügen, mit dem

ein Datensatz der Tabelle *tblUmsaetze* einem Datensatz der Tabelle *tblBestellungen* zugewiesen werden kann. Auf diese Weise können Sie einer Bestellung auch mehrere Umsätze zuweisen.

Andersherum soll man ein Datenmodell immer so flexibel anlegen wie möglich – zumindest, wenn es nicht schadet. Warum also nicht gleich eine m:n-Beziehung mithilfe einer Verknüpfungstabelle zwischen den Tabellen *tblBestellungen* und *tblUmsaetzen* anlegen, mit der Sie jeder Bestellung verschiedene Umsätze zuweisen können und umgekehrt? Aber warum der umgekehrte Fall?

Nun: Es kann ja auch einmal vorkommen, dass ein Kunde zwei oder mehr Bestellungen mit einer Überweisung bezahlt. Diesen Fall wollen wir in dieser Version der Anwendung nicht ausprogrammieren, aber wir schaffen so zumindest die Möglichkeit, dieses Feature zu gegebener Zeit hinzuzufügen. Die Verknüpfungstabelle sieht wie in Abbildung 7.6 aus.

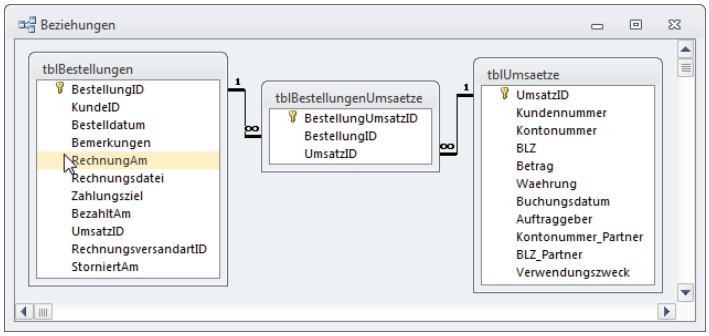


Abbildung 7.6: Verknüpfungstabelle für flexible Zuteilung von Umsätzen zu Bestellungen

Die Tabelle *tblUmsaetze* wird später im Kapitel »Onlinebanking« (Seite 301) genauer beschrieben, die Tabelle *tblBestellungen* kennen Sie ja bereits.

Mit diesen Informationen im Hinterkopf schauen wir uns zunächst das komplette Formular an – es ist ein wenig aufwendiger, daher kann ein vorheriger Überblick nicht schaden. Sie finden das Formular in Abbildung 7.7.

Das Formular *frmRechnungsuebersicht* zeigt im oberen Bereich alle Rechnungen in einem Unterformular in der Datenblattansicht an. Darüber finden Sie einige Filter-Steuerelemente. Damit können Sie die angezeigten Bestellungen nach folgenden Kriterien filtern:

» Bestelldatum

» Rechnungsdatum

» Rechnung bezahlt

» Rechnung storniert

Sie werden sicher weitere Kriterien finden, die für Ihren speziellen Fall sinnvoll sind.

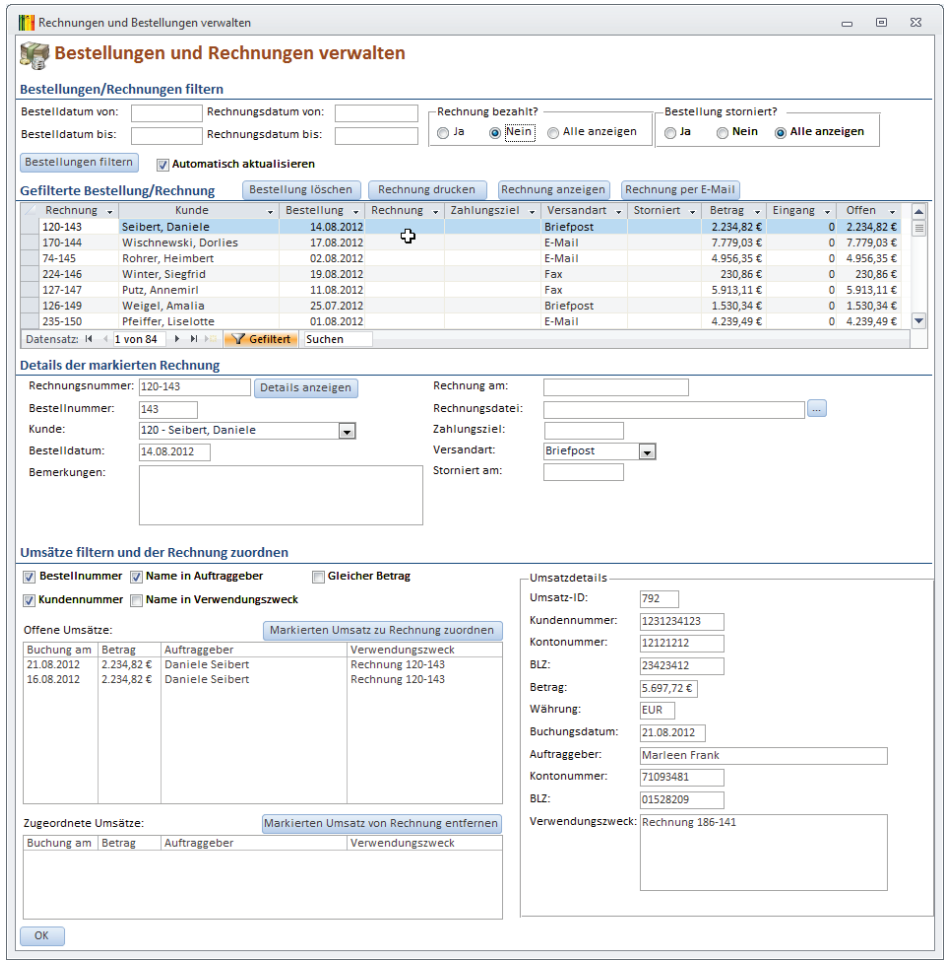


Abbildung 7.7: Verwaltung von Bestellungen/Rechnungen

Ein Doppelklick auf eines der Datumstextfelder öffnet den Dialog aus Abbildung 7.8, mit dem Sie komfortabel den anzuzeigenden Zeitraum auswählen können. Dort klicken Sie bei gedrückter Maustaste auf das erste Datum des gewünschten Zeitraums und lassen die Maus über dem zweiten Datum wieder los. Der Zeitraum wird im Formular angezeigt und dann in das aufrufende Formular übernommen.

Für die beiden übrigen Filterkriterien, die festlegen, ob nur bezahlte, nicht bezahlte, stornierte oder nicht stornierte Artikel angezeigt werden sollen, haben wir jeweils eine Optionsgruppe vorgesehen.

Zuerst war hier jeweils ein Kontrollkästchen geplant, aber damit kann man ja nur entweder alle bezahlten oder alle nicht bezahlten Bestellungen anzeigen – aber nicht einfach alle Bestellungen.

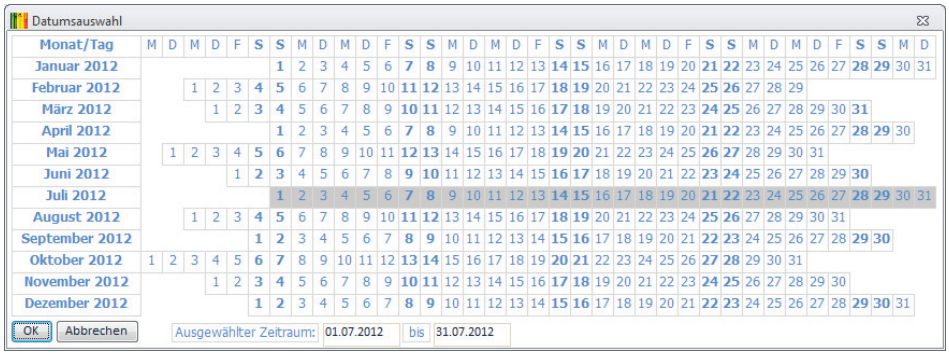


Abbildung 7.8: Zeiträume auswählen per Formular

Nach dem Filtern und Auswählen einer Bestellung können Sie über die Schaltflächen direkt über dem Unterformular eine Bestellung löschen oder die Rechnung zur Bestellung drucken, anzeigen oder gleich per E-Mail verschicken.

Außerdem zeigen die Steuerelemente unterhalb des Unterformulars alle Details der Bestellung an (mit Ausnahme der Bestellpositionen – aber die erhalten Sie, wenn Sie auf die Schaltfläche *Details anzeigen* klicken).

Neben einem Überblick über alle Rechnungen befindet sich im unteren Bereich noch eine ganz wichtige Funktion der Anwendung: Dort werden alle in der Tabelle *tblUmsaetze* verbuchten Umsätze aufgelistet und können einer Bestellung zugewiesen werden.

Dazu zeigt das obere von zwei Listenfeldern alle noch nicht zugewiesenen Umsätze an. Das untere Listenfeld hingegen liefert alle Umsätze der Tabelle *tblUmsaetze*, die über die Verknüpfungstabelle *tblBestellungenUmsaetze* bereits einer Bestellung zugewiesen wurden.

Wie soll man nun aus den möglicherweise vielen Eingängen genau den herausfinden, der zur aktuellen Bestellung passt? Ganz einfach: Dazu verwenden Sie die Kontrollkästchen über dem Listenfeld, die folgende Wirkung haben:

» *Bestellnummer*: Untersucht den Verwendungszweck auf das Vorkommen der Bestellnummer.

» *Kundennummer*: Untersucht den Verwendungszweck auf das Vorkommen der Kundennummer.

» *Name in Auftraggeber*: Sucht im Feld Auftraggeber nach dem Vornamen oder dem Nachnamen – es reicht für einen Treffer, wenn eines von beiden vorkommt.

» *Name in Verwendungszweck*: Sucht im Feld *Verwendungszweck* nach Vor- und Nachname.

» *Gleicher Betrag*: Sucht nach Datensätzen, deren Umsatz mit der Bestellsumme übereinstimmt. Hier könnte man gegebenenfalls noch die Schrauben lockern und ein paar Euro oder Cent Spiel einräumen.

Rechnung anzeigen

Das Anzeigen der Rechnung als Rechnungsbericht erfolgt auf Basis der aktuell vorliegenden Daten. Dazu ruft die durch die Schaltfläche *Rechnung anzeigen* ausgelöste Prozedur den Bericht *rptRechnung* mit den entsprechenden Parametern auf:

```
Private Sub cmdRechnungAnzeigen_Click()
    DoCmd.OpenReport "rptRechnung", View:=acViewPreview, _
        WhereCondition:="BestellungID = " & Me!sfmRechnungsuebersicht.Form!BestellungID
End Sub
```

Rechnung neu erstellen und drucken

Auch das Erstellen einer Rechnung als PDF-Datei und das anschließende Drucken soll möglich sein. Die Prozedur, die dies erledigt, sieht so aus:

```
Private Sub cmdRechnungDrucken_Click()
    Me!Rechnungsdatei = RechnungErstellen(Me!sfmRechnungsuebersicht.Form!BestellungID)
    Me!RechnungAm = Now
    DoCmd.OpenReport "rptRechnung", acViewNormal, WhereCondition:="BestellungID = " _
        & Me!sfmRechnungsuebersicht.Form!BestellungID
End Sub
```

Rechnung per E-Mail verschicken

Fehlt noch das Versenden einer Rechnung per E-Mail. Dies erledigt die Schaltfläche *Rechnung per E-Mail*, welche den Hauptteil der Arbeit an die Prozedur *RechnungPerMail* weitergibt – siehe »E-Mail mit Rechnung versenden« (Seite 441). Im Anschluss aktualisiert die Prozedur noch die Felder, welche Daten über die Erstellung der E-Mail speichern:

```
Private Sub cmdRechnungPerMail_Click()
    Dim strRechnungsdatei As String
    Dim datRechnungAm As Date
    Dim lngRechnungsversandartID As Long
    RechnungPerMail Me!sfmRechnungsuebersicht.Form!BestellungID, strRechnungsdatei, _
        datRechnungAm, lngRechnungsversandartID
    Me!txtRechnungsdatei = strRechnungsdatei
    Me!RechnungAm = datRechnungAm
    Me!RechnungsversandartID = lngRechnungsversandartID

End Sub
```

Fertig – mit diesem Formular können Sie nun prima Bestellungen und Rechnungen verwalten und diesen die Geldeingänge zuweisen.

8 Kommunikation verwalten

Die E-Mail-Kommunikation ist ein wichtiges Instrument, um Kundenanfragen entgegenzunehmen und zu beantworten. Die in diesem Buch vorgestellte Version der Anwendung lässt andere Kommunikationsarten (außer dem Erstellen von individuellen Anschreiben, die dann als PDF per Mail oder als Brief verschickt werden) außen vor und konzentriert sich auf die Verwaltung von E-Mails.

Die Kommunikation wird zunächst kundenbezogen verwaltet. Das bedeutet, dass wir im Formular zur Verwaltung der Kundendetails auch einen Bereich vorsehen, der sich allein der Kommunikation per E-Mail widmet.

Dieser Bereich soll auf einer eigenen Seite des Register-Steuerelements im Formular *frmKunde-Detail* landen (siehe Abbildung 8.1).

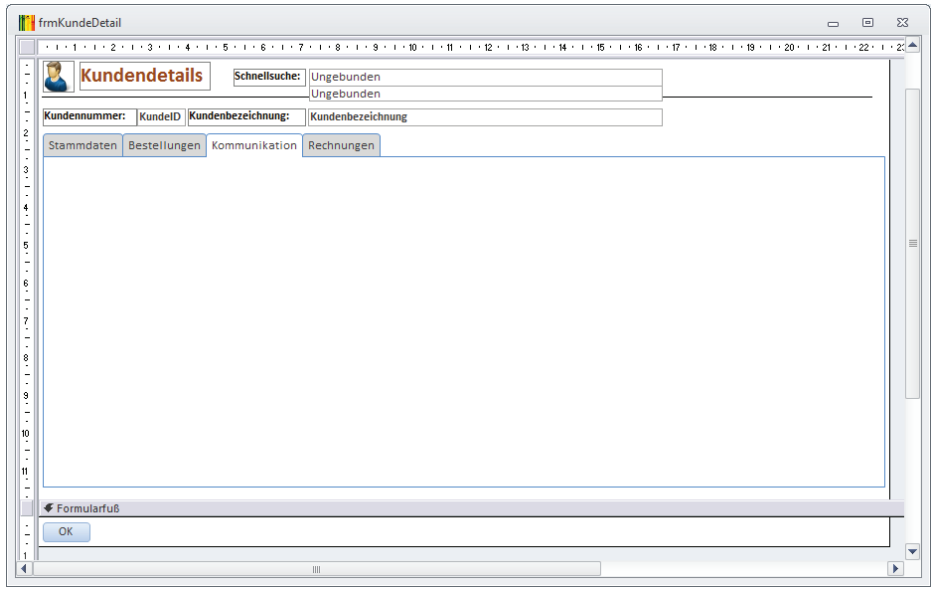


Abbildung 8.1: Der für die Kommunikationsdaten vorgesehene Bereich

Als Erstes fügen wir jedoch ein neues Formular zur Anwendung hinzu, um die weiteren E-Mail-Adressen eines Kunden erfassen zu können. Dieses Formular basiert auf der Tabelle *tblEMail-Adressen*, das ein Fremdschlüsselfeld zum Herstellen der Beziehung zum Kundendatensatz aufweist.

Das Formular soll durch einen Mausklick auf eine Schaltfläche angezeigt werden, die sich neben dem Feld zum Eintragen der primären E-Mail-Adresse befindet (siehe Abbildung 8.2).

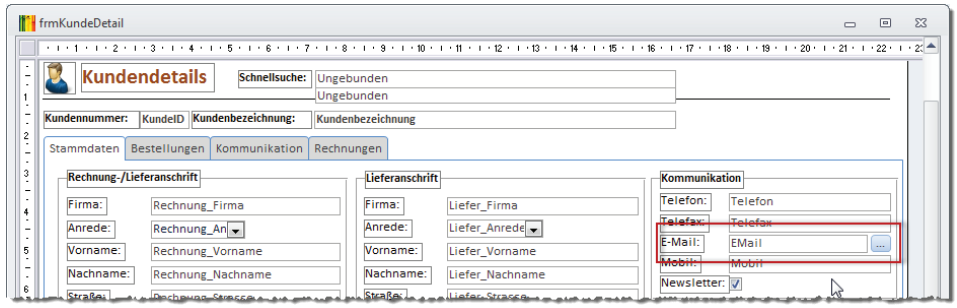


Abbildung 8.2: Schaltfläche zum Öffnen des Formulars mit weiteren E-Mail-Adressen

Das Unterformular verwendet die Tabelle *tblEMailAdressen* als Datenherkunft. Es soll nur das Feld *EMailAdresse* anzeigen, das Sie zu diesem Zweck in den Detailbereich des Formularentwurfs ziehen. Stellen Sie die Eigenschaft *Standardansicht* auf *Datenblatt* ein (siehe Abbildung 8.3). Außerdem können Sie gleich den Text des Bezeichnungsfeldes, das im Datenblatt als Spaltenüberschrift angezeigt wird, von *EMailAdresse* in *E-Mail-Adresse* ändern.

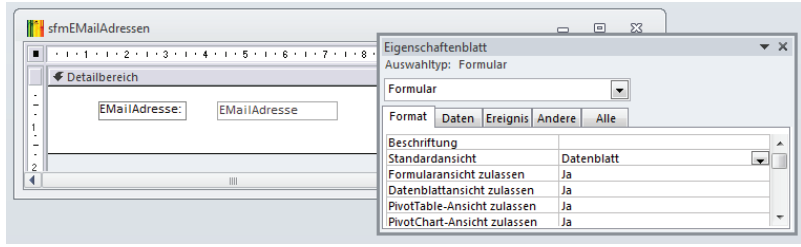


Abbildung 8.3: Das Unterformular *sfmEMailAdressen*

Anschließend erstellen Sie das Hauptformular, in welches das Unterformular eingebettet werden soll. Dieses verwendet die Tabelle *tblKunden* als Datenherkunft und zeigt die beiden Felder *KundeID* und *Kundenbezeichnung* an. Stellen Sie die Eigenschaften *Aktiviert* und *Gesperrt* auf die Werte *Nein* und *Ja* ein, damit der Benutzer diese in der aktuellen Ansicht nicht anpassen kann.

Fügen Sie dann das mittlerweile geschlossene Formular *sfmEMailAdressen* als Unterformular in das Formular *frmEMailAdressen* ein. Dadurch, dass das Hauptformular die Tabelle *tblKunden* als Datenherkunft verwendet und dass die Datenherkunft des Unterformulars ein Fremdschlüsselfeld mit Bezug auf diese Tabelle enthält, stellt Access beim Hinzufügen automatisch eine Verknüpfung zwischen Haupt- und Unterformular her. Diese wird in den Eigenschaften *Verknüpfen von* und *Verknüpfen nach* festgelegt (siehe Abbildung 8.4).

Damit ein Mausklick auf die Schaltfläche *cmdEMailAdressen* auf der Registerseite *Stammdaten* des Formulars *frmKundeDetail* nun das Formular *frmEMailAdressen* öffnet, fügen Sie diesem eine Ereignisprozedur hinzu, die wie folgt aussieht:

```
Private Sub cmdEMailAdressen_Click()
    DoCmd.OpenForm "frmEMailAdressen", WindowMode:=acDialog, _
        WhereCondition:="KundeID = " & Me!KundeID
End Sub
```

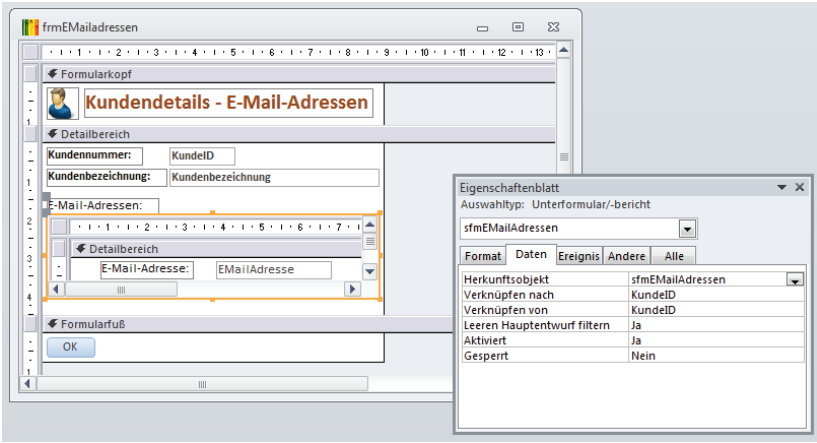


Abbildung 8.4: Einbau des Unterformulars ins Hauptformular

Dies zeigt das Formular *frmEMailAdressen* wie in Abbildung 8.5 an. Diesem fehlt nur noch eine Ereignisprozedur für die Schaltfläche *cmdOK*. Die folgende Prozedur schließt das Formular *frmEMailAdressen* wieder:

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

8.1 E-Mail-Adressen in tblKunden und tblEMailAdressen

Wie gehen wir nun mit den an verschiedenen Stellen gespeicherten E-Mail-Adressen um? Es gibt nun schließlich ein Feld namens *EMail* in der Tabelle *tblKunden* und eines in der verknüpften Tabelle *tblEMailAdressen*.

Das Feld in der Tabelle *tblKunden* hätte seinen Sinn, wenn wir damit festlegen, welche der E-Mail-Adressen in der Tabelle *tblEMailAdressen* die Standard-E-Mail-Adresse für einen Kunden ist.

Da wir noch in der Entwicklungsphase der Anwendung sind, machen wir das doch einfach: Das Feld *EMail* in der Tabelle *tblKunden* soll keinen Text mehr mit der primären E-Mail-Adresse enthalten, sondern einen Verweis auf den Datensatz der Tabelle *tblEMailAdressen*, welcher die primäre E-Mail-Adresse des Kunden enthält.

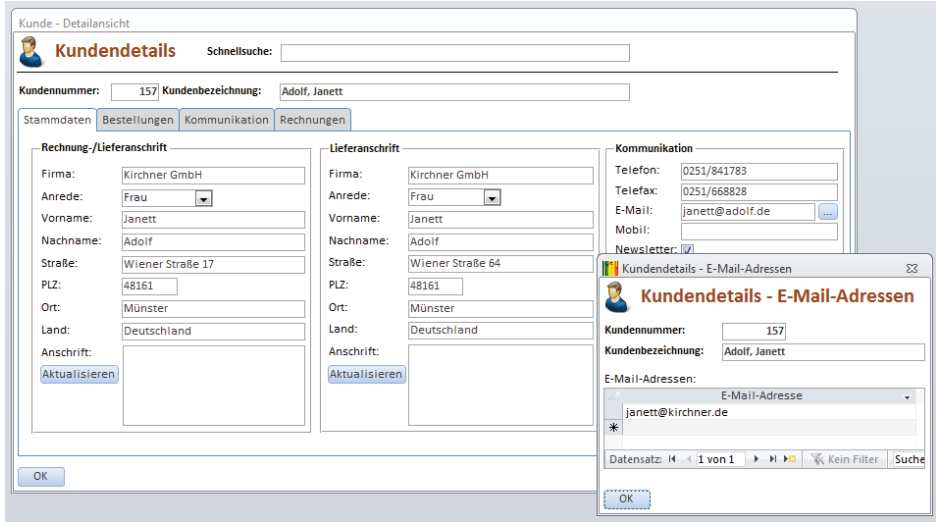


Abbildung 8.5: Anzeigen der weiteren E-Mail-Adressen per Mausklick vom Kundenformular aus

Wenn Sie diese Umstellung vornehmen, wenn bereits Daten im Feld *EMail* der Tabelle *tblKunden* enthalten sind, müssen Sie diese zunächst in die Tabelle *tblEMailAdressen* übertragen.

Danach legen wir ein neues Feld namens *EMailID* an und stellen dieses für jeden Kunden auf den Datensatz der Tabelle *tblEMailAdressen* ein, welcher dem vorherigen Wert im Feld *EMail* entspricht. Das Feld *EMail* kann danach aus der Tabelle *tblKunden* entfernt werden. Schließlich müssen Sie noch das Formular *frmKundeDetail* auf das neue Feld anpassen – dazu später mehr.

Um die E-Mail-Adressen unter Angabe des Wertes des Feldes *KundeID* aus der Tabelle *tblKunden* in die Tabelle *tblEMailAdressen* zu übertragen, verwenden wir eine Anfügeabfrage. Diese soll zunächst alle E-Mail-Adressen samt passender Kundennummer in die Tabelle *tblEMailAdressen* schreiben. Erstellen Sie dazu zunächst eine Abfrage namens *qryINSERTINTO_tblEMailAdressen*, welche die beiden Felder *KundeID* und *EMail* der Tabelle *tblKunden* enthält (siehe Abbildung 8.6).

Klicken Sie dann im Ribbon auf den Eintrag *Entwurf|Abfragetyp|Anfügen* und wählen Sie im folgenden Dialog den Eintrag *tblEMailAdressen* als Zieltabelle aus (siehe Abbildung 8.7).

Im Abfrageentwurf können Sie nun für jedes Quellfeld ein Zielfeld in der Tabelle *tblEMailAdressen* auswählen. Die Werte des Feldes *KundeID* sollen dabei im gleichnamigen Fremdschlüsselfeld landen, die Werte des Feldes *EMail* im Feld *EMailAdresse* (siehe Abbildung 8.8).

Wenn Sie die Anfügeabfrage nun mit dem Ribbon-Befehl *Entwurf|Ergebnisse|Ausführen* starten, trägt diese alle E-Mail-Adressen samt *KundeID* in die Tabelle *tblEMailAdressen* ein. Es könnte zu Fehlern beim Anfügen wegen Schlüsselverletzungen kommen. Dies liegt daran, dass vielleicht schon die eine oder andere E-Mail-Adresse in der Tabelle *tblEMailAdressen* vorhanden ist – dies

können Sie ignorieren. Wichtig ist allein, dass jede E-Mail-Adresse nun entweder hinzugefügt wurde oder bereits vorhanden ist.

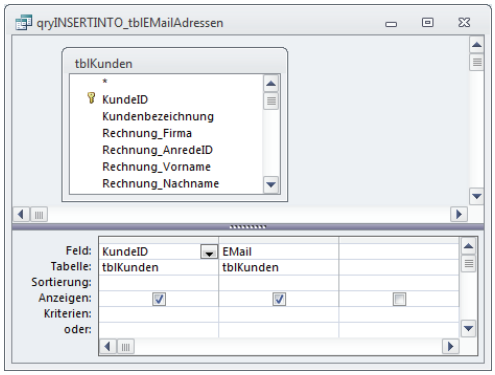


Abbildung 8.6: Vorbereitung der Anfügeabfrage zum Übertragen der E-Mail-Adressen

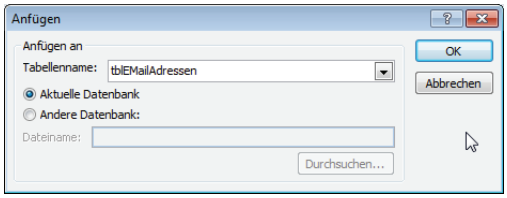


Abbildung 8.7: Auswahl der Zieltabelle für die Anfügeabfrage

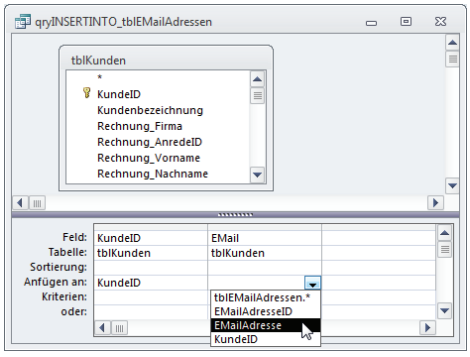


Abbildung 8.8: Zuteilen der Quell- und Zielfelder

Fügen Sie nun ein neues Feld namens *EMailAdresseID* zur Tabelle *tblKunden* hinzu – am besten gleich unterhalb des Feldes *EMail* (siehe Abbildung 8.9). Um die Auswahl zu vereinfachen, soll das Feld als Nachschlage-Feld eingerichtet werden, was Sie am schnellsten mit dem Nachschlage-Assistenten hinbekommen.

Nun enthält das Feld *EMailAdresseID* der Tabelle *tblKunden* noch keinen Wert. Dieser soll nun auf den Datensatz der Tabelle *tblEMailAdressen* eingestellt werden, der die zuvor im Feld *EMail* der Tabelle *tblKunden* enthaltene E-Mail-Adresse enthält.

Dazu erstellen Sie wiederum eine Aktionsabfrage, diesmal jedoch als Aktualisierungsabfrage ausgeführt. Nach dem Öffnen einer neue Abfrage in der Entwurfsansicht fügen Sie zunächst die beiden Tabellen *tblKunden* und *tblEMailAdressen* hinzu.

Dies liefert nun nach den Änderungen im Datenmodell ein interessantes Bild, denn beide Tabellen sind jeweils über ein Fremdschlüsselfeld mit der jeweils anderen Tabelle verknüpft: *tblKunden* referenziert *tblEMailAdressen* über das Feld *EMailAdresseID*, *tblEMailAdressen* wiederum legt über das Feld *KundeID* fest, zu welchem Kunden eine E-Mail-Adresse gehört.

Beide werfen wir im Rahmen dieser Abfrage über Bord und erstellen eine neue Beziehung zwischen den beiden Tabellen – nämlich zwischen den Feldern *EMail* und *EMailAdresse*. Ziehen Sie dann das Feld *EMailAdresseID* der Tabelle *tblKunden* in das Entwurfsraster und wandeln Sie die Abfrage mit dem Ribbon-Eintrag *Entwurf|Abfragetyp|Aktualisierung* in eine Aktualisierungsabfrage um. Nun tragen Sie in die Zeile *Aktualisieren* den Ausdruck *[tblEMailAdressen].[EMailAdresse]* ein (siehe Abbildung 8.10). Die Abfrage fügt nun beim Ausführen jeweils den Wert des Feldes *EMailAdresseID* der Tabelle *tblEMailAdressen* in das Fremdschlüsselfeld *EMailAdresseID* der Tabelle *tblKunden* ein, für welches die E-Mail-Adresse zwischen den beiden Feldern *EMail* und *EMailadresse* übereinstimmt.

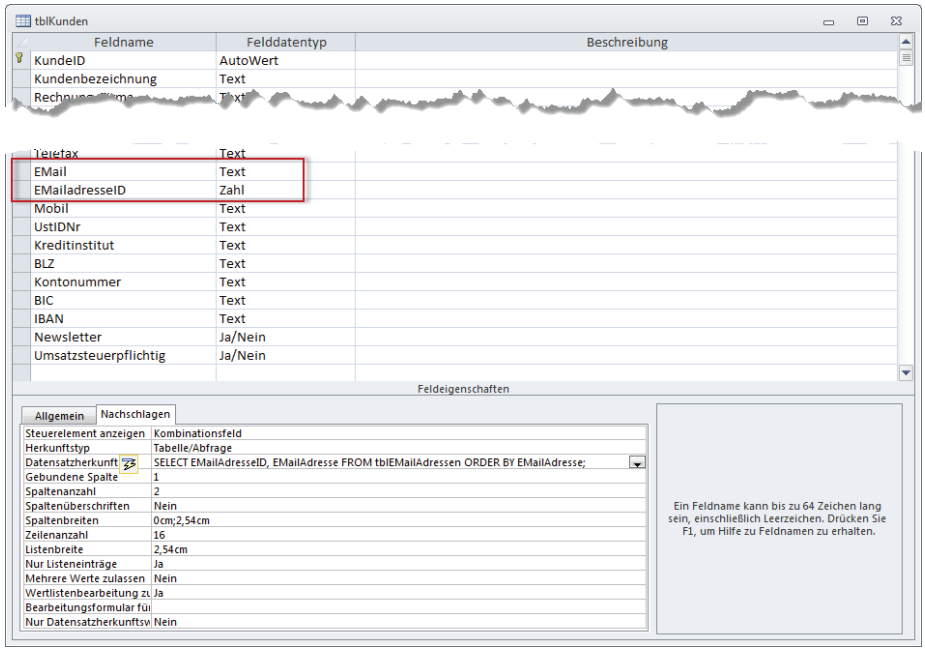


Abbildung 8.9: Neues Feld zum Einstellen der Standard-E-Mail-Adresse

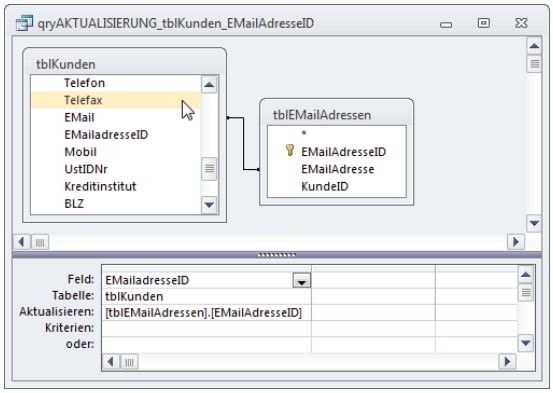


Abbildung 8.10: Herstellen der Verknüpfung zwischen den Datensätzen der Tabellen *tblKunden* und *tblEMailAdressen*

Anschließend können Sie das Feld *EMail* aus der Tabelle *tblKunden* entfernen. Aber Achtung: Vor solchen Aktionen lohnt es sich immer, eine Sicherungskopie der Datenbank anzulegen.

Nun müssen wir noch das Formular *frmKundeDetail* so anpassen, dass sich die primäre E-Mail-Adresse einfach auswählen lässt. Dazu stellen Sie zunächst den Steuerelementinhalt des Textfeldes *txtEMail* auf das neue Feld *EMailAdresseID* ein. Wenn Sie das Feld *EMail* bereits aus der Tabelle *tblKunden* gelöscht haben, dürfte dieses Steuerelement ohnehin bereits mit einem Hinweis versehen sein (siehe Abbildung 8.11).

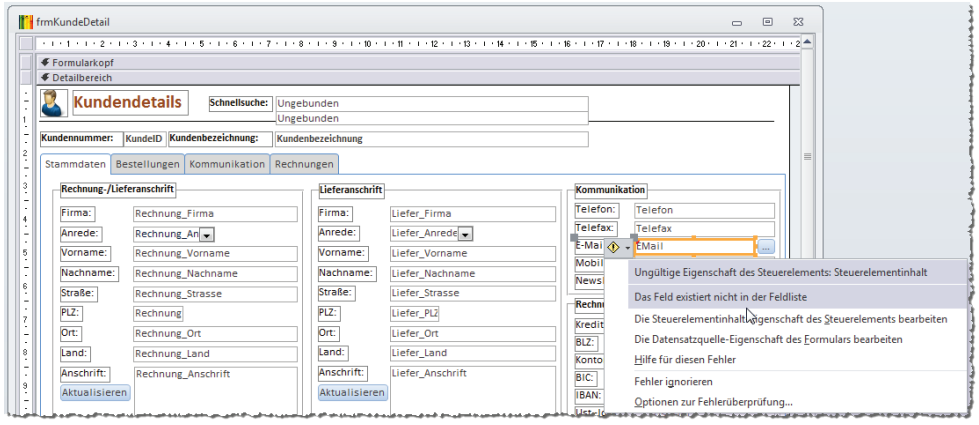


Abbildung 8.11: Access bemängelt den ungültigen Steuerelementinhalt.

Wandeln Sie das Feld mit dem Kontextmenü-Eintrag *Ändern zu|Kombinationsfeld* in ein Kombinationsfeld, stellen Sie seinen Namen auf *cboEMailAdresseID* ein und legen Sie für die Eigenschaften *Datensatzherkunft* den folgenden Ausdruck fest:

```
SELECT MailAdresseID, EMailAdresse FROM tblEMailAdressen
WHERE KundeID=[Forms]![frmKundeDetail]![KundeID];
```

Stellen Sie die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *2* und *0cm* ein, damit nur die E-Mail-Adresse angezeigt wird. Ändern Sie außerdem den Namen des Steuerelements in *cboEMailAdresseID*.

Damit der Inhalt des Kombinationsfeldes nach dem Ändern der E-Mail-Adressen im Formular *frmEMailAdressen* aktualisiert wird, fügen Sie zur Ereignisprozedur für die Schaltfläche *cmdMailAdressen* noch eine Zeile hinzu:

```
Private Sub cmdEMailAdressen_Click()
    DoCmd.OpenForm "frmEMailAdressen", WindowMode:=acDialog, _
        WhereCondition:="KundeID = " & Me!KundeID
    Me!cboEMailAdresseID.Requery
End Sub
```

Im Unterformular *sfmKundenuebersicht* des Formulars *frmKundenuebersicht* sind kleinere Änderungen nötig, die Sie in der Beispieldatenbank betrachten können. Dort wurde die als Datenherkunft verwendete Abfrage um das Feld *EMailAdresse* der Tabelle *tblEMailAdressen* erweitert.

Auch im Suchformular *frmKundensuche* muss das Textfeld *txtEMail* in *txtEMailAdresse* umbenannt werden.

8.2 E-Mail an Kunde verschicken

Es gibt eine Reihe von Aktionen, die Sie in Zusammenhang mit einem Kunden erledigen können. Ein einfaches Beispiel ist eine E-Mail an einen Kunden. Die folgenden Abschnitte zeigen, wie wir dies in die Anwendung integriert haben.

8.2.1 E-Mail-Adresse auswählen

Auf der Registerseite *Kommunikation* soll der Benutzer eine E-Mail an den aktuell ausgewählten Kunden erstellen können. Dazu soll er zunächst mit einem Kombinationsfeld namens *cboEMailAdresseNeueMail* die Mail-Adresse des Adressaten auswählen (siehe Abbildung 8.12). Dieses Kombinationsfeld verwendet den folgenden Ausdruck als *Datensatzherkunft*:

```
SELECT EMailAdresseID, EMailAdresse FROM tblEMailAdressen
WHERE KundeID=[Forms]![frmKundeDetail]![KundeID];
```

Damit dieses Steuerelement beim Öffnen jeweils die Standard-E-Mail-Adresse dieses Kunden anzeigt, fügen Sie dem Ereignis, das beim Anzeigen eines Datensatzes ausgelöst wird, die folgende Zeile hinzu:

```
Private Sub Form_Current()
    ...
    Me!cboEMailAdresseNeueMail = Me!cboEMailAdresseID
End Sub
```

Dies übernimmt die aktuelle Standardadresse aus dem Feld *MailAdresseID* in das Kombinationsfeld *cboEMailAdresseNeueMail*.

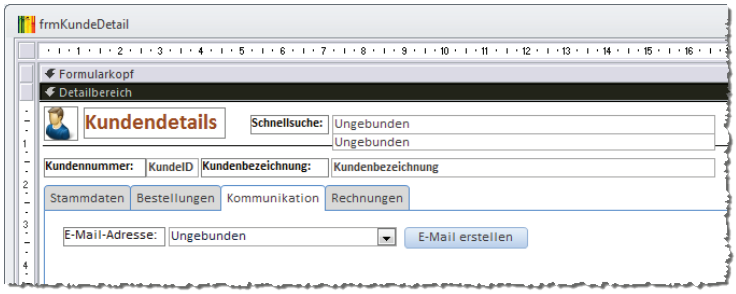
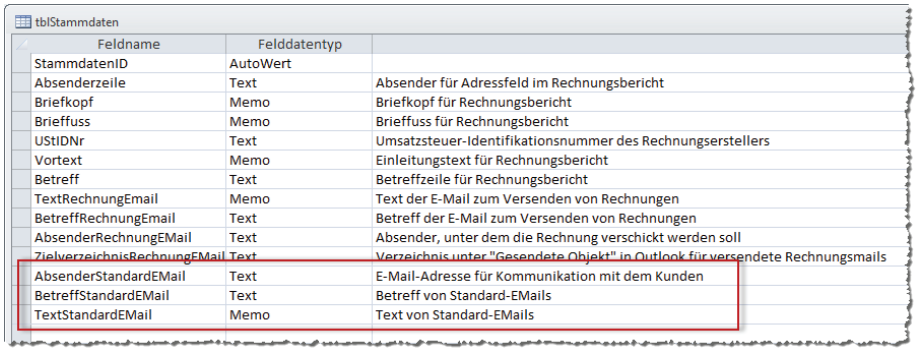


Abbildung 8.12: Kombinationsfeld und Schaltfläche zum Erstellen einer neuen E-Mail

8.2.2 Vorlage für eine Standard-E-Mail

Bevor wir die E-Mail erstellen, benötigen wir noch einen Satz von Stammdaten. Diese landen wie in einigen anderen Fällen in der Tabelle *tblStammdaten* (siehe Abbildung 8.13).


tblStammdaten

Feldname	Felddatentyp	
StammdatenID	AutoWert	
Absenderzeile	Text	Absender für Adressfeld im Rechnungsbericht
Briefkopf	Memo	Briefkopf für Rechnungsbericht
Brieffuss	Memo	Brieffuss für Rechnungsbericht
USt IDNr	Text	Umsatzsteuer-Identifikationsnummer des Rechnungserstellers
Vortext	Memo	Einleitungstext für Rechnungsbericht
Betreff	Text	Betreffzeile für Rechnungsbericht
TextRechnungEmail	Memo	Text der E-Mail zum Versenden von Rechnungen
BetreffRechnungEmail	Text	Betreff der E-Mail zum Versenden von Rechnungen
AbsenderRechnungEMail	Text	Absender, unter dem die Rechnung verschickt werden soll
ZielverzeichnisRechnungEMail	Text	Verzeichnis unter "Gesendete Objekt" in Outlook für versendete Rechnungsmails
AbsenderStandardEMail	Text	E-Mail-Adresse für Kommunikation mit dem Kunden
BetreffStandardEMail	Text	Betreff von Standard-EMails
TextStandardEMail	Memo	Text von Standard-EMails

Abbildung 8.13: Stammdaten für den Versand von E-Mails an den Kunden

Das Formular aus Abbildung 8.14 bietet dem Benutzer die Möglichkeit, die in diesen Feldern enthaltenen Daten gezielt zu bearbeiten.

In diesen Text können Sie auch Platzhalter eintragen, wobei Sie diese Vorlage möglichst einfach halten sollten – es gibt noch eine weitere Möglichkeit, auf spezielle Fälle zugeschnittene Vorlagen einzusetzen.

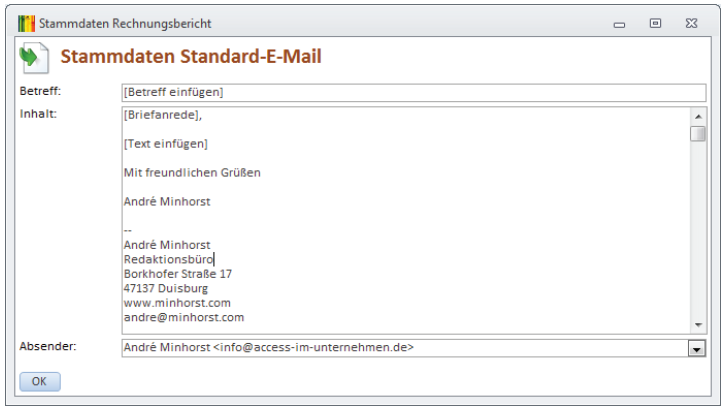


Abbildung 8.14: Formular zur Anzeige von E-Mail-Stammdaten

8.2.3 Standard-E-Mail erstellen und verschicken

Das eigentliche Erstellen der E-Mail auf Basis dieser Vorlage erläutern wir ein paar Abschnitte weiter unten. Warum? Weil es noch eine weitere Möglichkeit gibt, flexibel E-Mails auf Basis benutzerdefinierter Vorlagen zu erstellen. Beide Varianten verwenden aber die gleiche Funktion, um die E-Mail zu erstellen. Daher schauen wir uns an dieser Stelle an, was bei einem Mausklick auf die Schaltfläche *cmdEMailErstellen* geschieht:

```
Private Sub cmdEMailErstellen_Click()
    EMailErstellen Me!cboEMailAdresseNeueMail.Column(1), Me!KundeID
End Sub
```

Diese Prozedur ruft eine Funktion namens *EMailErstellen* auf, die nur zwei Parameter benötigt: die E-Mail-Adresse, an welche die E-Mail geschickt werden soll, sowie den Wert des Feldes *KundeID* für den aktuell angezeigten Kunden. Die Funktion verwendet noch einen dritten Parameter, der aber hier nicht zum Zuge kommt, da die Standardvorlage verwendet werden soll. Mehr zu dieser Funktion erfahren Sie weiter unten unter 8.3.

8.2.4 Individuelle E-Mail-Vorlagen

Neben dieser einen Standardvorlage gibt es für den Benutzer noch die Möglichkeit, weitere Vorlagen anzulegen – und zwar für verschiedenste Zwecke. Legen Sie eine Vorlage für die Antwort auf Beschwerden an, eine, um sich für ein Lob zu bedanken, oder auch Vorlagen, mit denen Sie auf häufige Anfragen von Kunden antworten können.

Die flexiblen Vorlagen werden in der Tabelle *tblVorlagentexte* gespeichert. Diese Tabelle enthält nur drei Felder:

» *VorlagenID*: Primärschlüsselfeld der Tabelle

» *VorlagenBezeichnung*: Bezeichnung der Vorlage

» *VorlageInhalt*: Memofeld mit dem eigentlichen Vorlagentext

» *VorlageBetreff*: Textfeld mit dem Betreff für die Vorlage

Interessant ist das Formular, mit dem Sie die Vorlagentexte bearbeiten können. Dieses Formular sieht im Entwurf wie in Abbildung 8.15 aus. Es verwendet die Tabelle *tblVorlagen* als Datenherkunft. Die drei Felder auf der rechten Seite, *txtVorlageBezeichnung, txtVorlageInhalt* und *txtVorlageBetreff*, sind an die Felder *VorlageBezeichnung, VorlageInhalt* und *VorlageBetreff* dieser Tabelle gebunden. Das Listenfeld *lstVorlagen* verwendet die gleiche Tabelle als *Datensatzherkunft*, allerdings nur die ersten beiden Felder – und nach der Bezeichnung des Vorlagentextes sortiert. Der Ausdruck für die Datensatzherkunft lautet wie folgt:

```
SELECT VorlagenID, VorlageBezeichnung FROM tblVorlagen
ORDER BY VorlageBezeichnung;
```

Damit das Listenfeld das erste, gebundene Feld ausblendet und nur das zweite Feld mit den Vorlagenbezeichnungen anzeigt, stellen Sie die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *2* und *0cm* ein.

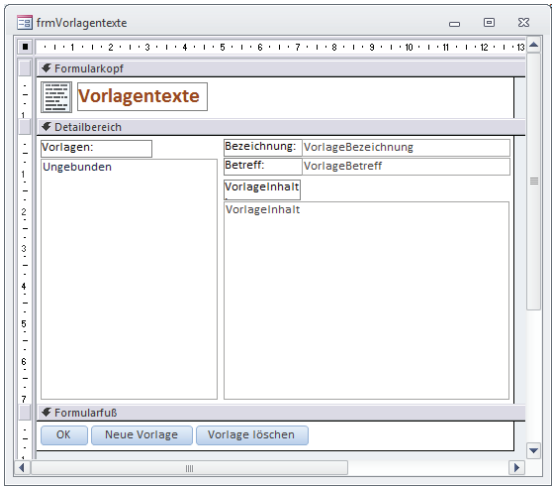


Abbildung 8.15: Entwurfsansicht des Formulars *frmVorlagentexte*

8.2.5 Aktionen beim Öffnen des Formulars

Beim Öffnen des Formulars wird das Ereignis *Beim Laden* ausgelöst, für das Sie die folgende Ereignisprozedur hinterlegen:

```
Private Sub Form_Load()
    Me!lstVorlagen = Me!lstVorlagen.ItemData(0)
```

```
    If Not IsNull(Me!lstVorlagen) Then
        Me.Recordset.FindFirst "VorlageID = " & Me!lstVorlagen
    End If
End Sub
```

Diese sorgt dafür, dass im Listenfeld gleich zu Beginn der erste Datensatz markiert wird. Dazu ermitteln Sie mit der *ItemData*-Eigenschaft den Wert der gebundenen Spalte für den ersten Datensatz (der Index ist *0*-basiert) und weisen diesen dem Listenfeld als Wert zu. Die folgende *If...Then*-Bedingung prüft, ob damit ein Datensatz markiert wurde oder ob das Listenfeld wie vorher den Wert *Null* hat – was bedeutet, dass gar kein Datensatz in der zugrunde liegenden Tabelle *tblVorlagen* enthalten ist. Wurde jedoch durch die erste Anweisung ein Eintrag im Listenfeld markiert, sucht die Prozedur den ersten Datensatz des Recordsets des Formulars und setzt den Datensatzzeiger auf diesen Datensatz.

Formular nach Listenauswahl aktualisieren

Wenn der Benutzer einen anderen Eintrag im Listenfeld auswählt, löst dies das Ereignis *Nach Aktualisierung* des Listenfeldes aus. Dies ist gleichbedeutend mit dem Aufruf der folgenden Ereignisprozedur, die den Datensatz mit dem entsprechenden Wert im Feld *VorlagentextID* im Formular anzeigt:

```
Private Sub lstVorlagentexte_AfterUpdate()
    Me.Recordset.FindFirst "VorlageID = " & Me!lstVorlagen
End Sub
```

Anlegen eines neuen Vorlagentextes

Um einen neuen Vorlagentext anzulegen, klickt der Benutzer auf die Schaltfläche *cmdNeueVorlage*. Dies löst die folgende Prozedur aus:

```
Private Sub cmdNeueVorlage_Click()
    Dim strVorlage As String
    Dim db As DAO.Database
    Dim lngVorlageID As Long
    strVorlage = InputBox("Geben Sie die Bezeichnung der neuen Vorlage ein.", _
        "Neue Vorlage")
    If IsNull(DLookup("VorlagenID", "tblVorlagen", _
            "VorlageBezeichnung = '" & strVorlage & "'")) Then
        If Len(strVorlage) > 0 Then
            Set db = CurrentDb
            db.Execute "INSERT INTO tblVorlagen(VorlagenBezeichnung) VALUES('" _
                & strVorlage & "')", dbFailOnError
            lngVorlageID = db.OpenRecordset("SELECT @@IDENTITY").Fields(0)
            Me!lstVorlagen.Requery
```

```
            Me!lstVorlagen = lngVorlageID
            Me.Requery
            Me.Recordset.FindFirst "VorlageID = " & lngVorlageID
            Me!txtVorlageInhalt.SetFocus
            Set db = Nothing
        End If
    Else
        MsgBox "Die Vorlagenbezeichnung '" & strVorlage & "' ist bereits vorhanden."
    End If
End Sub
```

Wenn Sie denken, dass das Anlegen eines neuen Datensatzes doch schneller zu erledigen wäre, wenn man einfach den folgenden Befehl absetzt, haben Sie Recht:

```
DoCmd.GoToRecord Record:=acNewRec
```

Aber diese Variante erledigt nicht das Gleiche wie die oben abgebildete Ereignisprozedur. Diese fragt zuerst den Namen für den neuen Vorlagentext ab und speichert diesen in der Variablen *strVorlage*. Wenn der Benutzer hier auf *Abbrechen* klickt, enthält *strVorlage* eine leere Zeichenkette. Nun folgen zwei *If...Then*-Bedingungen.

Die erste prüft, ob es bereits einen Datensatz mit dieser *Vorlagentextbezeichnung* in der Tabelle *tblVorlagentexte* gibt, und bricht gegebenenfalls mit einer entsprechenden Meldung ab. Die zweite *If...Then*-Bedingung prüft, ob *strVorlage* leer ist, was darauf hindeutet, dass der Benutzer entweder keine Bezeichnung eingegeben oder aber die *Abbrechen*-Schaltfläche betätigt hat. In beiden Fällen werden die innerhalb der *If...Then*-Bedingung enthaltenen Anweisungen nicht ausgeführt.

Hat der Benutzer jedoch eine gültige Bezeichnung eingegeben, legt die Prozedur zunächst einen neuen Datensatz mit der entsprechenden Bezeichnung in der Tabelle *tblVorlagen* an. Eine Anweisung später ermittelt sie den Autowert des zuletzt in dieser Session angelegten Datensatzes, also den Wert des Feldes *VorlagentextID* des neuen Datensatzes (*SELECT @@IDENTITY* liest den zuletzt angelegten Wert für ein Autowertfeld ein).

Diesen benötigt die Prozedur, um nach dem Aktualisieren des Inhalts des Listenfeldes *lstVorlagentexte* gleich den neuen Datensatz zu markieren. Im gleichen Zuge aktualisiert die Prozedur die Datenherkunft des Formulars und setzt den Datensatzzeiger auf den neuen Datensatz, der nun zwar bereits eine Bezeichnung, aber noch keinen Vorlagentext enthält. Schließlich verschiebt die Prozedur gleich den Fokus in das Textfeld *txtVorlagentext*, damit der Benutzer gleich mit der Bearbeitung beginnen kann.

Vorlagentext löschen

Ein Klick auf die Schaltfläche *cmdVorlageLoeschen* soll den aktuell im Listenfeld markierten Eintrag aus der Tabelle *tblVorlagentexte* löschen und die Steuerelemente entsprechend aktuali-

sieren. Dazu fragt die Prozedur zunächst mithilfe einer *MsgBox*-Anweisung ab, ob der Benutzer den Datensatz wirklich löschen möchte, und bricht den Vorgang gegebenenfalls ab.

Soll der Datensatz gelöscht werden, erledigt eine *DELETE*-Aktionsabfrage als Parameter der *Execute*-Methode des aktuellen *Database*-Objekts diesen Job. Danach aktualisiert die Prozedur den Inhalt des Listenfeldes und markiert den obersten Eintrag, der anschließend auch im Formular angezeigt wird. Dies geschieht jedoch nur, wenn die Tabelle *tblVorlagentexte* und somit auch das Listenfeld überhaupt noch einen Eintrag enthält.

Anderenfalls ruft die Prozedur noch die *Requery*-Methode des Formulars auf, um eventuell verbleibende *#Gelöscht*-Texte aus den Textfeldern zu verbannen (siehe Abbildung 8.16):

```
Private Sub cmdVorlageLoeschen_Click()
    Dim db As DAO.Database
    If MsgBox("Vorlage '" & Me!txtVorlageBezeichnung & "' wirklich löschen?", _
            vbYesNo, "Vorlage löschen") = vbYes Then
        Set db = CurrentDb
        db.Execute "DELETE FROM tblVorlagen WHERE VorlageID = " _
            & Me!VorlageID, dbFailOnError
        Me!lstVorlagen.Requery
        Me!lstVorlagen = Me!lstVorlagen.ItemData(0)
        If Not IsNull(Me!lstVorlagen) Then
            Me.Recordset.FindFirst "VorlageID = " & Me!lstVorlagen
        Else
            Me.Requery
        End If
        Set db = Nothing
    End If
End Sub
```

Abbildung 8.16: Diese Werte werden angezeigt, wenn der zugrunde liegende Datensatz per VBA gelöscht wird, ohne das Formular anschließend zu aktualisieren.

Steuerelemente aktivieren und deaktivieren

Wenn das Formular noch keinen Datensatz enthält, sollen die beiden Steuerelemente *txtVorlagentext* und *txtVorlagentextbezeichnung* deaktiviert sein. Diese Einstellung soll bei jedem

Datensatzwechseln erneut geprüft und gegebenenfalls korrigiert werden, was die folgende, beim Anzeigen eines jeden Datensatzes ausgelöste Prozedur erledigt:

```
Private Sub Form_Current()
    Me!txtVorlageInhalt.Enabled = Not IsNull(Me!VorlageID)
    Me!txtVorlageInhalt.Locked = IsNull(Me!VorlageID)
    Me!txtVorlageBezeichnung.Enabled = Not IsNull(Me!VorlageID)
    Me!txtVorlageBezeichnung.Locked = IsNull(Me!VorlageID)
End Sub
```

Formular schließen

Bevor wir uns um den eigentlichen Clou des Formulars kümmern, hier der Vollständigkeit halber noch die Prozedur, welche die Schaltfläche *cmdOK* beim Anklicken auslöst, um das Formular zu schließen:

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

Platzhalter einfügen

Sie können die Vorlagentexte mit Platzhaltern ausstatten, welche beim Einsatz der Vorlage als E-Mail an einen bestimmten Kunden mit den Daten aus einer speziellen Abfrage, hier *qryPlatzhalterKontextmenue*, gefüllt werden. Das heißt, dass ein Text wie

```
[Rechnung_Briefanrede]
bitte bestätigen Sie uns die Richtigkeit Ihrer E-Mail-Adresse [EMail].
```

mit dem folgenden Text ersetzt wird:

```
Sehr geehrter Herr Müller,
bitte bestätigen Sie uns die Richtigkeit Ihrer E-Mail-Adresse heinz@mueller.de.
```

Um einen entsprechenden Vorlagentext zu erstellen, muss der Benutzer alle möglichen Platzhalter kennen.

Da dieser jedoch keinen Zugriff auf die Tabellen und Abfragen hat, müssen Sie die Platzhalter auf anderem Wege zugänglich machen – und zwar auf einem möglichst ergonomischen Weg.

Eine praktische Lösung ist das Einfügen der Platzhalter über ein Kontextmenü, das alle möglichen Platzhalter anzeigt und diese nach dem Auswählen an der gewünschten Stelle einfügt – siehe auch Abbildung 8.17.

Dabei kann der Benutzer sowohl einfach die Einfügemarke an der entsprechenden Stelle platzieren und dort den Text einfügen, oder er kann einen Text markieren, den der Platzhalter überschreiben soll. In beiden Fällen wählt der Benutzer den Platzhalter über das Kontextmenü aus.

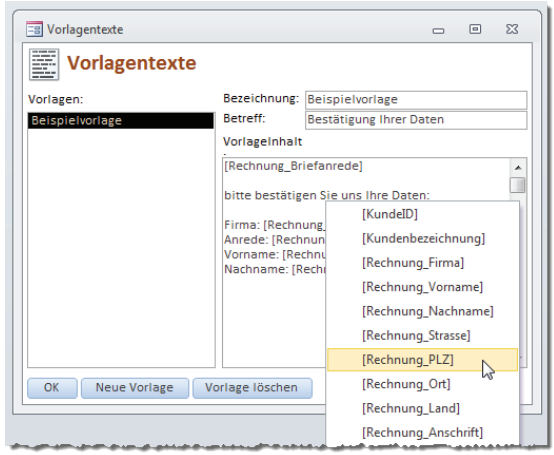


Abbildung 8.17: Einfügen von Platzhaltern per Kontextmenü

Die verwendete Technik erfordert nur zwei Prozeduren. Die erste wird beim Loslassen der rechten Maustaste ausgelöst, also durch das Ereignis *Bei Maustaste auf*. Hinterlegen Sie für diese Ereigniseigenschaft die folgende Prozedur:

```
Private Sub txtVorlageInhalt_MouseUp(Button As Integer, Shift As Integer, _
        X As Single, Y As Single)
    Dim db As DAO.Database
    Dim cbr As Office.Commandbar
    Dim cbc As Office.CommandBarControl
    Dim qdf As DAO.QueryDef
    Dim fld As DAO.Field
    If Not IsNull(Me!VorlageID) Then
        If Button = acRightButton Then
            Me.ShortcutMenu = False
            Set db = CurrentDb
            On Error Resume Next
            CommandBars("cbrPlatzhalter").Delete
            On Error GoTo 0
            Set cbr = CommandBars.Add("cbrPlatzhalter", msoBarPopup, , True)
            Set qdf = db.QueryDefs("qryPlatzhalter")
            For Each fld In qdf.Fields
                Set cbc = cbr.Controls.Add(msoControlButton, , , , True)
                cbc.Caption = "[" & fld.Name & "]"
                cbc.onAction = "=PlatzhalterHinzufuegen(""[" & fld.Name & "]"")"
            Next fld
            cbr.ShowPopup
```

```
        End If
    End If
End Sub
```

Die Prozedur prüft zunächst, ob das Formular überhaupt einen Datensatz anzeigt oder ob der Datensatzzeiger auf einem neuen, leeren Datensatz steht. In letzterem Fall wird die Prozedur beendet – genauso, wenn der Mausklick nicht durch die rechte Maustaste erfolgte (*Button = acRightButton*).

Hat der Benutzer jedoch die rechte Maustaste betätigt, schaltet die Prozedur zunächst das eingebaute Kontextmenü ab und löscht ein eventuell bereits vorhandenes Kontextmenü namens *cbrPlatzhalter* aus der *CommandBars*-Auflistung.

Dieses fügt die Prozedur gleich im Anschluss neu hinzu. Die Parameter legen fest, dass das Kontextmenü den Namen *cbrPlatzhalter* erhalten und als temporäres Kontextmenü eingerichtet werden soll. Danach öffnet die Prozedur die Struktur der Abfrage, welche die Felder für die Platzhalter liefern soll – hier *qryPlatzhalter*. Für jedes der enthaltenen *Field*-Elemente der Abfrage soll die Prozedur einen Eintrag zum Kontextmenü hinzufügen.

Dabei legt die Prozedur zunächst ein neues Element des Typs *msoControlButton* an, trägt als Bezeichnung den in eckigen Klammern eingefassten Namen des Feldes ein und fügt für die *onAction*-Eigenschaft einen Ausdruck wie *=PlatzhalterHinzufuegen("[Rechnung_Vorname]")* hinzu. Letzteres gibt die beim Anklicken des Kontextmenü-Eintrags aufzurufende VBA-Funktion samt Parameter an. Schließlich zeigt die *ShowPopup*-Methode des mit *cbr* referenzierten, frisch erstellten *Commandbar*-Objekts das Kontextmenü an.

Aktion bei Kontextmenü-Klick

Wenn der Benutzer einen der Kontextmenü-Einträge anklickt, löst dies die Funktion *PlatzhalterHinzufuegen* aus, welche den Namen des Platzhalters als Parameter erhält. Die Funktion ermittelt zunächst die aktuelle Markierung im Textfeld *txtVorlagentext*, wobei *SelStart* das erste Zeichen der Markierung und *SelLength* die Länge liefert. Befindet sich die Einfügemarke einfach nur im Text, ohne einen Teil davon zu markieren, hat *SelLength* den Wert *0*.

Die Variable *strVorlage* nimmt dann den kompletten Inhalt des Textfeldes auf und fügt den unberührten Teil vor Beginn der Markierung, den Platzhalter und den unberührten Teil hinter der Markierung zum neuen Text zusammen. Dieser bearbeitete Text wird zurück in das Textfeld geschrieben. Schließlich markiert die Funktion noch den hinzugefügten Platzhalter – siehe Abbildung 8.18.

```
Private Function PlatzhalterHinzufuegen(strName As String)
    Dim intSelStart As Integer
    Dim intSelLength As Integer
    Dim strVorlage As String
    intSelStart = Me!txtVorlageInhalt.SelStart
```

```
    intSelLength = Me!txtVorlageInhalt.SelLength
    strVorlage = Me!txtVorlageInhalt.Text
    strVorlage = Left(str, intSelStart) & strName & Mid(str, _
        intSelStart + 1 + intSelLength)
    Me!txtVorlageInhalt.Value = strVorlage
    Me!txtVorlageInhalt.SelStart = intSelStart
    Me!txtVorlageInhalt.SelLength = Len(strName)
End Function
```

Abbildung 8.18: Eingefügte Platzhalter werden markiert dargestellt.

8.2.6 Vorlagen bearbeiten

Fehlt noch eine Möglichkeit, diesen Dialog zu öffnen. Diesen legen wir im Ribbon an – mehr dazu siehe »Ribbons« (Seite 491).

8.2.7 Vorlage verwenden

Wenn Sie eine dieser Vorlagen verwenden möchten, wählen Sie diese mit dem Kombinationsfeld *cboVorlageID* aus, das Sie direkt neben der Schaltfläche zum Anlegen einer Standard-E-Mail einfügen. Rechts davon platzieren Sie die Schaltfläche *cmdEMailNachVorlage*, die eine E-Mail auf Basis der gewählten Vorlage erstellt (siehe Abbildung 8.19).

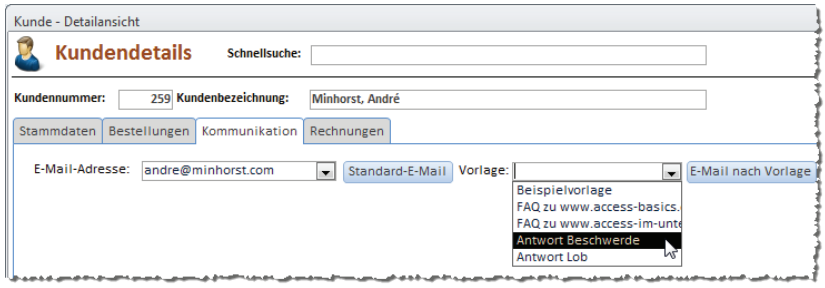


Abbildung 8.19: Auswählen einer Vorlage zum Erstellen einer E-Mail

Das Kombinationsfeld *cboVorlageID* verwendet den folgenden Ausdruck für die Eigenschaft *Datensatzherkunft*:

```
SELECT tblVorlagen.VorlageID, tblVorlagen.VorlageBezeichnung FROM tblVorlagen;
```

Damit nur die Bezeichnung angezeigt und das gebundene Feld ausgeblendet wird, stellen Sie die beiden Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *2* und *0cm* ein.

Die Schaltfläche *cmdEMailNachVorlage* löst schließlich die folgende Prozedur aus:

```
Private Sub cmdEMailNachVorlage_Click()
    EMailErstellen Me!cboEMailAdresseNeueMail.Column(1), Me!KundeID, Me!cboVorlageID
End Sub
```

Dies ist der zweite Aufruf der Funktion *EMailErstellen* (der erste wird weiter oben beschrieben und öffnet eine neue E-Mail unter Verwendung der Standardvorlage). Im vorliegenden Fall übergibt der Aufruf ebenfalls die gewählte Empfängeradresse und den Wert des Feldes KundeID. Zusätzlich weist die Prozedur dem dritten Parameter der Funktion *EMailErstellen* noch die *VorlageID* der mit dem Kombinationsfeld *cboVorlageID* gewählten Vorlage zu.

8.3 E-Mail erstellen und öffnen

Damit kommen wir nun auch zur Funktion *EMailErstellen*, die nicht direkt auf Outlook zugreift, sondern die dazu vorgesehene Klasse *clsMail* verwendet. Diese beschreibt das Kapitel »Outlook« (Seite 425) im Detail.

Grundsätzlich erstellt diese Prozedur E-Mails auf Basis zweier verschiedener Vorlagenarten – erstens der Standardvorlage mit den Daten aus den Feldern *TextStandardEMail* und *BetreffStandardEMail* aus der Tabelle *tblStammdaten* und zweitens mit den speziellen Vorlagen für verschiedene Zwecke aus den Feldern *VorlageBetreff* und *VorlageInhalt* aus der Tabelle *tblVorlagen*.

Woher weiß die Funktion *EMailErstellen* nun, ob die Vorlage nun aus der Tabelle *tblStammdaten* oder aus der Tabelle *tblVorlagen* stammt? Ganz einfach: Wenn die aufrufende Prozedur den Parameter *lngVorlageID* nicht füllt, soll die Funktion die Standardvorlage verwenden. Ist *lngVorlageID* hingegen gefüllt, soll die Vorlage mit der übergebenen ID verwendet werden.

Die Prozedur sieht wie folgt aus:

```
Public Function EMailErstellen(strEMail As String, lngKundeID As Long, _
        Optional lngVorlageID As Long)
    Dim objMail As clsMail
    Dim strInhalt As String
    Dim strBetreff As String
    Set objMail = New clsMail
```

```
    With objMail
        If lngVorlageID = 0 Then
            strBetreff = DLookup("BetreffStandardEmail", "tblStammdaten")
            strInhalt = DLookup("TextStandardEMail", "tblStammdaten")
        Else
            strBetreff = DLookup("VorlageBetreff", "tblVorlagen", _
                "VorlageID = " & lngVorlageID)
            strInhalt = DLookup("VorlageInhalt", "tblVorlagen", _
                "VorlageID = " & lngVorlageID)
        End If
        strBetreff = PlatzhalterErsetzen(strBetreff, "qryPlatzhalter", "KundeID", _
            lngKundeID)
        strBetreff = PlatzhalterErsetzen(strBetreff, "qryPlatzhalter", "KundeID", _
            lngKundeID)
        .Betreff = strBetreff
        strInhalt = PlatzhalterErsetzen(strInhalt, "qryPlatzhalter", "KundeID", _
            lngKundeID)
        strInhalt = PlatzhalterErsetzen(strInhalt, "qryPlatzhalter", "KundeID", _
            lngKundeID)
        .ToHinzufuegen strEMail
        .Inhalt = strInhalt
        .Anzeigen
        If .Gesendet = True Then
            EMailSpeichern lngKundeID, .Betreff, .Inhalt, Now, .Von, .An, .CC, .BCC
            Call TreeViewKommunikationAktualisieren
        End If
    End With
End Function
```

Sie deklariert und instanziert zunächst ein neues Objekt auf Basis der Klasse *clsMail*. Um diese drehen sich die folgenden Anweisungen, wobei die meisten schlicht Eigenschaften der E-Mail-Klasse füllen. Allerdings wertet die Prozedur zuvor noch den Wert des Parameters *lngVorlageID* aus. Ist dieser *0*, trägt die Prozedur die Daten aus der Tabelle *tblStammdaten* in die beiden Variablen *strBetreff* und *strInhalt* ein. Anderenfalls füllt sie die beiden Variablen mit den Daten der Tabelle *tblVorlagen* für den entsprechenden Wert des Primärschlüsselfeldes *VorlageID*.

Sowohl die Inhalte der Variablen *strBetreff* als auch *strInhalt* werden danach durch die Funktion *PlatzhalterErsetzen* gejagt. Diese ersetzt alle in eckigen Klammern eingefassten Feldnamen durch die in dem entsprechenden Datensatz enthaltenen Kundendaten. Als Datenquelle kommt die Abfrage *qryPlatzhalter* zum Einsatz, die alle notwendigen Felder enthält.

Schließlich füllt die Prozedur die Eigenschaften *Betreff* und *Inhalt* des Objekts *objMail* mit den Inhalten der Variablen *strBetreff* und *strInhalt*. Ein Aufruf der Methode *ToHinzufuegen* über-

9 Onlinebanking

Wer ein echter Access-Entwickler ist, der gibt sich natürlich nicht mit einer Homebanking-Software von der Stange ab. Nein, richtige Cracks bauen sich ihren Geldverwalter selbst – und natürlich mit Access. Was, Sie haben keine Idee, wie Sie per VBA auf Ihre Konten zugreifen können? Kein Problem: Dafür gibt es komplette Funktionssammlungen und gar ganze Frameworks – zum Beispiel die für diese Lösung verwendeten *DataDesign Banking Application Components* (*DDBAC*) der Firma *B+S Banksysteme AG*.

Und das Beste ist: Die benötigten Komponenten sind für den Privatgebrauch sogar kostenlos. Erst wenn Sie eine Anwendung entwickeln, welche mit DDBAC-Komponenten zusammenarbeitet, und Sie diese vertreiben, fallen Lizenzkosten an (nähere Informationen über die Konditionen erhalten Sie beim Hersteller selbst).

Vorerst aber schauen Sie sich anhand unserer Onlinebanking-Lösung an, wie das Ganze funktioniert und wie Sie zum Beispiel Ihren Kontostand abfragen, die Umsätze einlesen oder gar Überweisungen tätigen.

Dieses Kapitel ist etwas VBA-lastig, weil wir ausschließlich per Code auf Kontostände oder Umsatzdaten zugreifen oder auch Überweisungen tätigen können. Nachdem die diesbezüglichen Möglichkeiten erschlossen wurden, kümmern wir uns darum, wie Sie die gewonnenen Daten in Tabellen speichern und in Formularen angezeigen können.

In »Umsätze anzeigen und zuordnen« (Seite 247) finden Sie bereits ein Formular, mit dem Sie die per Onlinebanking eingelesenen Umsätze den Bestellungen beziehungsweise Rechnungen zuweisen können.

9.1 Voraussetzungen

Die vorliegende Lösung setzt die Installation der *DDBAC*-Komponenten der Firma *DataDesign* voraus. Diese können Sie in der aktuellen Fassung unter folgendem Link herunterladen: *http://www.ddbac.de/Dev.SDK.shtml*. Sie benötigen das *DDBAC Software Development Kit* in der jeweils aktuellen Version. Nach dem Herunterladen und dem Installieren wäre dieser Teil der Vorbereitungen schon abgehakt. Bitte vergessen Sie nicht, die Lizenzbedingungen sorgfältig zu lesen.

Eine weitere Voraussetzung ist ein Bankkonto, dessen HBCI-Funktionen freigeschaltet sind, und ein Satz von Informationen, die Sie für den Einsatz der DDBAC-Komponenten benötigen. Details hierzu finden Sie beim Kreditinstitut Ihres Vertrauens – aber wahrscheinlich sind Sie ohnehin schon längst online unterwegs, was Ihre Bankgeschäfte angeht.

Die Installation des DDBAC-Pakets hat eine ganze Reihe Dateien auf Ihre Festplatte geschaufelt, die Sie sich über das Startmenü von Windows zugänglich machen können. Dort finden Sie unter

anderem eine Reihe Beispiele und haufenweise Dokumentation. Welche Informationen für Sie interessant sein können, zeigen wir Ihnen im Laufe dieses Kapitels.

9.2 Homebanking-Kontakt erstellen

Funktionell betrachtet brauchen Sie diese ganzen Dateien jedoch erstmal nicht. Was Sie brauchen, ist ein sogenannter Homebanking-Kontakt, der einem Benutzerkonto bei einem Kreditinstitut entspricht. Das ist keinesfalls identisch mit einem Bankkonto – auch wenn es möglicherweise später den Anschein hat, weil die Nummern von Benutzerkonto und Bankkonto Ähnlichkeiten aufweisen oder gar gleich sind.

Das Benutzerkonto entspricht vielmehr dem, was Sie auch vorfinden, wenn Sie sich über die Internetseite Ihrer Bank einloggen, um dort etwa Kontostände abzurufen oder Überweisungen vorzunehmen. In vielen Fällen scheinen Benutzerkonto und Bankkonto identisch, wenn Sie nur ein Konto bei der Bank eingerichtet haben – etwa ein Girokonto.

Wenn Sie noch weitere Konten, Depots oder Sparkonten bei dieser Bank hätten, würden diese jedoch wahrscheinlich online über das gleiche Benutzerkonto zugänglich sein.

Und den Zugriff auf ein solches Benutzerkonto müssen Sie nun einrichten, wenn Sie mit Access auf Ihre Bankkonten zugreifen möchten. Die nötigen Daten erhalten Sie von Ihrem Kreditinstitut, den Rest erledigt der *Administrator für Homebanking Kontakte*. Den finden Sie nicht etwa im Startmenü bei den übrigen Dateien der DDBAC-Installation, sondern gut untergebracht in der Systemsteuerung (siehe Abbildung 9.1).

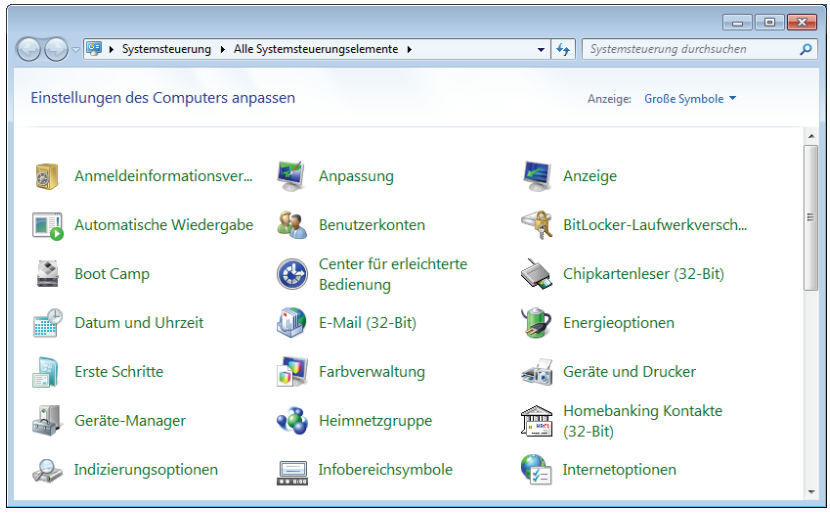


Abbildung 9.1: Über den Eintrag *Homebanking Kontakte (32-bit)* in der Systemsteuerung bereiten Sie den Zugriff auf Ihre Kontodaten vor.

Mit einem Doppelklick auf diesen Eintrag öffnen Sie den Dialog aus Abbildung 9.2. Er enthält zunächst keine Einträge, was wir aber in den nächsten Schritten ändern werden – sofern Ihnen die notwendigen Daten bereits vorliegen. Das könnte zumindest der Fall sein, wenn Sie schon mit einer anderen Homebanking-Software arbeiten. Klicken Sie hier auf die Schaltfläche *Neu*, um einen neuen Kontakt anzulegen (der wiederum einem Benutzerkonto entspricht):

Abbildung 9.2: Ein Banking-Kontakt ist prinzipiell mit einem Konto bei einem Kreditinstitut identisch.

Der erste Schritt nach dem Willkommen-Dialog ist ganz einfach. Hier geben Sie einfach nur die Bankleitzahl Ihres Kreditinstituts ein (siehe Abbildung 9.3).

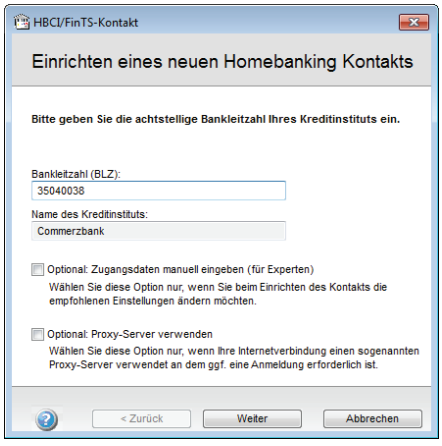


Abbildung 9.3: Als Erstes benötigt der Assistent die Bankleitzahl Ihres Kreditinstituts.

Dann ist erstmal der Assistent gefragt, der die möglichen Zugangsarten für den Zugriff auf die Konten dieser Bank ermittelt (siehe Abbildung 9.4).

Abbildung 9.4: Anschließend prüft der Assistent die für das angegebene Kreditinstitut verfügbaren Zugangsarten.

Dabei gibt es verschiedene Varianten:

» PIN/TAN: Die klassische Variante und von einigen Banken schon nicht mehr unterstützte Variante. Sie erfordert keine weiteren Hilfsmittel außer einer PIN-Nummer, mit der Sie sich bei der Bank anmelden, und eine Liste mit TANs (TransAktion-Nummer). Wenn Sie während einer Sitzung, für die Sie sich mit Ihrer PIN-Nummer angemeldet haben, Aktionen wie etwa eine Überweisung vornehmen möchten, brauchen Sie eine der auf der Liste enthaltenen TANs. Diese veraltete, aber wegen der geringen technischen Anforderungen auf Client-Seite immer noch sehr verbreitete Methode werden Sie vermutlich bereits einmal durchgeführt haben.

» Chipkarte: Hierbei erhalten Sie von Ihrem Kreditinstitut einen Chipkartenleser und eine entsprechende Chipkarte, die für die Signatur verschlüsselter Nachrichten verwendet wird. Wichtig ist hier die Authentifizierung, also der Vorgang, bei dem sichergestellt wird, dass sich auch der entsprechende Kunde mit der Chipkarte anmeldet. Die Authentifizierung besteht hier schlicht darin, dass die Bank dem Kunden die Karte übergibt.

» Ähnlich sieht es bei der Verwendung einer Schlüsseldatei aus: Auch diese hilft bei der Verschlüsselung der Nachrichten. Allerdings erfolgt die Authentifizierung hier dadurch, dass der Kunde den öffentlichen Part eines durch die notwendige Software erzeugten Schlüsselpaars gleichzeitig auf elektronischem als auch auf postalischem Wege an die Bank schickt. Ein erfolgreicher Vergleich dieser beiden Schlüssel entspricht der Authentifizierung.

Wenn sichergestellt ist, dass nur der Benutzer die Chipkarte beziehungsweise den privaten Teil des Schlüsselpaares besitzt (dafür ist letztlich der Benutzer verantwortlich), kann dieser seine Nachrichten damit verschlüsseln und somit signieren.

Der Dialog aus Abbildung 9.5 stellt die möglichen Zugangsarten zur Auswahl.

Abbildung 9.5: In diesem Fall ist der Zugang per PIN/TAN, Chipkarte und per Schlüsseldatei möglich.

Wir wählen zu Beispielzwecken den Eintrag *Schlüsseldatei* aus. Im folgenden Schritt geben Sie dann an, ob Sie bereits eine Schlüsseldatei besitzen oder nicht (siehe Abbildung 9.6).

Falls nicht, startet der oben beschriebene Vorgang – es wird ein Schlüsselpaar erzeugt und durch die Zusendung des öffentlichen Schlüssels auf zwei Wegen an die Bank wird dieser als Ihr Schlüssel authentifiziert.

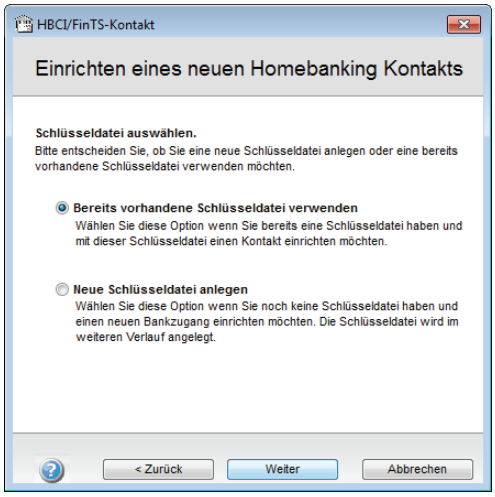


Abbildung 9.6: Möglicherweise verfügen Sie durch den Einsatz einer alternativen Homebanking-Software bereits über eine Schlüsseldatei – sonst müssen Sie eine neue anlegen.

Davon ausgehend, dass bereits eine Schlüsseldatei vorliegt, geben Sie den Speicherort dieser Datei an (siehe Abbildung 9.7). Stellen Sie sicher, dass das Medium mit dieser Datei (heutzutage wohl eher ein USB-Stick als eine Diskette) an einem geschützten Ort untergebracht ist.

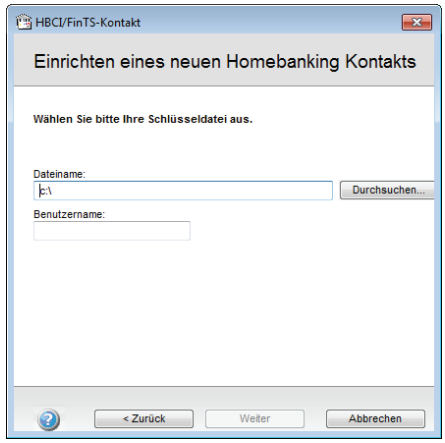


Abbildung 9.7: Falls eine Schlüsseldatei vorhanden ist, geben Sie ihren Speicherort hier an.

Zum Anlegen des Kontakts fehlt nun noch die Angabe der Passphrase, die Sie beim Erstellen des Schlüsselpaars angegeben haben. Diese geben Sie im Dialog aus Abbildung 9.8 ein.

Abbildung 9.8: Wenn Sie bereits eine Schlüsseldatei besitzen, brauchen Sie nur noch das Passwort einzugeben.

Es fehlen noch ein paar Kundendaten, die der Dialog aus Abbildung 9.9 abfragt.

Gleich im Anschluss nimmt der Assistent nochmals Kontakt zum Server des Kreditinstituts auf und ruft die letzten erforderlichen Informationen ab (siehe Abbildung 9.10).

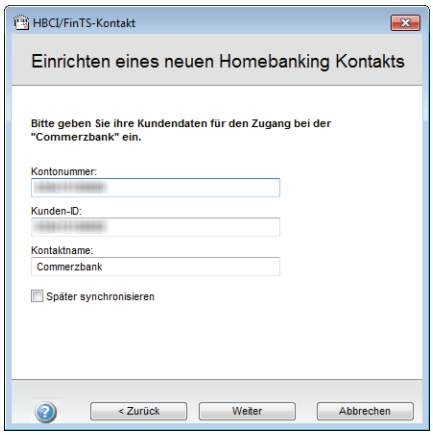


Abbildung 9.9: Jetzt nur noch schnell die Kundendaten eingeben, die Sie ebenfalls von Ihrer Bank erhalten ...

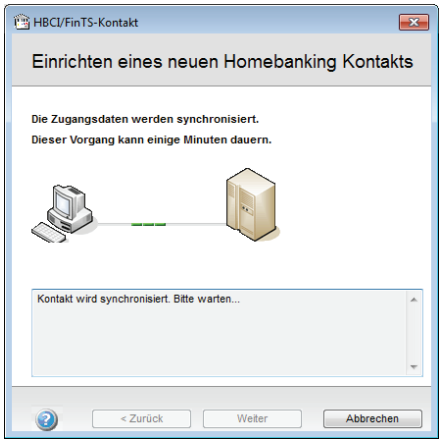


Abbildung 9.10: ... und synchronisieren – fertig!

Mit diesen im Gepäck erstellt er schließlich den neuen Eintrag in der Liste der Bank-Kontakte. Wenn Sie diesen Kontakt markieren, können Sie ihn neu synchronisieren (was von Zeit zu Zeit nötig sein kann, aber auch per Code durchführbar ist), ihn bearbeiten oder auch entfernen (siehe Abbildung 9.11). Natürlich können Sie hier auch noch weitere Kontakte anlegen – vielleicht haben Sie ja etwa sowohl ein privates als auch ein Geschäftskonto:

9.3 Funktionen der Lösung

Damit Sie wissen, worauf dieses Kapitel hinausläuft, wollen wir kurz die geplanten Funktionen vorstellen:

» Einlesen des aktuellen Kontostands

» Einlesen der Umsätze der letzten maximal 90 Tage

» Speichern der Umsatzdaten in einer Tabelle

» Ausführen von Inlandsüberweisungen inklusive Formular zur Eingabe der Überweisungsdaten

» Speichern der Daten der Überweisungsempfänger für weitere Vorgänge

» Anzeige der Überweisungen in einer Übersicht und als Detailansicht

» Anzeige der Umsätze in einer Übersicht und als Detailansicht

Desweiteren soll die Lösung die Möglichkeit bieten, die Umsätze den Rechnungsdatensätzen zuzuweisen, um diese gleichzeitig als bezahlt zu markieren.

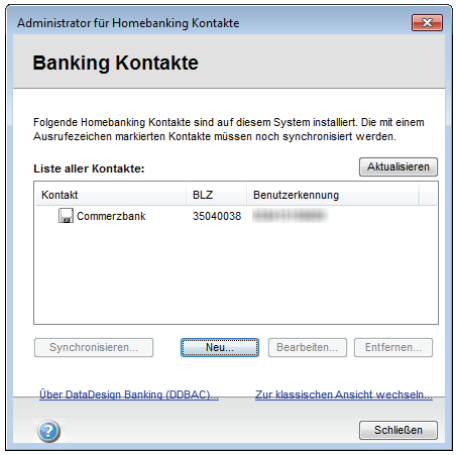


Abbildung 9.11: Anschließend finden Sie im *Administrator für Homebanking Kontakte* einen ersten Eintrag, dem Sie bei Bedarf noch weitere hinzufügen können.

9.4 Kontakt herstellen

Wenn Sie das DDBAC Software Development Kit installiert und einen oder mehrere Kontakte eingerichtet haben, können Sie gleich per VBA auf die eingerichteten Daten zugreifen. Das ist wichtig, weil Sie ja im Kontext eines bestimmten Benutzers und eines Bankkontos arbeiten möchten, um etwa Kontostände oder Umsätze einzulesen oder auch eine Überweisung zu tätigen.

Sie müssen also zunächst einmal das Benutzerkonto (später abgebildet durch das *Contact*-Objekt) und das Bankkonto (entspricht dem *Account*-Objekt) auswählen, mit dem Sie arbeiten

möchten. Grundlage dafür ist ein Formular, das zwei abhängige Formulare zur Auswahl der gewünschten Einträge anbietet.

Erstellen Sie also ein neues, leeres Formular, speichern Sie es unter dem Namen *frmHomebanking* und öffnen Sie es in der Entwurfsansicht. Fügen Sie nun wie in Abbildung 9.12 zwei Kombinationsfelder hinzu, deren Beschriftungen *Bankverbindung* und *Konto/Depot* lauten und die *cboContacts* und *cboAccounts* heißen (wir weichen hier ausnahmsweise von der sonst in diesem Buch üblichen deutschen Benennung von Steuerelementen und Variablen ab, um die Benennung der verwendeten DDBAC-Komponente weiterzuverwenden und nicht allzuviel Durcheinander zu erzeugen).

Eine Datenherkunft für das Formular oder Datensatzherkünfte für die Kombinationsfelder in Form einer Tabelle oder Abfrage brauchen wir vorerst nicht. Das Formular zeigt die gewohnten Daten später in einem Unterformular an, und die Kombinationsfelder füllen wir dynamisch mit VBA-Funktionen, welche die mit dem DDBAC-Assistenten erstellten Bankverbindungen und Konten auslesen.

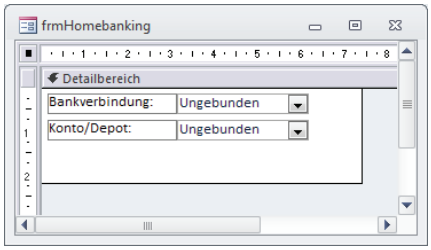


Abbildung 9.12: Das Formular zur Auswahl der Bankverbindung im Entwurf

Beim ersten Öffnen des Formulars soll dieses automatisch die Datensatzherkunft des Kombinationsfeldes *cboContacts* füllen. Nach der Auswahl eines Eintrags durch Benutzer trägt eine entsprechende Prozedur dann die Kontos/Depots dieser Bankverbindung in die Datensatzherkunft des zweiten Kombinationsfelds *cboAccounts* ein.

9.4.1 Bankverbindungen einlesen

Als Erstes lesen wir die Bankverbindungen ein und schreiben die resultierende Liste als Datensatzherkunft in die entsprechende Eigenschaft des Kombinationsfeldes *cboContacts*. Dies erledigen wir innerhalb der Ereignisprozedur, die durch die Ereigniseigenschaft *Beim Laden* des Formulars ausgelöst wird. Diese sieht zunächst ganz einfach so aus:

```
Private Sub Form_Load()
    Me!cboContacts.RowSource = GetContacts
End Sub
```

Nun fehlt natürlich noch die Funktion *GetContacts*, die wir selbst anlegen müssen.

Für diese und alle weiteren für den Zugriff auf die DDBAC-Komponenten nötigen Prozeduren legen wir ein eigenes Standardmodul namens *mdlHBCI* an. Um die DDBAC-Objekte zu nutzen, setzen Sie über den Verweise-Dialog (VBA-Editor, Menüeintrag *Extras|Verweise*) zunächst einen Verweis auf die Bibliothek *DataDesign DDBAC HBCI Banking Application Components*, die Sie durch die oben beschriebene Installation zum System hinzugefügt haben (siehe Abbildung 9.13).

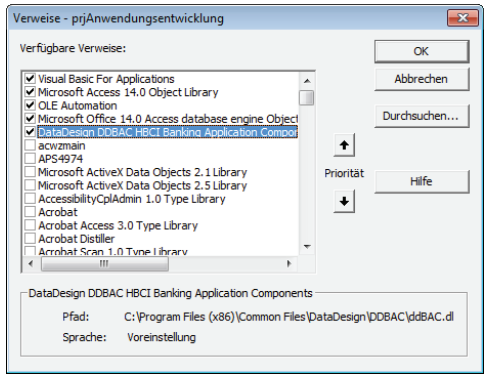


Abbildung 9.13: Verweis auf die DDBAC-Bibliothek

Für den Zugriff auf die Bibliothek benötigen wir zunächst eine Objektvariable, die einen Verweis auf ein Objekt des Typs *BACContacts* erhalten soll. Dieses macht die einzelnen *BACContact*-Objekte verfügbar. Ein *BACContact*-Objekt entspricht einem Kontakt, den Sie mit dem oben beschriebenen DDBAC-Assistenten angelegt haben.

Da Sie normalerweise mehrmals während einer Session auf die *BACContacts*-Auflistung zugreifen werden, gestalten wir den Zugriff darauf so, dass Sie dieses beim ersten Zugriff erzeugen und bei jedem weiteren Zugriff auf diese Instanz zugreifen können – bis diese beabsichtigt oder unbeabsichtigt (etwa durch einen unbehandelten Laufzeitfehler) gelöscht wird. In diesem Fall erhalten Sie einfach einen Verweis auf eine neue Instanz.

Im Modul *mdlHBCI* deklarieren Sie zunächst eine private Variable namens *m_Contacts*, welche gleich mit dem Objektverweis gefüllt wird:

```
Private m_Contacts As BACContacts
```

Nun können Sie natürlich nicht von überall auf diese private deklarierte Variable zugreifen, weshalb wir nun eine Funktion zum Modul hinzufügen, die den Zugriff auf den in der Variable gespeicherten Verweis ermöglicht.

Diese Funktion prüft, ob *m_Contacts* bereits einmal gefüllt wurde, und liefert dann den bestehenden Objektverweis zurück oder sie erstellt einfach ein neues *BACContacts*-Objekt und reicht dieses dann weiter. Vorher füllt die Routine die Auflistung noch mit der *Populate*-Methode (leere Zeichenkette als Parameter nicht vergessen!), damit jederzeit der aktuelle Stand zurückgeliefert wird:

```
Public Function objContacts() As BACContacts
    If m_Contacts Is Nothing Then
        Set m_Contacts = New BACContacts
    End If
    m_Contacts.Populate ""
    Set objContacts = m_Contacts
End Function
```

Der Clou hierbei ist, dass Sie somit keine globale Variable haben, die versehentlich gelöscht werden kann. Die folgende Funktion namens *GetContacts* schließlich liefert die Zeichenkette, die Sie als Datensatzherkunft des Kombinationsfelds *cboContact* verwenden können.

Damit dieses eine Zeichenkette als Datensatzherkunft akzeptiert, müssen Sie noch die Eigenschaft *Herkunftsart* auf den Wert *Wertliste* einstellen.

Alternativ können Sie auch die folgende Zeile als erste Anweisung in die Prozedur *Form_Load* des Formulars *frmHomebanking* integrieren (das Gleiche sollten Sie dann auch für das Kombinationsfeld *cboAccounts* erledigen):

```
Me!cboContacts.RowSourceType = "Value List"
```

Stellen Sie gegebenenfalls auch noch die Eigenschaft *Wertlistenbearbeitung zulassen* auf *Nein* ein. Die Funktion *GetContacts* liefert eine Zeichenkette wie die folgende zurück – hier mit nur einem Bank-Kontakt:

```
0;1234567890;Commerzbank|12121212|1234567890;
```

Das Zusammenstellen dieser Wertliste erledigt die folgende Funktion:

```
Public Function GetContacts() As String
    Dim objContact As BACContact
    Dim strContacts As String
    Dim i As Integer
    For Each objContact In objContacts
        strContacts = strContacts & i & ";" & objContact.UserID & ";" _
            & objContact("Contact") _
            & "|" & objContact.BankCode
            & "|" & objContact.UserID & ";"
        i = i + 1
    Next objContact
    GetContacts = strContacts
End Function
```

Die Prozedur deklariert zunächst ein Objekt namens *objContacts*, das einen Bank-Kontakt repräsentiert. Die Zeichenkette *strContacts* soll den späteren Rückgabewert zwischenspeichern, und *i* ist eine Zählervariable.

Danach durchläuft die Prozedur die Elemente der *objContacts*-Auflistung und schreibt diese in eine Zeichenkette, die zuerst den Wert der Zählervariable *i*, dann ein Semikolon, einmal das Benutzerkonto allein und schließlich die drei Informationen *Bankname*, *Bankleitzahl* und *Benutzerkonto* aneinanderhängt. Die hinteren drei Informationen werden dabei durch das Pipe-Zeichen (|) voneinander getrennt. Wenn mehr als ein Kontakt vorliegt, werden die übrigen unter Angabe einer jeweils um eins erhöhten Zählervariablen an die Zeichenkette angehängt. Die Kundennummer, die an zweiter Positon der Semikola-separierten Liste gespeichert wird, brauchen wir erst später.

Die letzte Anweisung schreibt das Ergebnis schließlich in den Rückgabewert der Funktion, damit dieser auch der Eigenschaft *RowSource* des Kombinationsfeldes zugewiesen werden kann.

Damit das Kombinationsfeld jeweils den Wert der Zählervariablen (also den Wert vor dem Semikolon) als gebundenen, nicht sichtbaren Wert, das Benutzerkonto allein ebenfalls unsichtbar und die durch die Pipe-Zeichen voneinander getrennten Informationen als sichtbaren Wert anzeigt, stellen Sie noch die Eigenschaften *Spaltenanzahl* und *Spaltenbreiten* auf die Werte *3* und *0cm;0cm* ein (Gleiches gilt für das Kombinationsfeld *cboAccounts*). Die dritte Spalte wird so über die gesamte Breite des Kombinationsfelds angezeigt, die dritte erscheint wie die erste gar nicht.

Wenn Sie das Formular nun in der Formularansicht öffnen und das Kombinationsfeld aufklappen, finden Sie etwa die Ansicht aus Abbildung 9.14 vor.

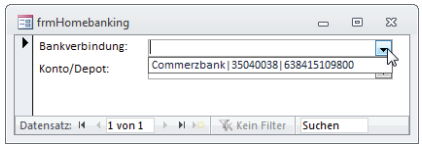


Abbildung 9.14: Auswahl des bislang einzigen Bank-Kontakts per Kombinationsfeld

Nun wollen wir zunächst dafür sorgen, dass der Inhalt des zweiten Kombinationsfeldes in Abhängigkeit von dem im ersten Kombinationsfeld ausgewählten Bank-Kontakt gefüllt wird. Dazu kehren wir zunächst wieder zum Modul *mdlHBCI* zurück. Wir brauchen eine Funktion, die in Abhängigkeit vom ausgewählten Kontakt die entsprechenden Konten und Depots zurückliefert. Diese arbeitet prinzipiell genau so wie die Funktion *GetContacts* und sieht wie folgt aus:

```
Public Function GetAccounts(intAccount As Integer) As String
    Dim objContact As BACContact
    Dim objAccount As BACAccount
    Dim strAccounts As String
    Dim i As Integer
    Set objContact = objContacts.Item(intAccount)
    For Each objAccount In objContact.Accounts
        strAccounts = strAccounts & i & ";" _
            & objAccount.AccountNumber & ";" _
```

```
            & objAccount.AccountNumber _
            & "|" & objAccount.AcctName & ";"
        i = i + 1
    Next objAccount
    GetAccounts = strAccounts
End Function
```

Die Funktion erwartet allerdings einen Parameter, und zwar den Index des zu durchsuchenden Kontakts. Dieser ist nullbasiert, entspricht also genau den Werten der ersten Spalte der Datensatzherkunft des Kombinationsfeldes *cboContacts* (welch ein Zufall!) und wird gleich der *Item*-Eigenschaft der *objContacts*-Auflistung als Parameter übergeben. Dies wird, wie bereits in der Funktion *GetContacts*, ständig aktuell über die Funktion *objContacts* bezogen.

Die Objektvariable *objContact* nimmt schließlich den per Parameterwert identifizierten Eintrag der *objContacts*-Auflistung auf. Elemente vom Typ *BACContact* enthalten wiederum eine Auflistung namens *Accounts*, die wir im Folgenden durchlaufen und dabei einige Informationen zu einer Zeichenkette zusammenstellen. Diese sieht beispielsweise so aus:

```
0;121212300|Kontokorrent;1;121212390|Spar;2;121212300|Depot;
```

Dies soll im Kombinationsfeld *cboAccounts* des Formulars *frmHomebanking* mal drei Einträge ergeben, wobei die Indexwerte *0*, *1* und *2* wieder in der unsichtbaren erste Spalte landen und die folgenden Ausdrücke (etwa *121212300|Kontokorrent*) als angezeigter Wert dienen. Damit *cboAccounts* nach der Auswahl in *cboContacts* aktualisiert wird, legen Sie für das Ereignis *Nach Aktualisierung* die folgende Ereignisprozedur an:

```
Private Sub cboContacts_AfterUpdate()
    If Not IsNull(Me!cboContacts) Then
        Me!cboAccounts.RowSource = GetAccounts(Me!cboContacts)
    Else
        Me!cboAccounts.RowSource = ""
    End If
End Sub
```

Das Ergebnis sieht dann wie in Abbildung 9.15 aus.

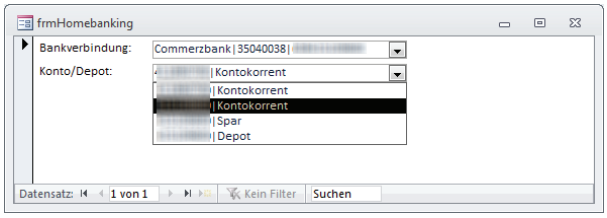


Abbildung 9.15: Nach der Auswahl eines Bank-Kontakts offeriert das Formular auch dessen Konten und Depots.

9.4.2 Letzten Contact und Account speichern

Ein kleines Feature des Formulars soll sein, dass der zuletzt verwendete Bank-Kontakt und das letzte Konto oder Depot gespeichert werden, damit das Formular diese beim nächsten Öffnen gleich wieder herstellen kann.

Dazu sollen die relevanten Informationen automatisch in der bereits vorhandenen Optionstabelle gespeichert werden, damit der Benutzer beim erneuten Öffnen gleich auf die Bankverbindung und die Kontodaten zugreifen kann, die beim letzten Zugriff verwendet wurden.

Dazu fügen Sie zur Tabelle *tblOptionen* zwei weitere Felder hinzu, und zwar *LetzterBankkontakt* und *LetztesKonto* (siehe Abbildung 9.16).

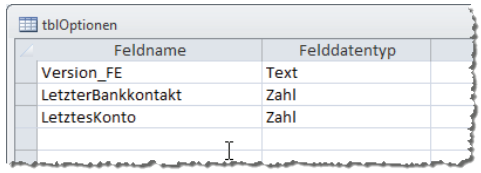


Abbildung 9.16: Erweiterung der Tabelle *tblOptionen* zum Speichern der zuletzt verwendeten Bankdaten

Nun brauchen wir eine *OK*-Schaltfläche, die das Formular schließt und vorher die verwendete Konfiguration in der Tabelle *tblOptionen* sichert. Diese Schaltfläche heißt *cmdOK* und soll für die Ereigniseigenschaft *Beim Klicken* die folgende Ereignisprozedur erhalten:

```
Private Sub cmdOK_Click()
    Dim db As DAO.Database
    Set db = CurrentDb
    If Not IsNull(Me!cboContacts) Then
        db.Execute "UPDATE tblOptionen SET LetzterBankkontakt = '" & Me!cboContacts _
            & "'", dbFailOnError
        If Not IsNull(Me!cboAccounts) Then
            db.Execute "UPDATE tblOptionen SET LetztesKonto = '" & Me!cboAccounts _
                & "'", dbFailOnError
        End If
    End If
    DoCmd.Close acForm, Me.Name
End Sub
```

Nachdem wir das Formular ein wenig angehübscht haben, sieht dieses wie in Abbildung 9.17 aus. Dazu haben wir die Beschriftung des Formulars auf *Online-Banking* eingestellt, die Eigenschaften *Navigationsschaltflächen*, *Bildlaufleisten*, *Trennlinien* und *Datensatzmarkierer* auf *Nein* und *Automatisch zentrieren* auf *Ja* eingestellt. Außerdem enthält der Formularkopf nun ein Bildsteuerelement mit einem hübschen Icon und eine entsprechende Überschrift.

10 Individuelle Anschreiben mit Word

Hin und wieder wird vorkommen, dass Sie einem Kunden ein individuelles Anschreiben zukommen lassen möchten. Normalerweise sollte das zwar in Zeiten der E-Mail nicht vorkommen, aber es kann ja beispielsweise einmal sein, dass der Kunde eine falsche E-Mail-Adresse angegeben hat und Sie ihn deshalb nicht erreichen können. Für diesen Fall ist ein schnell auf Basis einer Vorlage gefülltes, ausgedrucktes und verschicktes Word-Dokument doch genau die richtige Lösung.

10.1 Funktionsweise

Dieser Teil der Anwendung liefert ein Formular, das als Schaltfläche für die Erstellung von (halb) individuellen Word-Dokumenten dient. Will meinen: Sie werden zumindest eine Vorlage anlegen, welche beispielsweise die Anschrift des Kunden als Platzhalter enthält.

Sie können mit der hier vorgestellten Lösung auch noch einen Schritt weitergehen und Word-Dokumente für spezielle Zwecke vorbereiten: Diese enthalten dann bereits den vollständigen Text mit Platzhaltern, die in Abhängigkeit vom Kunden gefüllt werden sollen.

Und es kommt noch besser: Sie können wahlweise Dokumente erstellen, die Platzhalter nicht nur mit Kundendaten füllen, sondern auch noch mit den Daten einer bestimmten Bestellung. Auf diese Weise können Sie dem Kunden etwa ein Schreiben schicken, mit dem Sie sich für die Lieferung eines falschen oder kaputten Artikels entschuldigen, und dabei direkt die Bestellnummer oder das Bestelldatum in Form eines Platzhalters verwenden.

Wichtig ist an dieser Stelle, dass wir nicht von klassischen Word-Vorlagen sprechen, sondern von einfachen Word-Dokumenten, die schlicht Platzhalter in der Form *[Platzhalter]* enthalten.

10.1.1 Ablauf beim Erstellen individueller Dokumente

Die Erstellung eines individuellen Dokuments können Sie beispielsweise durch eine Schaltfläche auf der Registerseite *Kommunikation* des Formulars *frmKundeDetail* starten (siehe Abbildung 10.1).

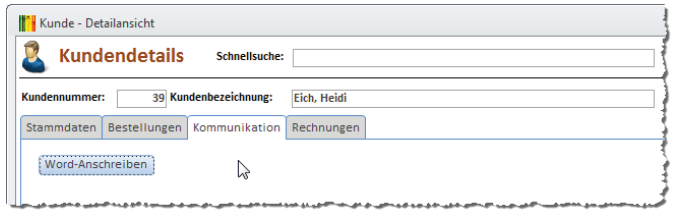


Abbildung 10.1: Aufruf des Dialogs zum Erstellen eines individuellen Anschreibens

Ein Klick auf diese Schaltfläche öffnet das Formular aus Abbildung 10.2. Es teilt den Ablauf in zwei Schritte ein:

» Vorbereiten des zu füllenden Word-Dokuments

» Erstellen des individuellen Anschreibens auf Basis des vorbereiteten Dokuments und der für die Platzhalter einzufügenden Daten

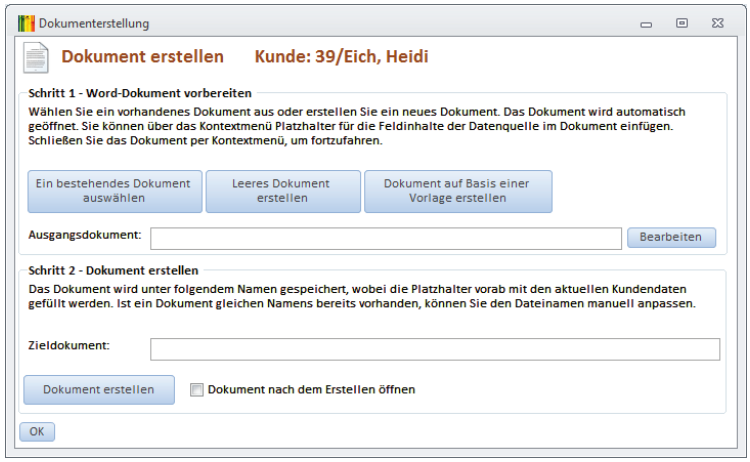


Abbildung 10.2: Schaltzentrale zum Erstellen von Word-Anschreiben

10.1.2 Word-Dokument vorbereiten

Für die Vorbereitung gibt es drei Möglichkeiten:

» Sie wählen ein bestehendes Word-Dokument aus, das Sie bereits zuvor für die Verwendung mit dieser Anwendung vorbereitet haben und das bereits über entsprechende Platzhalter verfügt.

» Sie erstellen ein neues, leeres Word-Dokument.

10.1.3 Vorlage mit Daten füllen

Wir zeigen beispielhaft, wie Sie ein leeres Dokument erstellen und mit Platzhaltern füllen. Die Platzhalter sind in eckige Klammern eingefasste Feldnamen. Um ein neues Dokument zu erstellen, klicken Sie einfach auf die Schaltfläche *Leeres Dokument erstellen*. Word wird geöffnet und zeigt ein leeres Dokument an.

Wir könnten nun eine einfache Lösung bauen, bei welcher der Benutzer die Platzhalter selbst eintragen muss, indem er die entsprechenden Feldnamen in eckige Klammern einfasst und diese im Dokument anlegt. Aber wir wollen ja eine richtig coole Anwendung erstellen, die dem

Anwender größtmöglichen Komfort bietet – und deshalb stellen wir alle zur Verfügung stehenden Platzhalter per Kontextmenü bereit (siehe Abbildung 10.3). Mit der Auswahl eines der Platzhalter wird dieser an der aktuellen Position der Einfügemarke eingefügt.

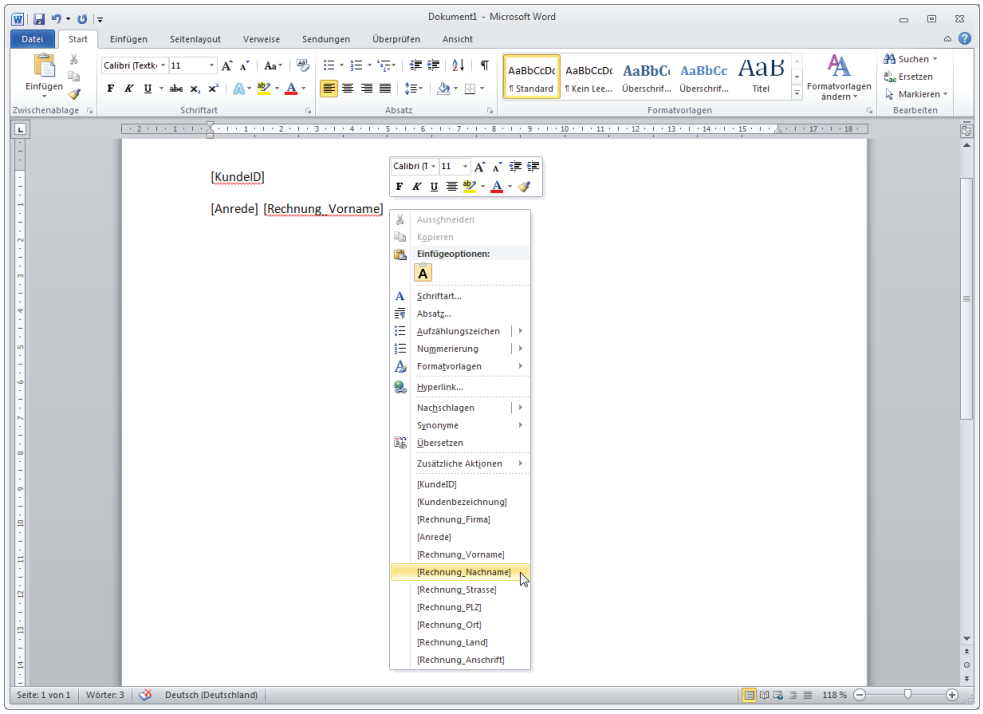


Abbildung 10.3: Hinzufügen von Platzhaltern

Um das mit Platzhaltern ausgestattete Dokument zu speichern, brauchen Sie es einfach nur zu schließen. Es erscheint dann eine *Inputbox*, mit der Sie den Dateinamen ohne Dateiendung angeben (siehe Abbildung 10.4). Das Verzeichnis zum Speichern der Vorlagen ist in der Tabelle *tblOptionen* gespeichert.

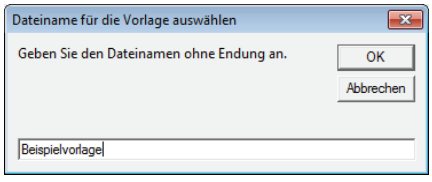


Abbildung 10.4: Eingabe des Dateinamens für die Vorlage

Das Verzeichnis der Vorlagendatei wird im Formular *frmWordAnschreiben* schließlich unter *Ausgangsdokument* eingetragen (siehe Abbildung 10.5).

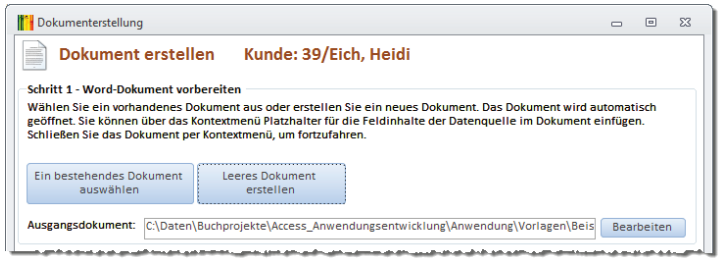


Abbildung 10.5: Anzeige der erstellten oder ausgewählten Vorlagendatei

Dies geschieht auch, wenn Sie mit der Schaltfläche *Ein bestehendes Dokument auswählen* den Dateiauswahl-Dialog aus Abbildung 10.6 öffnen und damit eine der Vorlagen auswählen. Als Startverzeichnis wird hier das im Feld *PfadVorlagenAnschreiben* der Tabelle *tblOptionen* gespeicherte Verzeichnis verwendet.

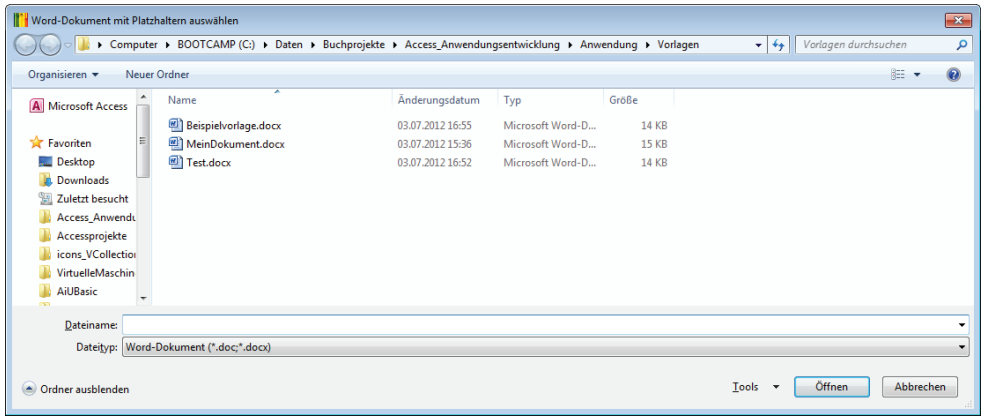


Abbildung 10.6: Auswahl einer bereits erstellten Vorlagendatei

Danach brauchen Sie nur noch auf die Schaltfläche *Dokument erstellen* zu klicken (siehe Abbildung 10.7). Die Anwendung öffnet dann das als Vorlage angegebene Dokument, ersetzt die Platzhalter und speichert das Dokument. Der Zieldateiname wurde bereits beim Öffnen des Formulars vorbelegt. Dabei verwendet die Anwendung die in den beiden Feldern *PfadKundenanschreiben* und *DateinameKundenanschreiben* enthaltenen Angaben und ersetzt auch dort die Platzhalter.

10.2 Programmierung des Word-Exports

Der Aufruf des Dialogs zum Erstellen und Füllen von Word-Vorlagen auf Basis der Daten in den Tabellen der Datenbank erfolgt über eine Schaltfläche, die sich auf der Registerseite *Kommunikation* des Formulars *frmKundeDetail* befindet (siehe Abbildung 10.8).

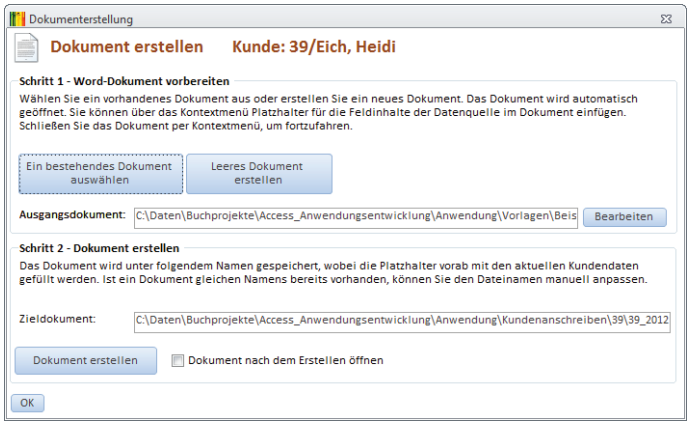


Abbildung 10.7: Alle zum Erstellen des neuen Dokuments benötigten Daten liegen nun vor.

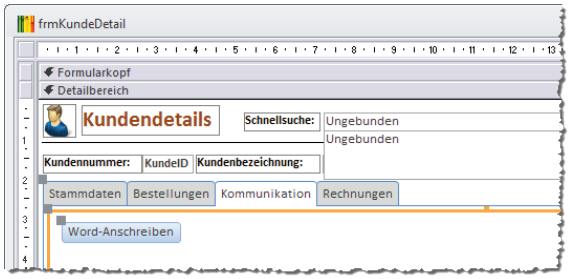


Abbildung 10.8: Schaltfläche zum Öffnen des Word-Exports

Diese Schaltfläche löst die folgende Prozedur aus, die das Formular *frmWordAnschreiben* als modalen Dialog öffnet und mit dem *OpenArgs*-Parameter den Wert des Feldes *KundeID* des aktuell im Formular *frmKundeDetail* angezeigten Kunden als Öffnungsargument übergibt:

```
Private Sub cmdWordAnschreiben_Click()
    DoCmd.OpenForm "frmWordAnschreiben", OpenArgs:=Me!KundeID, WindowMode:=acDialog
End Sub
```

Das Formular *frmWordAnschreiben* sieht wie in Abbildung 10.9 aus. Gleich beim Laden wird das Bezeichnungsfeld im Kopfbereich mit den Daten des Benutzers gefüllt, für den das Anschreiben erstellt werden soll. Dazu hinterlegen Sie für die Ereigniseigenschaft *Beim Laden* des Formulars die folgende Prozedur:

```
Private Sub Form_Load()
    Dim strKunde As String
    Dim strBeschriftung As String
    lngKundeID = Nz(Me.OpenArgs)
```

```
    strKunde = DLookup("Kundenbezeichnung", "tblKunden", "KundeID = " & lngKundeID)
    strBeschriftung = "Kunde: " & lngKundeID & "/" & strKunde
    Me!lblKunde.Caption = strBeschriftung
End Sub
```

Die Prozedur liest den Wert des Öffnungsarguments ein und ermittelt mit einer *DLookup*-Abfrage den Wert des Feldes *Kundenbezeichnung* für diesen Kunden. Beides wird in der *String*-Variablen *strBeschriftung* zusammengefasst und schließlich der Eigenschaft *Caption* des Bezeichnungsfeldes *lblKunde* zugewiesen.

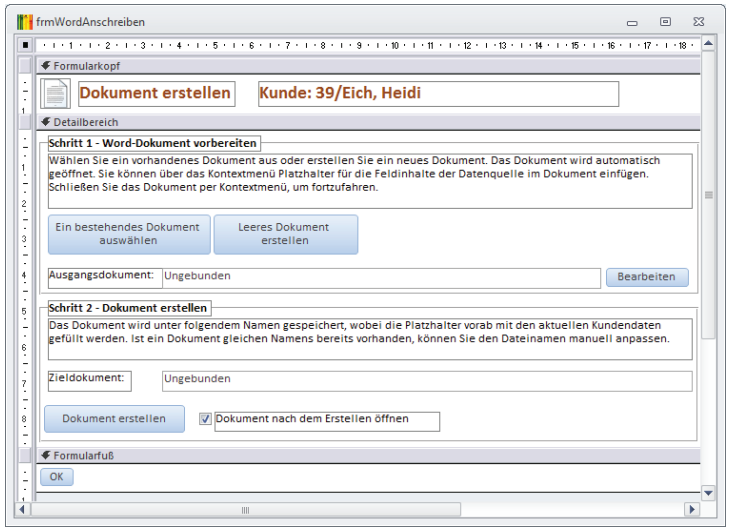


Abbildung 10.9: Entwurf des Formulars *frmWordAnschreiben*

Was aber geschieht, wenn kein Öffnungsargument übergeben wird, weil das Formular beispielsweise direkt über den Navigationsbereich geöffnet wird (was natürlich nicht passieren wird, weil wir dieses ausblenden)? Diesen Fall deckt das Ereignis *Beim Öffnen* ab. Es prüft, ob *OpenArgs* den Wert *Null* hat, und bricht den Öffnungsvorgang gegebenenfalls ab:

```
Private Sub Form_Open(Cancel As Integer)
    If IsNull(Me.OpenArgs) Then
        Cancel = True
    End If
End Sub
```

10.2.1 Quelldokument auswählen

Die Schaltfläche mit der Beschriftung *Ein bestehendes Dokument auswählen* löst die folgende Ereignisprozedur aus:

```
Private Sub cmdDokumentAuswaehlen_Click()
    Dim strVorlagenpfad As String
    strVorlagenpfad = DLookup("PfadVorlagenAnschreiben", "tblOptionen")
    strVorlagenpfad = Replace(strVorlagenpfad, "[Backend]", Backendpfad)
    Me!txtQuelldokument = OpenFilename(strVorlagenpfad, _
        "Word-Dokument mit Platzhaltern auswählen", "Word-Dokument (*.doc, *.docx)")
    If Len(Nz(Me!txtQuelldokument, "")) = 0 Then
        Exit Sub
    End If
    Me!txtZieldokument = DokumentErmitteln(lngKundeID)
End Sub
```

Zunächst ermittelt die Prozedur den Pfad, in dem die Vorlagen für die Anwendung gespeichert sind. Die Basisinformation dazu befindet sich im Feld *PfadVorlagenAnschreiben* der Tabelle *tblOptionen*. Diese Daten stellen Sie über ein einfaches Formular ein, das an die Tabelle *tblOptionen* gebunden ist (siehe Abbildung 10.10).

Abbildung 10.10: Eingeben der Optionen für den Einsatz von Word-Anschreiben

Vorlagenpfad zusammenstellen

Als Vorlagen-Pfad ist beispielsweise der Ausdruck *[Backend]\Vorlagen* angegeben. Das heißt, dass das Vorlagen-Verzeichnis sich im gleichen Verzeichnis befinden soll wie die Backend-Datenbank bei einer aufgeteilten Datenbank, also meist auf einem Server-Rechner. Wenn die Datenbank noch nicht aufgeteilt ist, spielt das auch keine Rolle – dann verwendet die Prozedur einfach das Verzeichnis der aktuellen Datenbank. Bei der Variante, die zum Zeitpunkt der Erstellung dieses Kapitels verwendet wurde, war die Datenbank noch nicht aufgeteilt. Die Prozedur *cmdDokumentAuswaehlen* liest dann mit der *DLookup*-Funktion den Wert des Feldes *PfadVorlagenAnschreiben* aus der Tabelle *tblOptionen* ein. Die folgende Zeile ersetzt, soweit dieser vorhanden ist, den Ausdruck *[Backend]* durch den tatsächlichen Pfad des Backends oder, wie im diesen Fall, den Anwendungspfad. Diesen ermittelt wiederum die Funktion *Backendpfad*, die das Feld *Database* der Systemtabelle für die Tabelle *tblKunden* ausliest. Dort befindet sich der Dateiname des Backends, wenn die Tabelle aus einer anderen Datenbank stammt. Ist dieses Feld leer, gibt es kein Backend und die Funktion liefert den Pfad der aktuellen Datenbank:

```
Public Function Backendpfad() As String
    Dim strPfad As String
```

```
    strPfad = Nz(DLookup("Database", "MSysObjects", "Name = 'tblKunden'"), "")
    If Len(strPfad) > 0 Then
        strPfad = Left(strPfad, InStrRev(strPfad, "\") - 1)
    Else
        strPfad = CurrentProject.Path
    End If
    Backendpfad = strPfad
End Function
```

Mit der Einstellung *[Backend]\Vorlagen* wird übrigens ein Verzeichnis wie in Abbildung 10.11 verwendet.

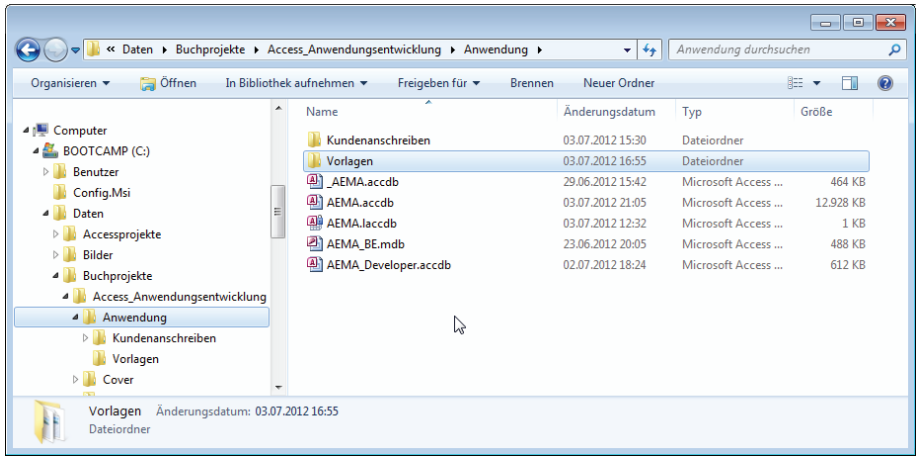


Abbildung 10.11: Verzeichnisstruktur der Anwendung

Anschließend soll mit der im Modul *mdlDateidialoge* enthaltenen Funktion *OpenFilename* das als Vorlage zu verwendende Word-Dokument ausgewählt werden. Diese Funktion erwartet als ersten Parameter das Verzeichnis, das gleich beim Öffnen des Dialogs angezeigt werden soll, eine Überschrift für die Titelleiste und einen Filter für die anzuzeigenden Dateitypen.

Wählt der Benutzer mit diesem Dialog keine Datei aus, wird die Prozedur beendet, sonst füllt diese das Textfeld *txtQuelldokument* mit dem entsprechenden Dateinamen.

Zielfunktion ermitteln

Davon wird auch gleich die Vorgabe für das Zieldokument abgeleitet, was mithilfe einer weiteren Funktion namens *DokumentErmitteln* geschieht. Diese Funktion erwartet als Parameter den Wert des Feldes *KundeID*, der hier in der Variablen *lngKunde* übergeben wird.

Warum das? Nun: Wenn Sie häufiger individuelle Anschreiben für Kunden erstellen, werden Sie es zu schätzen wissen, wenn Sie Elemente der Kundendaten wie beispielsweise *KundeID*, *Vorname*

11 Suchen in Formularen

Eine Datenbankanwendung wäre nur die Hälfte wert, wenn Sie damit nicht auch einfach nach den gewünschten Datensätzen suchen könnten. Deshalb fügen wir den Formularen dieser Anwendung entsprechende Suchfunktionen hinzu.

11.1 Schnellsuche für das Kundendetail-Formular

Das Kundendetail-Formular zeigt immer nur einen Kunden an – entweder einen soeben neu angelegten oder einen über das Übersichtsformular zum Bearbeiten geöffneten Kunden. Im letzteren Fall möchten Sie vielleicht schnell einmal zu einem anderen Kundendatensatz wechseln. Eigentlich müssten Sie dazu das Detailformular schließen, den anderen Kunden im Übersichtsformular auswählen und das Detailformular zu diesem Kunden öffnen.

Wenn Sie gerade einige Aktionen in Zusammenhang mit Kunden oder Bestellungen durchführen, möchten Sie aber vielleicht etwas schneller von Kunde A zu Kunde B gelangen. Für diesen Fall fügen wir dem Kundendetail-Formular ein eigenes Suchfeld hinzu, mit dem Sie eine Zeichenkette eingeben können, nach der verschiedene Felder der Kundenanschrift durchsucht werden.

Das Ergebnis wird in Form eines Listenfeldes gleich unter dem Suchfeld eingeblendet. Durch einen Klick auf einen der Einträge können Sie den gewünschten Kunden anzeigen.

11.1.1 Steuerelemente hinzufügen

Für die Schnellsuche mit Ergebnisliste benötigen Sie zwei Steuerelemente: ein Textfeld, das Sie *txtSuche* nennen, und ein Listenfeld, dem Sie den Namen *lstSuchergebnisse* zuweisen.

Das Textfeld platzieren Sie oben rechts im Detailbereich, das Listenfeld genau darunter – und zwar in der Breite des Textfelds. Nehmen Sie sich ruhig ein wenig Platz dafür – je nachdem, wie viele Felder Sie in die Suche einbeziehen, benötigt das Listenfeld eine entsprechende Breite.

Das Bezeichnungsfeld des Listenfeldes können Sie entfernen, dem Bezeichnungsfeld des Textfeldes weisen Sie die Beschriftung *Suche:* zu. Das Ergebnis sieht etwa wie in Abbildung 11.1 aus.

Sie werden nun mit Recht anmerken, dass die Ergebnisliste ja einige Steuerelemente des Formulars verdeckt. Richtig: Aber es soll ja auch nur erscheinen, wenn der Benutzer einen Suchbegriff in das Textfeld *txtSuche* eingegeben hat, und sonst ausgeblendet werden.

Damit dies gleich beim Anzeigen des Formulars der Fall ist, stellen Sie die Eigenschaft *Sichtbar* des Listenfeld-Steuerelements auf *Nein* ein. Ein Wechsel in die Formularansicht zeigt, dass das Listenfeld tatsächlich verborgen wird (siehe Abbildung 11.2).

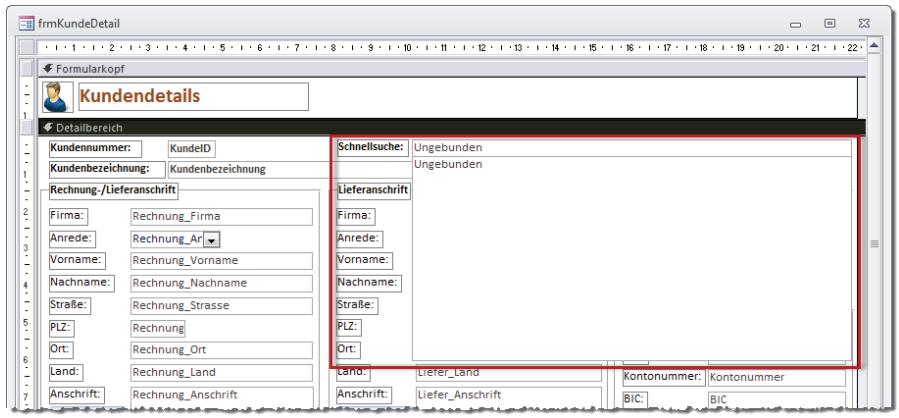


Abbildung 11.1: Die beiden Steuerelemente zur Umsetzung der Schnellsuche in den Kundendetails

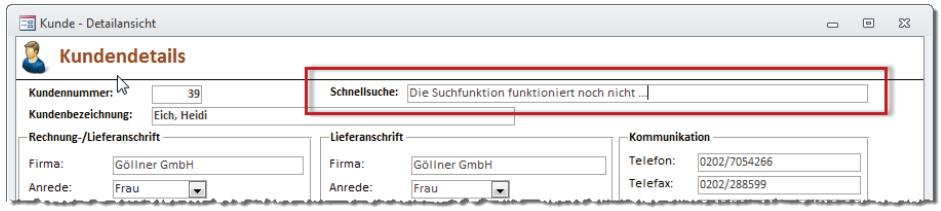


Abbildung 11.2: Die Suchfunktion mit ausgeblendeter Ergebnisliste

11.1.2 Bei Eingabe suchen

Bei jedem Zeichen, das der Benutzer in das Textfeld *txtSuche* eingibt, soll die Anzeige der gefundenen Datensätze aktualisiert werden. Dabei wird diese typischerweise mit jedem eingegebenen Zeichen reduziert. Wie bereits weiter oben einmal verwenden wir dazu wieder die Ereignisprozedur *Bei Änderung* des Textfeldes.

In diesem Fall soll die Prozedur einen SQL-Ausdruck zusammenstellen, der zweierlei erledigt: Erstens soll dieser eine Menge anzuzeigender Felder zu einer einzigen Zeichenkette zusammensetzen, damit diese möglichst platzsparend im Listenfeld *lstSuchergebnis* angezeigt werden können.

Zweitens soll das Kriterium so formuliert werden, dass alle gewünschten Felder durchsucht werden. Im folgenden Ansatz sind dies alle Felder der Anschriften für die Rechnung und die Lieferung. Die zusammengesetzte Abfrage sieht etwa so aus, wenn der Benutzer den Buchstaben *A* als Suchbegriff eingegeben hat:

```
SELECT KundeID, Rechnung_Firma & '|' & Rechnung_Vorname & ' ' & Rechnung_Nachname & '|'
& Rechnung_Strasse & '|' & Rechnung_PLZ & ' ' & Rechnung_Ort & '|' & Rechnung_Land FROM
```

```
tblKunden WHERE Rechnung_Firma & '|' & Rechnung_Vorname & ' ' & Rechnung_Nachname & '|'
& Rechnung_Strasse & '|' & Rechnung_PLZ & ' ' & Rechnung_Ort & '|' & Rechnung_Land & '|'
& Liefer_Firma & '|' & Liefer_Vorname & ' ' & Liefer_Nachname & '|' & Liefer_Strasse &
'|' & Liefer_PLZ & ' ' & Liefer_Ort & '|' & Liefer_Land  LIKE '*A*'
```

Die Abfrage liefert also zwei Felder zurück. Das erste enthält nur das Feld *KundeID* des gefundenen Kundendatensatzes. Das zweite setzt einen Ausdruck zusammen, der aus den durch das Pipe-Zeichen (|) getrennten Feldwerten besteht, also etwa *Göllner GmbH|Heidi Eich|Moosstraße 96|42289 Wuppertal|Deutschland*. Das Feld *KundeID* soll als gebundenes, unsichtbares erstes Feld des Listenfeldes genutzt werden, der zweite Ausdruck soll im Listenfeld erscheinen.

Die *WHERE*-Klausel setzt einen Ausdruck aus den gleichen Feldern und zusätzlich den Feldern der Lieferanschrift zusammen, also eine Reihe von Feldinhalten, die ebenfalls durch das Pipe-Zeichen voneinander getrennt werden. Dieser Ausdruck wird mit dem eingegebenen Suchbegriff verglichen, sodass die Abfrage alle Datensätze zurückgibt, bei denen irgendeines der Felder den Suchbegriff enthält.

Die komplette Prozedur sieht nun wie folgt aus:

```
Private Sub txtSuche_Change()
    Dim strSQL As String
    If Len(Me!txtSuche.Text) > 0 Then
        strSQL = "SELECT KundeID, Rechnung_Firma & '|' & Rechnung_Vorname & ' ' ¬
               & Rechnung_Nachname & '|' & Rechnung_Strasse & '|' & Rechnung_PLZ & ' ' ¬
              & Rechnung_Ort & '|' & Rechnung_Land FROM tblKunden WHERE Rechnung_Firma ¬
           & '|' & Rechnung_Vorname & ' ' & Rechnung_Nachname & '|' & Rechnung_Strasse ¬
                 & '|' & Rechnung_PLZ & ' ' & Rechnung_Ort & '|' & Rechnung_Land & '|' ¬
                   & Liefer_Firma & '|' & Liefer_Vorname & ' ' & Liefer_Nachname & '|' ¬
                                & Liefer_Strasse & '|' & Liefer_PLZ & ' ' & Liefer_Ort ¬
                                 & '|' & Liefer_Land  LIKE '*" & Me!txtSuche.Text & "*'"
        Me!lstSuchergebnis.RowSource = strSQL
        If Me!lstSuchergebnis.ListCount > 0 Then
            Me!lstSuchergebnis.Visible = True
        Else
            Me!lstSuchergebnis.Visible = False
        End If
End Sub
```

Als Erstes prüft die Prozedur, ob der Benutzer überhaupt einen Wert in das Suchfeld *txtSuche* eingegeben hat. Falls ja, weist die Prozedur den zu Beginn erstellten und in der Variablen *strSQL* gespeicherten SQL-Ausdruck der Eigenschaft *RowSource* des Listenfeldes *lstSuchergebnis* zu (also der Eigenschaft *Datensatzherkunft*). Wenn das Listenfeld danach mindestens einen Datensatz enthält, wird dieses eingeblendet. Dafür sorgt die Einstellung der Eigenschaft *Visible* (im Eigenschaftsfenster *Sichtbar*) auf den Wert *True*. Bleibt das Listenfeld leer, wird es wieder

ausgeblendet. Die Variable *lngListenhoehe* speichert einen Wert für die Höhe des Listenfeldes, das aus der Anzahl der Einträge und einer experimentell ermittelten Zeilenhöhe berechnet wird (hier *230*). Die maximale Höhe soll *2950* betragen.

Auch wenn das Suchfeld *txtSuche* beim Auslösen der Ereignisprozedur *txtSuche_Change* leer ist (was etwa nach dem Löschen des einzigen Zeichens der Fall ist), wird das Listenfeld ausgeblendet und gleichzeitig die Datensatzherkunft des Listenfeldes geleert.

Wie wirkt sich die Eingabe nun auf das Listenfeld aus? Wie Abbildung 11.3 zeigt, nicht unbedingt erwartungsgemäß: Das Listenfeld zeigt nur die Werte des Feldes *KundeID* der Suchergebnisse an.

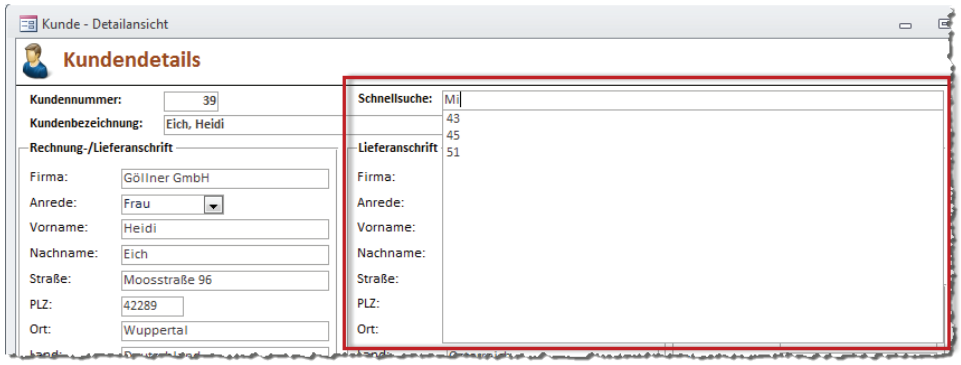


Abbildung 11.3: Das Listenfeld zeigt nur die Kunden-IDs an.

Das ist jedoch kein Problem, denn wir haben nur vergessen, die Spaltenanzahl und Spaltenbreiten des Listenfeldes entsprechend einzustellen. Wenn Sie wie in Abbildung 11.4 die Eigenschaft *Spaltenanzahl* auf den Wert *2* und *Spaltenbreiten* auf den Wert *0cm* einstellen, sollte das Listenfeld das Feld *KundeID* als erste Spalte mit der Breite *0cm* anzeigen. Das zweite Feld mit den für den Benutzer eigentlich interessanten Daten hingegen nimmt den verbleibenden Platz im Listenfeld ein.

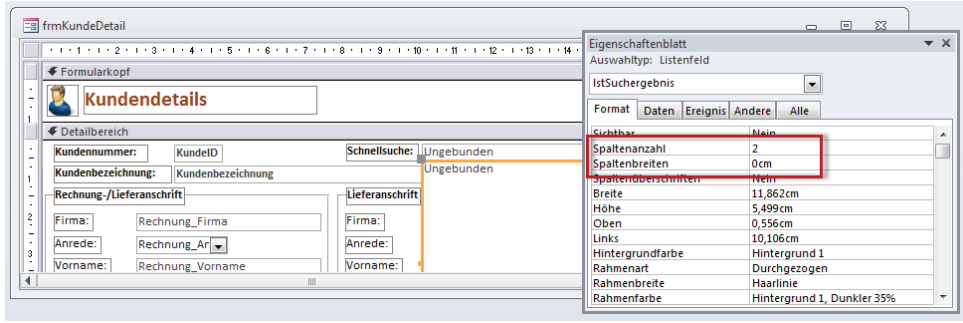


Abbildung 11.4: Einstellen der Spaltenanzahl und Spaltenbreiten des Listenfeldes

Ein Wechsel zur Formularansicht und ein erneuter Test der Suchfunktion liefert dann die gewünschten Ergebnisse (siehe Abbildung 11.5).

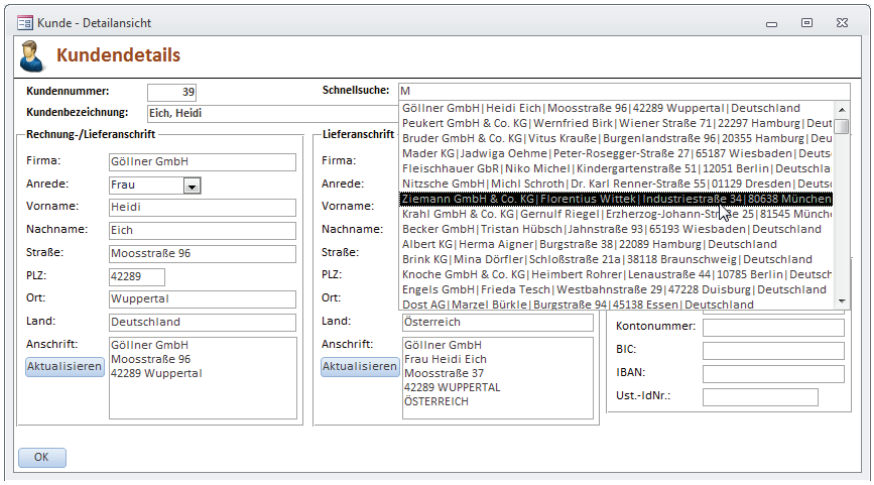


Abbildung 11.5: Das Listenfeld mit korrektem Suchergebnis

11.1.3 Suchtreffer auswählen

Nun müssen wir noch eine Funktion hinzufügen, die den gewünschten Kundendatensatz nach der Auswahl des entsprechenden Eintrags im Listenfeld anzeigt. Der Benutzer hat zwei Möglichkeiten, einen der Listeneinträge auszuwählen: per Mausklick oder durch Auswählen des Eintrags und anschließendes Betätigen der Eingabetaste. Wir schauen uns zunächst die Methode mit dem Mausklick an.

Suchtreffer per Doppelklick

Genau wie die meisten Steuerelemente bietet auch das Listenfeld eine Ereigniseigenschaft namens *Beim Doppelklicken*. Für diese hinterlegen Sie die folgende Ereignisprozedur:

```
Private Sub lstSuchergebnis_DblClick(Cancel As Integer)
    Call DatensatzAnzeigen
End Sub
```

Diese Prozedur ruft eine weitere Prozedur namens *DatensatzAnzeigen* auf, die wie folgt aussieht:

```
Private Sub DatensatzAnzeigen()
    Me.Filter = ""
    Me.Recordset.FindFirst "KundeID = " & Me!lstSuchergebnis
    Me!lstSuchergebnis = Null
```

```
    Me!txtSuche = Null
    Me!txtSuche.SetFocus
    Me!lstSuchergebnis.Visible = False
End Sub
```

Die Prozedur führt die folgenden Schritte durch:

» Deaktivieren des Filters, der gegebenenfalls durch das Öffnen mit *WhereCondition* noch besteht und verhindern würde, dass andere als der aktuell angezeigte Datensatz angezeigt werden können.

» Setzen des Datensatzzeigers auf den Datensatz, dessen Feld *KundeID* mit dem Wert des angeklickten Eintrags des Listenfeldes übereinstimmt

» Leeren der Ergebnisliste *lstSuchergebnis*

» Leeren des Suchfeldes *txtSuche*

» Setzen des Fokus auf das Suchfeld *txtSuche*

» Ausblenden der Ergebnisliste *lstSuchergebnis*

Dies funktioniert auf Anhieb – die Auswahl von Kunden per Eingabe des Suchbegriffes und anschließendem Anklicken des gewünschten Eintrags liefert den entsprechenden Kunden.

Suchtreffer per Auswahl und Eingabetaste

Die Auswahl eines Suchtreffers durch Auf- und Abnavigieren mit den Cursor-Tasten und anschließendes Betätigen der Eingabetaste ist um einiges komplizierter als die Methode mit dem Doppelklick. Als Erstes müssen Sie – der Ergonomie zuliebe – dafür sorgen, dass der Benutzer ganz einfach durch Betätigen der *Nach unten*-Taste vom Textfeld *txtSuche* zum Listenfeld *lstSuchergebnis* springen kann.

Dies erledigen wir, wie schon fast zu erwarten, mit der Ereignisprozedur *Bei Taste ab* des Textfeldes *txtSuche*. Die Prozedur sieht wie folgt aus:

```
Private Sub txtSuche_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim bolZurListe As Boolean
    If Me!lstSuchergebnis.Visible = False Then
        Exit Sub
    End If
    Select Case KeyCode
        Case vbKeyTab, vbKeyDown
            bolZurListe = True
        Case vbKeyRight
            If Me!txtSuche.SelStart = Len(Me!txtSuche.Text) Then
                bolZurListe = True
```

```
            End If
        Case Else
            Debug.Print KeyCode
    End Select
    If bolZurListe Then
        Me!lstSuchergebnis.SetFocus
        Me!lstSuchergebnis = Me!lstSuchergebnis.ItemData(0)
        KeyCode = 0
    End If
End Sub
```

Im ersten Schritt prüft die Prozedur, ob das Listenfeld *lstSuchergebnis* überhaupt angezeigt wird. Falls nicht, gibt es anscheinend keine Suchergebnisse und wir brauchen keine spezielle Aktion durchzuführen – die Prozedur wird hier schlicht verlassen.

Anderenfalls prüft die Prozedur zunächst wieder den Wert des Parameters *KeyCode*, der ja einen Wert für die gedrückte Taste liefert. Entspricht dieser dem Betätigen der *Tabulator*-Taste oder der *Nach unten*-Taste, wird die Variable *bolZurListe* auf *True* eingestellt. Diese Variable gibt an, ob der Fokus anschließend auf das Steuerelement *txtSuchergebnis* verschoben werden soll. Auch beim Betätigen der *Nach rechts*-Taste soll ein Wechsel zum Listenfeld erfolgen, allerdings nur, wenn die Einfügemarke sich bereits ganz rechts im Textfeld *txtSuche* befindet. Dies prüft die Prozedur, indem sie die Eigenschaft *SelStart* des Textfeldes, das die Startposition der aktuellen Markierung liefert, mit der Länge der angezeigten Zeichenkette vergleicht (ermittelt mit der *Len*-Funktion). Sind diese gleich, befindet sich die Einfügemarke ganz rechts und *bolZurListe* wird ebenfalls auf *True* eingestellt.

Hat *bolZurListe* am Ende der Prozedur den Wert *True*, wird der Fokus auf das Steuerelement *lstSuchergebnis* verschoben und der erste Datensatz des Listenfeldes markiert. Dies erledigt die Prozedur, indem Sie den Wert der Eigenschaft *ItemData* für den ersten Eintrag des Listenfeldes ermittelt, also den Eintrag mit dem Index *0*. *ItemData* liefert den Wert der gebundenen Spalte. Nach dem Einstellen des Listenfeldes *lstSuchergebnis* selbst auf diesen Wert wird der entsprechende Eintrag markiert.

Nun kommen wir zum interessanten Teil: Dem Navigieren im Listenfeld und der Auswahl des aktuellen Eintrags mit der *Eingabe*-Taste. Das Navigieren selbst, also das Bewegen der Markierung mit der *Nach oben*- und der *Nach unten*-Taste ist noch nicht einmal kompliziert (zumindest nicht, wenn nicht gerade der erste oder der letzte Eintrag markiert ist). Interessant sind vor allem die Betätigungen der Tasten *Escape*, *Eingabe* oder *Tabulator*. Diese Prozedur zerlegen wir aus Gründen der Übersicht wieder in kleine Teile.

Die Prozedur, die erneut durch Niederdrücken einer Taste ausgelöst wird, erhält wieder die beiden Parameter *KeyCode* und *Shift*, die beide eine Rolle spielen werden. Außerdem deklariert sie eine Variable namens *bolZurueck*, die auf *True* eingestellt wird, wenn der Fokus zurück zum Suchfeld *txtSuche* verschoben werden soll:

```
Private Sub lstSuchergebnis_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim bolZurueck As Boolean
```

Anschließend untersucht eine *Select Case*-Bedingung die gedrückte Taste. Der erste *Case*-Fall beschäftigt sich mit der *Escape*-Taste. Wird diese gedrückt, soll der Fokus zurück auf das Steuerelement *txtSuche* verschoben werden, die Einfügemarke auf der letzten Position dieses Textfeldes landen, das Listenfeld *lstSuchergebnis* ausgeblendet und geleert und die gedrückte Taste gelöscht werden:

```
    Select Case KeyCode
        Case vbKeyEscape
            Me!txtSuche.SetFocus
            Me!txtSuche.SelStart = Me!txtSuche.SelLength
            Me!lstSuchergebnis.Visible = False
            Me!lstSuchergebnis = Null
            KeyCode = 0
```

Der zweite *Case*-Fall nimmt sich das Betätigen der *Eingabe*-Taste vor.

Geschieht dies, soll der aktuell markierte Eintrag des Suchergebnisses im Formular *frmKunde-Detail* angezeigt werden:

```
        Case 13
            DatensatzAnzeigen
            KeyCode = 0
            Exit Sub
```

Im dritten Fall untersucht die Prozedur die Tasten *Nach oben* und *Nach links*. In beiden Fällen erhält die Variable *bolZurueck* den Wert *True*:

```
        Case vbKeyUp, vbKeyLeft
            bolZurueck = True
```

Gleiches geschieht, wenn der Benutzer die *Tabulator*-Taste drückt – aber nur, wenn er dabei die *Umschalt*-Taste gedrückt hält:

```
        Case vbKeyTab
            If Shift = acShiftMask Then
                bolZurueck = True
            End If
```

In allen anderen Fällen soll einfach der KeyCode im Direktfenster ausgegeben werden – dies dient aber eher Zwecken zur Untersuchung der Funktionsweise der Prozedur:

```
        Case Else
            Debug.Print KeyCode
    End Select
```

Sollte sich herausstellen, dass der Benutzer eine Taste gedrückt hat, die potenziell nach einem Verschieben des Fokus zurück zum Textfeld *txtSuche* verlangt (*bolZurueck = True*), ist noch eine weitere Untersuchung nötig – nämlich die, ob gerade der erste Eintrag des Listenfeldes markiert ist (*ListIndex = 0*). Ist dies der Fall, wird der Fokus zurück zum Textfeld *txtSuche* verschoben, die Einfügemarke an das Ende gesetzt und die Auswahl im Suchergebnis aufgehoben.

Das Suchergebnis wird jedoch nicht ausgeblendet:

```
    If bolZurueck And Me!lstSuchergebnis.ListIndex = 0 Then
        Me!txtSuche.SetFocus
        Me!txtSuche.SelStart = Me!txtSuche.SelLength
        Me!lstSuchergebnis = Null
        KeyCode = 0
    End If
End Sub
```

Fertig – die Schnellsuche funktioniert und ist außerdem sehr ergonomisch. Fehlt nur noch eine kleine Optimierung ...

11.1.4 Mit Komfort zur Suche

Um die Suche richtig benutzerfreundlich zu gestalten, sorgen wir dafür, dass der Benutzer nur noch die Tastenkombination *Strg + F* drücken muss, um die Einfügemarke in das Suchfeld zu verschieben. Welche Ereigniseigenschaft ist für diesen Fall die richtige? Klar ist: Es muss eines der Ereignisse *Bei Taste auf* oder *Bei Taste ab* sein.

Wie finden Sie heraus, welches das Richtige ist? Ganz einfach: Verwenden Sie die Tastenkombination *Strg + F*, bevor Sie eine eigene Funktion dafür hinterlegen, und prüfen Sie, ob Access beim Herunterdrücken oder beim Loslassen der Tasten den Suchen-Dialog einblendet. Das Ergebnis ist: Wir verwenden die Ereigniseigenschaft *Bei Taste ab*!

Damit können Sie die Ereignisprozedur für das Ereignis *Bei Taste ab* anlegen und wie folgt füllen:

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
        Case vbKeyF
            If Shift = acCtrlMask Then
                Me!txtSuche.SetFocus
                KeyCode = 0
            End If
    End Select
End Sub
```

Die Prozedur erhält zwei Übergabeparameter. Der erste heißt *KeyCode* und liefert einen Zahlencode für die gedrückte Taste. Der zweite heißt *Shift* und liefert ebenfalls einen Zahlenwert,

der angibt, ob eine der drei Tasten *Umschalt* (*1*), *Strg* (*2*) oder *Alt* (4) gedrückt wurde. Wenn der Benutzer Kombinationen dieser Tasten betätigt, werden die entsprechenden Zahlenwerte addiert zurückgegeben. Drücken Sie also beispielsweise alle drei Tasten gleichzeitig, liefert der Parameter *Shift* beim Auslösen der Prozedur den Wert *7*.

Durch Auswertung der Parameter *KeyCode* und *Shift* können Sie also genau auf bestimmte Tastenkombinationen reagieren. Wenn Sie unsicher sind, welchen Zahlenwert die verschiedenen Tasten haben, lassen sie diese einfach zu Beginn der Prozedur ins Direktfenster schreiben – beispielsweise mit einer Anweisung wie der folgenden:

```
Debug.Print "Keycode: " & KeyCode & " Shift: " & Shift
```

Es gibt allerdings noch eine kleine Hürde: Die Prozedur *Form_KeyDown* wird nur ausgelöst, wenn das Formular aktuell den Fokus hat – nicht aber, wenn beispielsweise eines der Steuerelemente den Fokus besitzt. In diesem Fall müsste man eigentlich das gleichnamige Ereignis des Steuerelements definieren. Zum Glück besitzen Formulare jedoch eine Eigenschaft namens *Tastenvorschau*. Wenn Sie diese auf *Ja* einstellen, wird das Ereignis *Bei Taste ab* des Formulars vor dem gleichnamigen Ereignis der Steuerelemente ausgelöst.

Die obige Prozedur prüft nun gleich zu Beginn in einem *Select Case*-Statement, ob der Benutzer die Taste *F* gedrückt hat. Statt des Zahlenwertes *70*, der normalerweise dem *KeyCode* der Taste *F* entspricht, verwenden wir hier die wesentlich aussagekräftigere Visual Basic-Konstante *vbKeyF*.

Hat der Benutzer die Taste *F* betätigt, prüft die Prozedur den Wert des Parameters *Shift*. Hat dieser den Wert *acCtrlMask*, was dem Zahlenwert *2* oder der *Strg*-Taste entspricht, ist die Bedingung der *If...Then*-Bedingung erfüllt und das Steuerelement *txtSuche* wird mit der *SetFocus*-Methode aktiviert. Außerdem stellt die Prozedur den Parameter *KeyCode* auf den Wert *0* ein, was dazu führt, dass die Verarbeitung der Eingabe des Buchstabens *F* hier endet.

11.1.5 Listenfeld stört in der Entwurfsansicht

Ein kleines Problem gibt es noch: Wenn Sie das Formular weiterentwickeln möchten, vor allem in dem Bereich, der nun vom Listenfeld *lstSuchergebnis* verdeckt wird, müssen Sie das Feld immer verschieben, um die anderen Steuerelemente zu sehen. Das Problem lösen wir auf einfache Weise: Ändern Sie einfach die Höhe des Listenfeldes auf den Wert *0* (oder zumindest auf einen sehr kleinen Wert, wenn Sie das Listenfeld zumindest noch sehen oder anklicken wollen).

Damit die Höhe des Listenfeldes beim Anzeigen wieder vergrößert wird, fügen Sie der Prozedur *txtSuche_Change* unter der Zeile

```
If Me!lstSuchergebnis.ListCount > 0 Then
```

die folgende Zeile hinzu:

```
Me!lstSuchergebnis.Height = 3000
```

Oder Sie verfeinern den Algorithmus noch so weit, dass sich die Höhe des Listenfeldes der Anzahl der angezeigten Einträge anpasst:

```
Dim lngListenhoehe As Long
'...
lngListenhoehe = Me!lstSuchergebnis.ListCount * 230
If lngListenhoehe > 2950 Then
    Me!lstSuchergebnis.Height = 2950
Else
    Me!lstSuchergebnis.Height = lngListenhoehe
End If
```

Den Wert für die Höhe einer Zeile ermitteln Sie experimentell. Wenn die Höhe eine maximale Höhe übersteigt, begrenzen Sie diese durch Setzen eines fixen Wertes, hier *2.950*.

11.2 Detailsuche für die Kunden-Übersicht

Die Kunden-Übersicht enthält erstens eine Reihe Spalten und zweitens, je nach Anwendungsfall, eine große Menge Datensätze. Da ist es sinnvoll, eine Suchfunktion bereitzustellen, welche das genaue Filtern nach allen Feldinhalten erlaubt. Je nach Suchergebnis möchten Sie vielleicht die Suchanfrage im zweiten Anlauf noch verfeinern, ohne die vorherige Suchkonstellation zu verwerfen – und gleichzeitig auch noch in der Liste der Suchergebnisse blättern, ohne den Blick auf die Suchkriterien zu verlieren.

Kein Problem: Mit dem Suchformular für die Kunden-Übersicht haben Sie all dies im Blick.

11.2.1 Suchfunktion aufrufen

Der Aufruf der Suchfunktion erfolgt im offensichtlichsten Fall über eine Schaltfläche namens *cmdSuche*, die Sie im Formularkopf des Formulars *frmKundenuebersicht* anbringen (siehe Abbildung 11.6). Sie können diese Schaltfläche natürlich auch im Fußbereich des Formulars unterbringen. Diese Funktion soll das Formular *frmKundensuche* aufrufen. Dazu hinterlegen Sie die folgende Ereignisprozedur für das Ereignis *Beim Klicken* der Schaltfläche *cmdSuche*:

```
Private Sub cmdSuche_Click()
    DoCmd.OpenForm "frmSuche"
End Sub
```

Tabellenänderungen

Bevor wir das Suchformular erstellen, sind kleine Erweiterungen am Datenmodell der Tabelle *tblKunden* nötig. Damit dieses Suchformular auch die Suche nach dem Inhalt von Ja/*Nein*-Feldern unterstützt, soll die Tabelle *tblKunden* um zwei *Ja/Nein*-Felder erweitert werden. Die

Felder heißen *Umsatzsteuerpflichtig* und *Newsletter*. Das erste gibt an, ob der Kunde umsatzsteuerpflichtig ist oder nicht – bei manchen Kunden aus dem Ausland fällt die Umsatzsteuer weg.

Die zweite legt fest, ob der Kunde den Newsletter erhalten soll. Beide Felder erhalten den Standardwert *Ja* (siehe Abbildung 11.7).

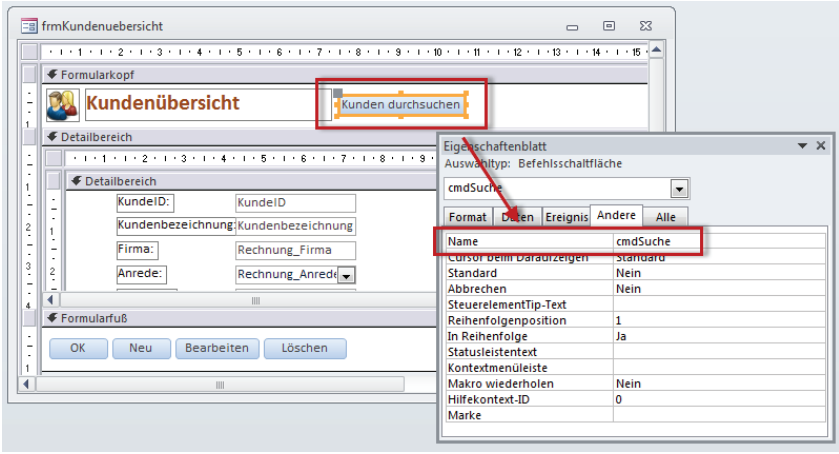


Abbildung 11.6: Schaltfläche zum Aufrufen der Suchfunktion

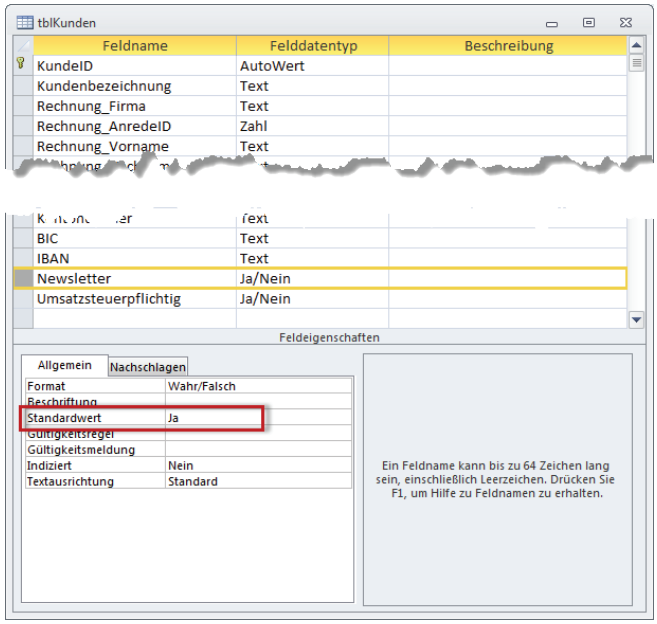


Abbildung 11.7: Erweiterung der Tabelle *tblKunden*

12 Formulare optimieren

In den vorherigen Kapiteln, die sich mit der Erstellung der Benutzeroberfläche und hier speziell mit der Programmierung der Formulare beschäftigt haben, wurden lediglich die Grundgerüste der Formulare erstellt. Es gibt noch eine Menge zu optimieren, was wir in diesem Kapitel erledigen. Die vorgestellten Techniken haben wir bei allen vorhandenen Formularen der Beispielanwendung durchgeführt.

12.1 Formulare mit der Eingabe-Taste schließen

Wenn ein Formular eine *OK*-Schaltfläche besitzt, sollten Sie für diese die Eigenschaft *Standard* auf *Ja* einstellen. Damit stellen Sie sicher, dass der Benutzer das Formular durch Betätigen der Eingabe-Taste schließen kann. Auch wenn es nicht explizit bei allen Formular-Beschreibungen erwähnt wurde, sollte doch jedes Formular dieses Feature aufweisen.

12.2 Formulare mit der Escape-Taste schließen

Gleiches gilt, wenn das Formular eine *Abbrechen*-Taste enthält. Diese sollte der Benutzer über die *Escape*-Taste ansteuern können. Alles, was Sie dazu tun müssen, ist das Einstellen der Eigenschaft *Abbrechen* der Schaltfläche auf den Wert *Ja*. Auch hier gilt: Wir haben jedes Formular, dass eine *Abbrechen*-Schaltfläche besitzt, so eingestellt, dass der Benutzer es schnell mit der *Escape*-Taste schließen kann – auch wenn es nicht überall beschrieben wird.

12.3 Validierung von Formularen

Ein wichtiges Thema beim Erstellen ergonomischer Formulare ist die Validierung der Eingaben des Benutzers. Wir schauen uns alle relevanten Fälle in den folgenden Abschnitten am Beispiel einiger Formulare dieser Anwendung an.

12.3.1 Geschäftsregeln

Eigentlich soll eine Validierung Geschäftsregeln durchsetzen. Sie möchten einen Artikel anlegen? Dann müssen Sie zumindest einen Artikelnamen, den Preis, den Mehrwertsteuersatz, eine Warengruppe und eine Einheit festlegen. Solche Geschäftsregeln können Sie an mehreren Stellen definieren, zum Beispiel direkt im Entwurf des Datenmodells. Oder Sie setzen diese Geschäftsregeln in den Formularen fest. In den meisten Fällen ist eine Kombination erforderlich: Wenn Sie nämlich Restriktionen in den Tabellen definieren und ein Benutzer diese durch die

Eingabe von Daten in einem Formular verletzt, zeigt Access seine Standardmeldung für den jeweiligen Fall an.

Dies werden Sie vermutlich abfangen wollen, um eine eigene, aussagekräftigere Meldung zu liefern.

12.3.2 Restriktionen auf Tabellenebene

Tabellen sehen die folgenden Möglichkeiten zum Festlegen von Restriktionen vor:

» *Felddatentypen*: Ein Zahlenfeld meckert, wenn Sie Text eingeben, ein Datumsfeld kann nur gültige Datumswerte aufnehmen.

» *Feldgröße*: Begrenzt die Anzahl der möglichen Zeichen

» *Gültigkeitsregel* und *Gültigkeitsmeldung*: Erlauben die Angabe von Regeln wie *Länge([PLZ])>3*, was die Anzeige der angegebenen Meldung nach sich zieht, wenn der Benutzer eine PLZ mit weniger als vier Zeichen einträgt. Die Prüfung erfolgt vor dem Verlassen des Feldes.

» *Eingabe erforderlich* und *Leere Zeichenfolge*: Stellen Sie *Eingabe erforderlich* auf *Ja* und *Leere Zeichenfolge* auf *Nein* ein, damit der Datensatz nicht gespeichert wird, wenn das Feld leer ist.

» Eindeutiger Index für einzelne Felder: Damit legen Sie fest, dass ein Feld nur eindeutige Werte enthalten darf.

» Eindeutiger Index für mehrere Felder: Legt fest, dass nur eindeutige Kombinationen von Daten in zwei oder mehr Feldern vorkommen. Wird vor allem für die Verknüpfungstabellen von m:n-Beziehungen genutzt.

» *Gültigkeitsregel* und *Gültigkeitsmeldung* auf Tabellenebene: Damit können Ausdrücke auch mit Bezug auf verschiedene Felder festgelegt werden (zum Beispiel *[Startdatum]<[Enddatum]*). Die Prüfung erfolgt vor dem Speichern des Datensatzes.

» *Referenzielle Integrität*: Auch die Festlegung dieser Eigenschaft für eine Beziehung ist eine Restriktion, sogar eine sehr mächtige. Sie sorgt dafür, dass ein Feld nur mit den Daten des Primärschlüsselfeldes der verknüpften Tabelle gefüllt werden darf.

Einige dieser Restriktionen legt man unbewusst fest, indem man beispielsweise den Datentyp auswählt, die anderen mit der Absicht, die Eingabe der Daten zu reglementieren.

12.3.3 Tabellen-, Feld- und Beziehungsrestriktionen bei der Dateneingabe

Wenn Sie Daten in ein Formular eingeben, das an eine Tabelle gebunden ist, und dabei die für die Tabelle, die Felder oder die Beziehung festgelegten Restriktionen unterwandern, zeigt Access eine entsprechende Meldung an – übrigens die gleiche, die auch beim Eingeben ungül-

tiger Daten direkt in der Tabelle erscheint. Das wird unsere erste Baustelle: Wir wollen dem Benutzer eine aussagekräftige Meldung liefern, mit der dieser arbeiten kann – manche der eingebauten Meldungen sind für den Entwickler verständlich, aber der Benutzer wird damit nichts anfangen können.

Wie aber fängt man solche Meldungen ab? Dazu liefern Formulare die Ereigniseigenschaft *Bei Fehler*. Dieses Ereignis feuert immer, wenn eine im Datenmodell festgelegte Restriktion verletzt wird. Nehmen wir uns das Formular *frmArtikelDetail* vor. Das Feld *Einzelpreis* ist zwar mit dem Felddatentyp *Währung* versehen, aber intern wird es als Zahlenfeld behandelt. Wenn nun ein Benutzer, der es allzu genau nimmt, beispielsweise noch die Währung als *EUR* mit einträgt, löst dies den Fehler aus Abbildung 12.1 aus.

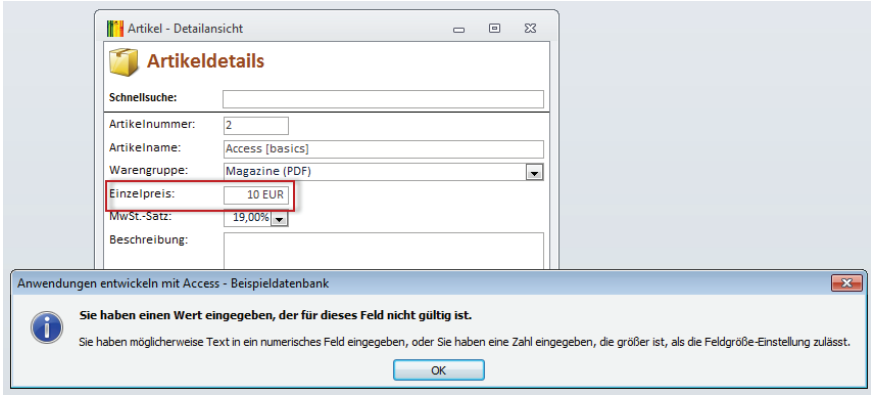


Abbildung 12.1: Meldung beim Eingeben eines ungültigen Wertes in ein Zahlenfeld

Öffnen Sie das Formular im Entwurf, aktivieren Sie das Eigenschaftsfenster und wählen Sie für die Eigenschaft *Bei Fehler* den Eintrag *[Ereignisprozedur]* aus. Klicken Sie dann auf die Schaltfläche mit den drei Punkten (...), um die durch dieses Ereignis ausgelöste Prozedur anzulegen. Füllen Sie die Prozedur wie folgt auf:

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    Select Case DataErr
        Case Else
            MsgBox "Fehler bei der Dateneingabe:" & vbCrLf & DataErr & vbCrLf _
                & AccessError(DataErr)
    End Select
End Sub
```

Der Parameter *DataErr* liefert die Fehlernummer, genau wie es *Err.Number* in einer herkömmlichen Fehlerbehandlung erledigt. Leider gibt es keinen Parameter, der die Fehlerbeschreibung ausgibt. Allerdings gibt es eine VBA-Funktion namens *AccessError*, welche die zu einer Fehlernummer hinterlegte Meldung liefert.

Mit der Prozedur *Form_Error* behandeln Sie alle Werte von *DataErr* in einer *Select Case*-Bedingung. Auf diese Weise können Sie sukzessive alle auftretenden Fehler bei der Eingabe erkennen und, was besonders wichtig ist, die Fehlernummer ermitteln – zum Beispiel den Fehler aus Abbildung 12.2.

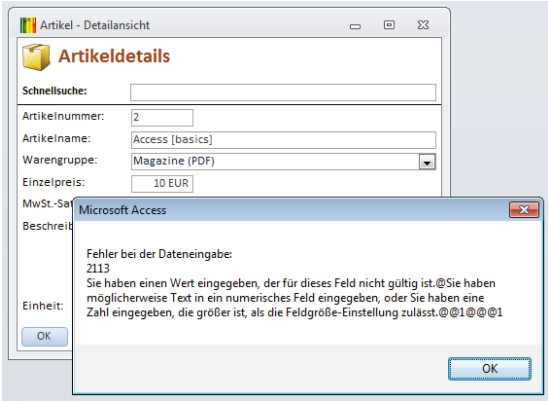


Abbildung 12.2: Fehler bei Eingeben eines ungültigen Wertes

In der nächsten Version der Prozedur *Form_Error* behandeln Sie diesen Fehler bereits gezielt und mit einer eigenen Fehlermeldung. Außerdem sorgen Sie mit dem Wert *acDataErrContinue* für den Rückgabeparameter *Response* dafür, dass die von Access selbst generierte Meldung ausbleibt:

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    Select Case DataErr
        Case 2113
            MsgBox "Sie haben einen ungültigen Wert eingegeben."
        Case Else
            MsgBox "Fehler bei der Dateneingabe:" & vbCrLf & DataErr & vbCrLf _
                & AccessError(DataErr)
    End Select
End Sub
```

Nun weiß der Benutzer allerdings noch nicht, was genau an dem eingegebenen Wert falsch ist. Warum? Weil die Fehlerbehandlung keine Information darüber hat, welches Feld beziehungsweise welches Steuerelement den Fehler ausgelöst hat. Also fügen wir eine weitere *Select Case*-Anweisung ein, die das zuletzt betätigte Steuerelement ermittelt und auf Basis des Namens des Steuerelements noch genauere Informationen zum Fehler liefert. Der *Select Case*-Zweig für den Fehler *2113* sieht nun so aus:

```
Case 2113
    Select Case Screen.ActiveControl.Name
```

```
        Case "txtEinzelpreis"
            MsgBox "Sie haben einen ungültigen Wert für das Feld 'Einzelpreis' eingegeben."
            Response = acDataErrContinue
```

Schauen wir uns das Formular *frmBestellungDetail*, genau genommen das Unterformular *sfmBestellungDetail*, an. Dort gibt es in der Datenherkunft das Feld *ArtikelID*. Für dieses Feld und das Feld *BestellungID* gibt es einen eindeutigen Index.

Wenn der Benutzer also einen Artikel zweimal zu einer Bestellung hinzufügt, löst dies den Fehler *3022* aus. Können wir nun genauso wie im vorherigen Fall validieren? Probieren wir es mit folgendem Code-Schnipsel für das Ereignis *Bei Fehler* des Unterformulars aus:

```
Case 3022
    Select Case Screen.ActiveControl.Name
        Case "cboArtikelID"
            MsgBox "Sie dürfen jeden Artikel nur einmal auswählen." & vbCrLf _
                & "Passen Sie gegebenenfalls die Menge an."
            Response = acDataErrContinue
    End Select
```

Leider funktioniert dies nicht zuverlässig, da dieser Fehler erst beim Speichern des Datensatzes ausgelöst wird. Das bedeutet, dass der Benutzer erst den gleichen Artikel ein zweites Mal auswählen kann, dann aber noch weitere Daten eingeben kann, bevor er den Datensatz etwa durch den Wechsel zu einem anderen Datensatz speichert. Und in diesem Moment hilft *Screen.ActiveControl* natürlich nicht mehr, das den Fehler auslösende Steuerelement zu ermitteln. Dann erscheint die Standardmeldung statt der für das Steuerelement *cboArtikelID* festgelegten Meldung (siehe Abbildung 12.3 – diese Meldung wurde übrigens nachgebaut, da die Originalmeldung zu breit ist, um sie abzudrucken).

Zum Validieren von Abhängigkeiten über mehrere Felder ist also ein alternativer Mechanismus erforderlich, der durch ein anderes Ereignis ausgelöst werden muss. Für diesen Fall bietet sich das Ereignis *Vor Aktualisierung* des Formulars an. Dieses Ereignis wird ausgelöst, bevor der Datensatz gespeichert wird.

Das Ereignis *Nach Aktualisierung* ist deshalb nicht sinnvoll, weil zu diesem Zeitpunkt schon die für die Tabellen und Felder festgelegten Mechanismen wie Gültigkeitsregeln, eindeutige Indizes et cetera ausgelöst wurden. Schauen wir uns also an, was wir mit dem Ereignis *Vor Aktualisierung* des Formulars bewirken können. Legen Sie dazu für das Ereignis die folgende Ereignisprozedur an:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If Not IsNull(DLookup("BestellpositionID", "tblBestellpositionen", "BestellungID = "  _
            & Me!BestellungID & " AND ArtikelID = " & Me!ArtikelID)) Then
        MsgBox "Sie dürfen jeden Artikel nur einmal auswählen." & vbCrLf _
            & "Passen Sie gegebenenfalls die Menge an."
```

```
        Me!cboArtikelID.SetFocus
        Cancel = True
    End If
End Sub
```

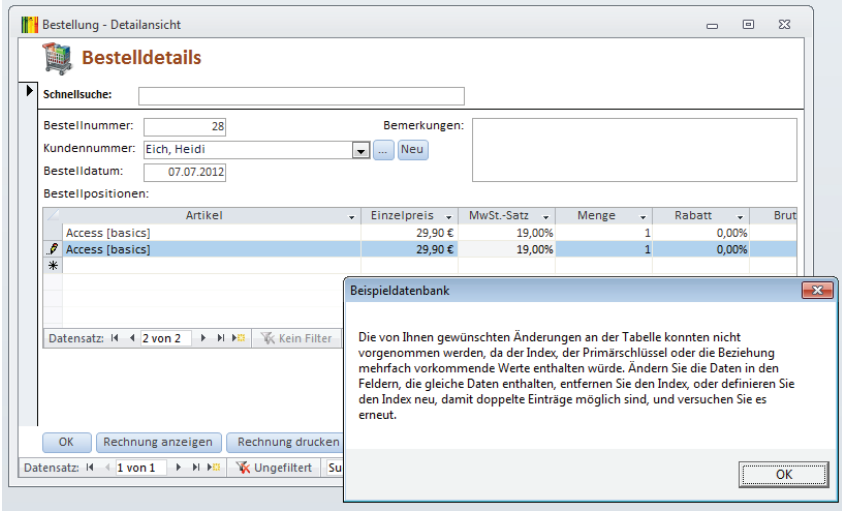


Abbildung 12.3: Die falsche Fehlermeldung beim Validieren

Dies prüft vor dem Speichern des Datensatzes, ob es bereits einen Datensatz gibt, der die gleiche Kombination von Werten für die beiden Felder *BestellungID* und *ArtikelID* aufweist. In diesem Fall zeigt es erstens eine entsprechende Meldung an, stellt den Fokus auf das Steuerelement *cboArtikelID* ein und bricht mit dem Wert *True* für den Parameter *Cancel* den Speichern-Vorgang für den Datensatz ab.

Das Gleiche gelingt natürlich auch für einzelne Steuerelemente. Sie können auch direkt bei der Auswahl des Artikels mit dem Kombinationsfeld *cboArtikelID* prüfen, ob es in der aktuellen Bestellung bereits eine Bestellposition mit diesem Artikel gibt. Dazu legen Sie eine Ereignisprozedur für das Ereignis *Vor Aktualisierung* des jeweiligen Steuerelements an:

```
Private Sub cboArtikelID_BeforeUpdate(Cancel As Integer)
    If Not IsNull(DLookup("BestellpositionID", "tblBestellpositionen", "BestellungID = " _
            & Me!BestellungID & " AND ArtikelID = " & Me!ArtikelID)) Then
        MsgBox "Sie dürfen jeden Artikel nur einmal auswählen." & vbCrLf _
            & "Passen Sie gegebenenfalls die Menge an."
        Me!cboArtikelID.Undo
        Cancel = True
    End If
End Sub
```

Die Prozedur funktioniert grundsätzlich wie die zuvor genannte, jedoch ist es hier nicht möglich (und auch nicht nötig), den Fokus auf das betroffene Feld einzustellen. Stattdessen leeren wir dieses Feld mit der *Undo*-Methode, damit der Benutzer es gleich direkt neu füllen kann.

Diese Art der Validierung gibt es natürlich auch eine Nummer einfacher, und zwar für Felder, deren Wert unabhängig von anderen Feldern bestimmten Einschränkungen unterworfen ist. Im einfachsten Fall darf ein Feld einfach nicht leer sein. Sie haben dann folgende Möglichkeiten:

» Sie prüfen den Feldwert vor der Aktualisierung des Feldes.

» Sie prüfen den Feldwert vor der Aktualisierung des Datensatzes.

Im Falle eines leeren Feldes reicht es, wenn die Prüfung beim Speichern des Datensatzes stattfindet. Der Benutzer hat sich dann anscheinend noch gar nicht um das Feld gekümmert und darf dies nachholen.

Wenn der Benutzer aber falsche Daten in ein Feld eingegeben hat, sollte direkt eine entsprechende Meldung erscheinen – der Benutzer ist dann gedanklich auch noch bei diesem Feld und kann den Inhalt schnell korrigieren.

In Fällen, bei denen der eingegebene Wert die durch das Datenmodell festgelegten Restriktionen verletzt, müssen Sie sogar direkt auf falsche Eingaben in ein Feld reagieren – und dies gelingt ausschließlich über die Ereignisprozedur für die Eigenschaft *Bei Fehler*. Diese wird nämlich bei feldbezogenen Restriktionen vor dem Ereignis *Vor Aktualisierung* ausgelöst.

12.3.4 Alternative zur herkömmlichen Validierung

Alles in allem ist zu erwähnen, dass eine Ausstattung aller Formulare und der enthaltenen Steuerelemente mit den Validierungsprozeduren ein relativ hoher Aufwand ist – ähnlich wie der Aufwand, der durch die Ausstattung mit Fehlerbehandlungen entsteht.

Ich selbst setze in meinen Anwendungen einen Satz von Klassen ein, mit der Sie die Validierung mit wenigen Zeilen definieren können.

In der Beispieldatenbank finden Sie diese Klassen bereits vor. Wenn Sie diese in Ihrer eigenen Anwendung einsetzen möchten, importieren Sie die folgenden Objekte:

» Standardmodule *mdlGlobal* und *mdlValidation*

» Klassenmodule *clsControlValidation*, *clsFormValidation* und *clsValidation*

Anschließend deklarieren Sie im Klassenmodul, das Sie mit der Validierung ausstatten möchten, ein Objekt auf Basis der Klasse *clsValidation*:

```
Dim objValidation As clsValidation
```

Legen Sie dann, soweit noch nicht vorhanden, eine Ereignisprozedur für das Ereignis *Beim Laden* an, die wie folgt aussieht:

```
Private Sub Form_Load()
    Set objValidation = New clsValidation
    With objValidation
        'Validierungen
    End With
End Sub
```

Dort, wo nun noch der Text *'Validierungen* steht, fügen Sie die einzelnen Validierungsanweisungen hinzu. Es gibt zwei verschiedene Validierungen:

» Validierungen, die beim Speichern des Steuerelementinhalts ausgelöst werden (*AddControlValidation*) und

» Validierungen, die beim Speichern des Datensatzes ausgelöst werden (*AddFormValidation*).

Nach den Ausführungen der vorherigen Abschnitte wissen Sie ja bereits, zu welchen Zeitpunkten welche Validierung erfolgen sollte.

Validierung des Artikelnamens I

Schauen wir uns einige Validierungen an, zum Beispiel im Formular *frmArtikelDetail*. Der Benutzer soll zum Beispiel keinen Datensatz anlegen können, ohne einen Wert für das Feld *Artikelname* einzutragen. Dazu fügen Sie der Prozedur die folgende Validierung hinzu (siehe Abbildung 12.4):

```
Private Sub Form_Load()
    Set objValidation = New clsValidation
    With objValidation
        .AddFormValidation Me!txtArtikelname, eIsNull, "Artikelname fehlt", _
            "Geben Sie einen Artikelnamen ein."
    End With
End Sub
```

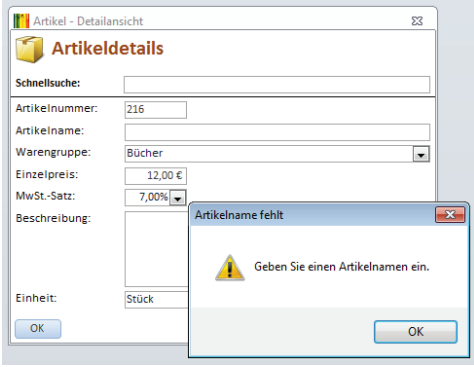


Abbildung 12.4: Validierungsmeldung

Schauen wir uns die Parameter der Methode *AddFormValidation* genauer an (IntelliSense hilft Ihnen beim Angeben der Werte für die Parameter wie in Abbildung 12.5):

» *ctl*: Verweis auf das Steuerelement, das die Validierung auslöst

» *eType*: Typ der Validierung, festgelegt in der Enumeration *eValidationType*. Mögliche Werte siehe weiter unten.

» *strTitle*: Zeichenkette, die beim Fehlschlagen der Validierung als Titel des Meldungsfensters angezeigt werden soll.

» *strMessage*: Zeichenkette, die beim Fehlschlagen der Validierung als Text des Meldungsfensters angezeigt werden soll.

» *strExpression*: Optionale Zeichenkette, die einen auswertbaren Ausdruck enthält. Liefert dieser Ausdruck den Wert *False*, ist die Validierung fehlgeschlagen.

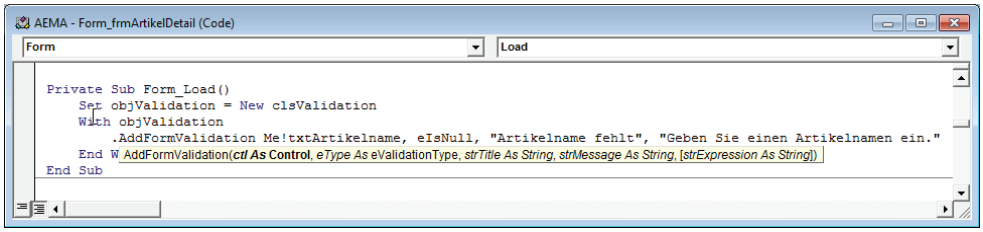


Abbildung 12.5: Angabe der Parameter für eine Validierung

Für *ctl* geben Sie einen Verweis auf das Steuerelement an, wobei Sie das Schlüsselwort *Me*, das Ausrufezeichen und den Namen des Steuerelements verwenden – also beispielsweise *Me!txtArtikelname*. Der zweite Parameter *eType* erwartet die Art der Prüfung. Hier gibt es die folgenden Werte:

» *eIsNumeric*: Prüft, ob der neue Wert ein numerischer Wert ist.

» *eIsNull*: Prüft, ob der neue Wert *Null* ist.

» *eIsDate*: Prüft, ob der neue Wert ein gültiges Datum enthält.

» *eIsNullstring*: Prüft, ob der neue Wert eine leere Zeichenkette ist.

» *eIsTrue*: Prüft, ob der neue Wert wahr ist.

» *eIsFalse*: Prüft, ob der neue Wert falsch ist.

» *eExpression*: Legt fest, dass der im Parameter *strExpression* angegebene Ausdruck das Ergebnis der Validierung liefert.

Diese Werte bietet IntelliSense beim Füllen der Parameter als Auswahlliste an (siehe Abbildung 12.6).

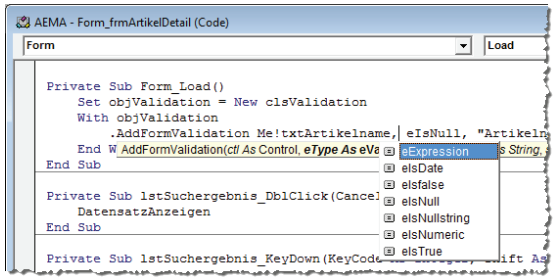


Abbildung 12.6: IntelliSense liefert alle möglichen Validierungsarten

Die Parameter *strTitle* und *strMessage* enthalten die Daten für das Meldungsfenster, das angezeigt wird, wenn die Validierung fehlschlägt. Schließlich gibt es noch einen fünften Parameter, der nur im Falle der Validierungsart *eExpression* zum Einsatz kommt. Wie Sie diesen Parameter nutzen, schauen wir uns im folgenden Beispiel an.

Validierung des Artikelnamens II

Dort soll eine Validierung sicherstellen, dass ein Artikelname mindestens drei Zeichen enthält. Dazu fügen Sie eine Validierung hinzu, die direkt nach der Eingabe feuert. Dazu verwenden Sie die Methode *AddControlValidation* wie folgt:

```
objValidation.AddControlValidation Me!txtArtikelname, eExpression, _
    "Artikelname zu kurz", "Der Artikelname muss mindestens drei Zeichen enthalten.", _
    "Len(Forms!frmArtikelDetail!txtArtikelname)<3"
```

Wichtig sind hier zwei Dinge:

» Der für die Validierung verwendete Ausdruck, hier *Len(Forms!frmArtikelDetail!txtArtikelname)<3*, muss in Anführungszeichen eingefasst werden und darf keine relativen Bezüge zu den Steuerelementen enthalten wie beispielsweise *Me!txtArtikelname*.

» Sie verwenden für solche Validierungen entweder die Validierungsklasse oder die Eigenschaften *Gültigkeitsregel*/*Gültigkeitsmeldung* der zugrunde liegenden Datenherkunft. Wenn Sie beide verwenden, erhalten Sie auch zwei Meldungen.

Die folgende Prozedur zeigt alle Validierungen für das Formular *frmArtikelDetail*:

```
Private Sub Form_Load()
    Set objValidation = New clsValidation
    With objValidation
        .AddControlValidation Me!txtEinzelpreis, eIsNumeric, "Ungültiger Einzelpreis", _
            "Geben Sie einen numerischen Wert für den Einzelpreis an."
        .AddControlValidation Me!txtArtikelname, eExpression, "Artikelname zu kurz", _
            "Der Artikelname muss mindestens drei Zeichen enthalten.", _
```

```
            "Len(Forms!frmArtikelDetail!txtArtikelname)<3"
        .AddFormValidation Me!txtArtikelname, eIsNull, "Artikelname fehlt", _
            "Geben Sie einen Artikelnamen ein."
        .AddFormValidation Me!cboWarengruppeID, eIsNull, "Warengruppe fehlt", _
            "Wählen Sie eine Warengruppe aus."
        .AddFormValidation Me!cboMehrwertsteuersatzID, eIsNull, "MwSt.-Satz fehlt", _
            "Wählen Sie einen Mehrwertsteuersatz aus."
        .AddFormValidation Me!cboEinheitID, eIsNull, "Einheit fehlt", _
            "Wählen Sie eine Einheit aus."
    End With
End Sub
```

Validierung abhängiger Steuerelemente

Dadurch, dass Sie Ausdrücke als Validierungskriterium angeben können, lassen sich auch die Inhalte mehrerer Steuerelemente einbeziehen. Ein tolles Beispiel für diesen Fall sind Datumsfelder, bei denen das Startdatum nicht hinter dem Enddatum liegen darf; da die Beispielanwendung jedoch keine solche Konstellation enthält, nehmen wir mit der folgenden vorlieb: Im Formular *frmKundeDetail* soll der Benutzer mindestens das Feld *Rechnung_Firma* oder die Felder *Rechnung_Vorname* und *Rechnung_Nachname* füllen.

Dazu deklarieren Sie auch im Kopf dieses Moduls die folgende Variable:

```
Dim objValidation As clsValidation
```

Das *Form_Load*-Ereignis ergänzen Sie wie folgt:

```
Private Sub Form_Load()
    With objValidation
        .AddFormValidation Me!txtRechnung_Firma, eExpression, "Fehlende Eingabe", _
            "Geben Sie eine Firma oder Vorname/Nachname an.", _
            "IsNull(Forms!frmKundeDetail!Rechnung_Firma) OR " _
            & "(IsNull(Forms!frmKundeDetail!Rechnung_Vorname) " _
            & "AND IsNull(Forms!frmKundeDetail!Rechnung_Nachname))"
    End With
End Sub
```

Wenn der Benutzer nun weder Firma noch Vorname/Nachname ausfüllt, erscheint die Meldung aus Abbildung 12.7.

Vorsicht beim Schließen des Formulars

Die hier vorgestellte Validierung ist in einer Beziehung mit Vorsicht zu genießen – und zwar dann, wenn die Validierung durch das Schließen eines Formulars ausgelöst wird. Dazu gibt es eine ganze Reihe von Möglichkeiten:

» das Anklicken der *Schließen*-Schaltfläche oben rechts,

» das Auswählen des Eintrags *Schließen* des Formular-Menüs (siehe Abbildung 12.8),

» das Betätigen der Tastenkombination *Strg + F4* und

» das Schließen mit dem Befehl *DoCmd.Close acForm, "<Formularname>"*.

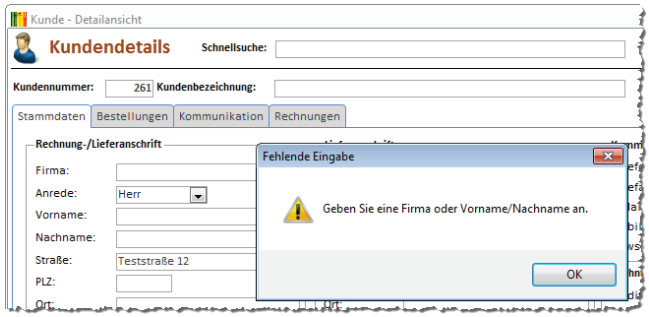


Abbildung 12.7: Meldung beim Auslassen von mindestens einem von drei Feldern

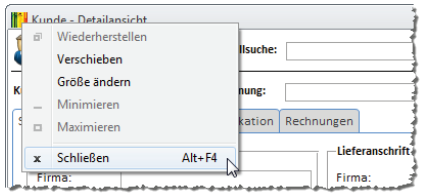


Abbildung 12.8: Schließen eines Formulars über das Formular-Menü

In all diesen Fällen erscheinen zwar noch die Validierungsmeldungen, das Formular wird jedoch dennoch geschlossen. Grundsätzlich lässt sich dieses Problem jedoch lösen. Um das Formular-Menü und die *Schließen*-Schaltfläche loszuwerden, stellen Sie die Eigenschaft *Mit Systemmenüfeld* auf *Nein* ein. Bleibt die Anweisung *DoCmd.OpenForm*, die sich normalerweise in einer Ereignisprozedur befindet, die durch das Anklicken einer Schaltfläche namens *cmdOK* ausgelöst wird. Hier sorgen wir dafür, dass das Schließen erst nach einer abschließenden Prüfung erfolgt:

```
Private Sub cmdOK_Click()
    If objValidation.Validated Then
        DoCmd.Close acForm, Me.Name
    End If
End Sub
```

Die Methode *Validated* des modulweit deklarierten Objekts *objValidation* sorgt nochmals für die Durchführung aller auf Formular- beziehungsweise Datensatzebene festgelegten Validierungen.

Reihenfolge der Validierungen

Die Validierung wird genau in der Reihenfolge durchgeführt, in der Sie die entsprechenden Anweisungen im *Form_Load*-Ereignis definiert haben. Sie sollten diese Reihenfolge an die Aktivierungsreihenfolge der Steuerelemente anpassen.

12.3.5 Validierung fehlerresistent machen

Durch eine Fehlerbehandlung, wie Sie im Kapitel beschrieben wird, sollten eigentlich alle Laufzeitfehler abgefangen werden. Dies sorgt dann auch dafür, dass die Inhalte von Objektvariablen nicht gelöscht werden.

Als Access-Entwickler wissen Sie jedoch: Niemand ist perfekt und die Benutzer decken alle Schwächen gnadenlos auf. Dementsprechend gelingt es den Benutzer sicher auch, einen Fehler in einem Formular auszulösen, dass Sie zuvor mit der Klasse *clsValidation* ausgestattet haben.

Das führt genau bei der oben beschriebenen Ereignisprozedur, die durch das Klicken auf die *OK*-Schaltfläche ausgelöst wird, zu einem Problem: Die Variable *objValidation* ist leer und dies löst einen Fehler aus. Dieser Fehler führt dazu, dass *objValidation.Validated* niemals den Wert True zurückliefert, ein Verlassen des Formulars ist somit unmöglich.

Das ändert sich auch nicht, wenn Sie vorab prüfen, ob *objValidation* noch einen Wert enthält. Sie wüssten dann zwar, dass keine Validierung mehr möglich ist, aber was dann? Das Formular einfach ohne Validierung verlassen? Nein, das ist keine Option. Also greifen wir zu härteren Maßnahmen: Das Objekt *objValidation* wird fehlerresistent gespeichert.

Schauen wir uns zum Beispiel das Formular *frmArtikelDetail* an. Dieses instanziert in der Prozedur, die durch das Ereignis *Beim Laden* des Formular ausgelöst wird, ein Objekt auf Basis der Klasse *clsValidation* und legt mit der Methode *AddFormValidation* eine Validierung an.

Danach folgt der entscheidende Schritt: Die Funktion *ObjPtr* ermittelt den Pointer auf dieses Objekts und weist den gefundenen Wert einer neuen temporären Variablen zu. Damit merkt sich die Anwendung die Speicheradresse des Objekts. Der Clou dabei: Die Inhalte temporärer Variablen auf Basis der *TempVars*-Auflistung werden beim Auftreten unbehandelter Laufzeitfehler nicht gelöscht:

```
Private Sub Form_Load()
    ...
    Set objValidation = New clsValidation
    With objValidation
        .AddFormValidation Me!txtRechnung_Firma, eExpression, "Fehlende Eingabe", _
            "Geben Sie eine Firma oder Vorname/Nachname an.", _
            "IsNull(Forms!frmKundeDetail!Rechnung_Firma) AND " _
            & " (IsNull(Forms!frmKundeDetail!Rechnung_Vorname) " _
```

```
        & "AND IsNull(Forms!frmKundeDetail!Rechnung_Nachname))"
    End With
    TempVars.Add "Validation_frmKundeDetail", ObjPtr(objValidation)
    ...
End Sub
```

Nun gibt es einen Zeitpunkt, zu dem wir unbedingt auf das Objekt *objValidation* zugreifen müssen – nämlich beim Schließen des Formulars, was durch das Betätigen der *OK*-Schaltfläche erfolgt.

Die dadurch ausgelöste Prozedur prüft nun, ob das Objekt *objValidation* leer ist und füllt es gegebenenfalls erneut mit einem Verweis auf Objekt, dessen Speicheradresse wir ja temporär gespeichert haben. Diese Aufgabe übernimmt die Funktion *RebuiltObject*, die den Namen der temporären Variablen mit der Speicheradresse als Parameter erwartet:

```
Private Sub cmdOK_Click()
    If objValidation Is Nothing Then
        RebuildObject "Validation_frmKundeDetail"
    End If
    If objValidation.Validated Then
        DoCmd.Close acForm, Me.Name
    End If
End Sub
```

Danach kann die Prozedur wie üblich auf das Objekt *objValidation* zugreifen. Wie sieht nun die Funktion *RebuildObject* aus? Diese erhält den Namen der temporären Variablen, in der wir die mit *ObjPtr* ermittelte Speicheradresse des Objekts gesichert haben, als Parameter und liefert einen Verweis auf das betroffene Objekt zurück – dies alles unter Einsatz der API-Funktion *CopyMemory*:

```
Public Function RebuildObject(strObject As String) As Object
    Dim lngObject As Long
    Dim obj As Object
    lngObject = TempVars(strObject)
    If lngObject <> 0 Then
        CopyMemory obj, lngObject, 4&
        Set RebuildObject = obj
        CopyMemory obj, 0&, 4&
    End If
End Function
```

Dabei kommt schließlich noch die folgende API-Funktion zum Einsatz:

```
Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, _
    Source As Any, ByVal Length As Long)
```

13 Outlook

Outlook bietet wie Word oder Excel die Möglichkeit der Fernsteuerung von einer Access-Anwendung aus. Es weist genau wie die übrigen Office-Anwendungen eine Objektbibliothek auf, über die Sie auf das Objektmodell von Outlook zugreifen können. Damit lassen sich eine Menge Aufgaben erledigen: Sie können damit E-Mails verschicken oder per Outlook erhaltene E-Mails einlesen, Termine anlegen oder auslesen oder Kontakte synchronisieren.

Das sind nur die meistgefragten Aufgaben – es gibt noch weitere Möglichkeiten wie etwa das Übertragen von Aufgaben von Access nach Outlook und umgekehrt. In diesem Kapitel schauen wir uns zunächst die Möglichkeiten von E-Mails in Zusammenhang mit unserer Kundenverwaltung und Outlook an. Dies ist ein wichtiger Baustein der Anwendung, denn mit den hier vorgestellten Grundlagen übertragen Sie die komplette E-Mail-Kommunikation von Outlook in Ihre Kundenverwaltung.

Wie ich aus eigener Erfahrung weiß, ist es ungemein wichtig, die Historie der bisherigen Kommunikation mit einem Kunden vorliegen zu haben, wenn eine neue Anfrage eingeht. In diesem Kapitel betrachten wir zunächst die technische Abwicklung unter Outlook, die im Kapitel »Kommunikation verwalten« (Seite 255) bereits von Anwendungsseite aus beschrieben wurde.

13.1 Outlook fernsteuern

Wenn Sie mit VBA auf die Objektbibliothek und somit auf die Objekte, Methoden, Eigenschaften und Ereignisse von Outlook zugreifen möchten, benötigen Sie zunächst einen Verweis auf die entsprechende Objektbibliothek. Diesen fügen Sie auf einfache Weise im VBA-Editor hinzu, den Sie von Access aus entweder mit *Alt + F11* oder *Strg + G* öffnen (Letzteres aktiviert gleich das Direktfenster, was hier allerdings nicht nötig ist).

Im VBA-Editor wählen Sie den Menüeintrag *Extras*|*Verweise* aus. Dort finden Sie die bereits gesetzten Verweise, die es überhaupt erst ermöglichen, dass Sie die VBA-Befehle, die DAO-Objekte und die Access-Objekte nutzen können.

Um nun auch noch Zugriff auf die Outlook-Objektbibliothek zu erhalten, wählen Sie aus der Liste den Eintrag *Microsoft Outlook 12.0 Object Library* beziehungsweise *Microsoft Outlook 14.0 Object Library* hinzu (siehe Abbildung 13.1).

Während es im Falle von Word oder Excel nicht relevant ist, müssen Sie unter Outlook explizit die Version der jeweils verfügbaren Objektbibliothek angeben. Das ist zumindest die beste Lösung, während Sie die Anwendung entwickeln. Dafür gibt es gleich mehrere Gründe:

» Sie können im VBA-Editor per IntelliSense auf die Methoden der jeweiligen Objektbibliothek zugreifen (Abbildung 13.2 zeigt ein Beispiel für den Einsatz von IntelliSense im VBA-Editor).

» Sie finden die Elemente der Objektbibliothek im Objektkatalog.

» Sie können Konstanten statt der dahinter stehenden Werte angeben, also beispielsweise *olMailItem* statt *0* – was wesentlich aussagekräftiger ist.

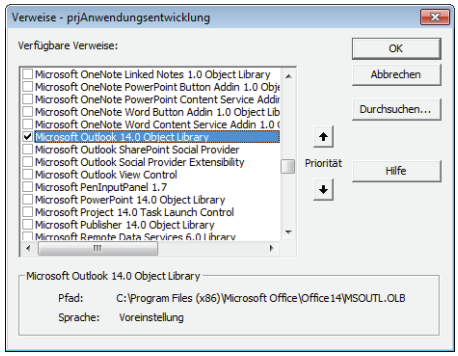


Abbildung 13.1: Hinzufügen eines Verweises auf die Outlook-Objektbibliothek

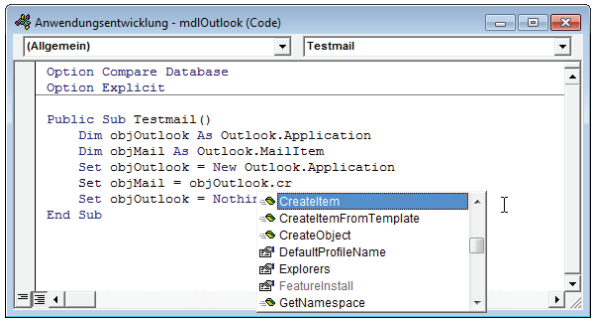


Abbildung 13.2: IntelliSense beim Zugriff auf Objekte des Outlook-Objektmodells

Allerdings führt es beispielsweise zu Problemen, wenn Sie eine Datenbank mit einem Verweis auf die Version 14.0 dieser Bibliothek an jemanden weitergeben, der nur die Version 12.0 der Bibliothek auf dem Rechner installiert hat. Wir zeigen in den kommenden Abschnitten jeweils, wie Sie mit der Objektbibliothek arbeiten und wie es ohne funktioniert.

Für die Zeit der Entwicklung der Anwendung können Sie jedoch ohne Probleme mit dem entsprechenden Verweis arbeiten, erst vor der Weitergabe der Anwendung sollten Sie den Verweis entfernen und die nachfolgend erläuterten Änderungen am Code vornehmen.

13.1.1 Outlook referenzieren

Für die meisten Aktionen, die in diesem Kapitel beschrieben werden, ist VBA-Code nötig. Immerhin möchten Sie automatisiert auf die Objektbibliothek von Outlook zugreifen, um bei-

spielsweise E-Mails zu versenden – und sollten Sie nicht gerade einen gut programmierten Roboter an der Tastatur sitzen haben, der diese Aufgabe für Sie erledigt, werden Sie ohne die folgenden VBA-Prozeduren kaum auskommen. Um auf das Objektmodell von Outlook zuzugreifen, müssen Sie zunächst einmal eine entsprechende Variable deklarieren. Diese heißt *objOutlook* und hat den Datentyp *Outlook.Application*:

```
Dim objOutlook As Outlook.Application
```

Die folgende Anweisung füllt diese Variable mit einem Verweis auf eine Outlook-Instanz, egal ob Outlook bereits gestartet ist oder ob dies zu diesem Zweck noch nötig ist:

```
Set objOutlook = New Outlook.Application
```

Nun können Sie gleich auf das Objektmodell von Outlook zugreifen – zum Beispiel, um sich davon zu überzeugen, dass Sie auch einen Verweis auf Outlook erhalten haben und dazu den Namen des Objekts *objOutlook* ausgeben:

```
Dim objOutlook As Outlook.Application
Set objOutlook = New Outlook.Application
Debug.Print objOutlook.Name
Set objOutlook = Nothing
```

Dies holt einen Verweis auf die bestehende oder eine neue Outlook-Instanz, gibt den Namen der referenzierten Anwendung aus (hier *Outlook*) und leert den Verweis wieder. Wenn Outlook soeben neu gestartet wurde, beendet *Set objOutlook = Nothing* Outlook gleich wieder. Wenn schon eine Instanz gestartet war, wird nur die Referenz gelöscht, die Instanz bleibt davon unberührt. Dass die obigen Codezeilen, ausgeführt etwa als Teil einer Prozedur, beim Fehlen einer aktiven Outlook-Instanz Outlook starten und auch wieder beenden, können Sie gut im Task-Manager von Windows nachvollziehen. Diesen starten Sie am schnellsten über den Kontextmenü-Eintrag *Task-Manager starten* der Taskleiste von Windows. Eine Outlook-Instanz finden Sie etwa wie in Abbildung 13.3 vor.

13.1.2 Late Binding

Beim Programmieren ohne einen Verweis auf die Outlook-Objektbibliothek können Sie logischerweise nicht auf Datentypen wie etwa *Outlook.Application* zugreifen – der VBA-Editor kennt nur solche Datentypen, die entweder in referenzierten Objekten definiert sind oder die Sie selbst definiert haben. Sie müssen solche Objekte dann mit dem allgemeinen Datentyp *Object* deklarieren:

```
Dim objOutlook As Object
```

Und natürlich können Sie das Objekt auch nicht als neue Instanz der Klasse *Outlook.Application* erzeugen, sondern müssen die Klasse als Parameter der *CreateObject*-Instanz angeben:

```
Set objOutlook = CreateObject("Outlook.Object")
```

Da Outlook eine Multi-Use-Instanz ist, können Sie immer mit *CreateObject* arbeiten – auch, wenn bereits ein Outlook-Objekt existiert (beispielsweise die Instanz, mit der Sie gerade Ihre Termine bearbeiten). Bei anderen Office-Anwendungen wie beispielsweise Word oder Excel läuft dies etwas anders – mehr dazu unter »Individuelle Anschreiben mit Word« (Seite 341).

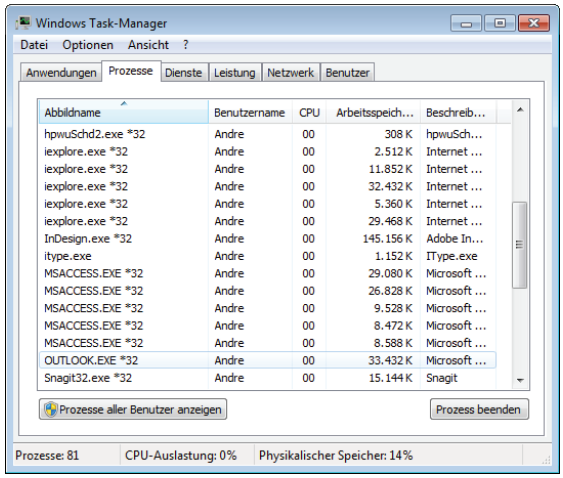


Abbildung 13.3: Task-Manager mit einer Outlook-Instanz

Wechseln zwischen Early Binding und Late Binding

Wenn sich die Entwicklung Ihrer Anwendung in einer Phase befindet, in der Sie häufig neue Versionen für den Kunden ausliefern, aber dennoch an Teilen der Anwendung entwickeln, bei denen Verweise wie etwa auf die Outlook-Bibliothek hilfreich sind, müssen Sie gegebenenfalls manuell zwischen den verschiedenen Ansätzen wechseln. Das heißt, dass Sie nach dem Entwickeln und vor dem Ausliefern jedesmal den Outlook-Verweis entfernen und Zeilen wie

```
Dim objOutlook As Outlook.Application
```

in diese umwandeln:

```
Dim objOutlook As Object
```

Ich selbst halte meist beide Varianten im Code vor und kommentiere die jeweils nicht benötigte Zeile aus:

```
Dim objOutlook As Outlook.Application
'Dim objOutlook As Object
```

Um das Entfernen und Hinzufügen des Verweises kommen Sie nicht herum, aber das Ein- und Auskommentieren der beiden oben erwähnten Zeilen können Sie sich mit der hilfreichen Technik der bedingten Kompilierung sparen. Dabei legen Sie zunächst etwa in einer Konstanten den Wert einer Bedingung fest:

```
#Const olReference = True
```

olReference bedeutet in diesem Fall: Die Referenz ist vorhanden (*True*) oder eben nicht (*False*). Ist die Referenz, also der Verweis auf die Outlook-Objektbibliothek, vorhanden und hat olReference somit den Wert *True*, wird in der folgenden kleinen Beispielprozedur der erste Teil der *If*-Bedingung ausgeführt, anderenfalls der zweite Teil:

```
Public Sub InstanzierenMitUndOhneVerweis()
    #If olReference = True Then
        Dim objOutlook As Outlook.Application
        Set objOutlook = New Outlook.Application
        Debug.Print "Early Binding: " & objOutlook.Name
        Set objOutlook = Nothing
    #Else
        Dim objOutlook As Object
        Set objOutlook = CreateObject("Outlook.Application")
        Debug.Print "Late Binding: " & objOutlook.Name
        Set objOutlook = Nothing
    #End If
End Sub
```

Der erste Teil bezieht sich explizit auf Objekte der Outlook-Bibliothek, der zweite Teil arbeitet so, als ob der Verweis nicht vorhanden wäre. Warum aber werden sowohl die Konstantendeklaration als auch die Bausteine des *If...Then...Else*-Konstrukts mit führenden *#*-Zeichen ausgestattet? Ganz einfach: Weil es sich hier um Anweisungen zum bedingten Kompilieren handelt. Das heißt, dass beim Kompilieren des Codes, bei dem unter anderem auch Syntaxfehler aufgedeckt werden, nur der Code kompiliert wird, für den die jeweilige Bedingung wahr ist. Wenn das VBA-Projekt keinen Verweis auf die Outlook-Bibliothek enthält, würde beispielsweise die Zeile

```
Dim objOutlook As Outlook.Application
```

zu einem Kompilierfehler führen – siehe Abbildung 13.4. Wenn Sie hingegen die Konstante *olReference* auf *False* einstellen und dem Kompiler somit das Signal geben, nur den zweiten Teil der *If...Then...Else*-Bedingung zu prüfen, lösen die Anweisungen im ersten Teil keinen Fehler aus. Dies ist nur eine kleine Anregung, wie Sie sich das möglicherweise nervtötende Ein- und Auskommentieren sparen können. Doch kommen wir nun zu den eigentlich interessanten Möglichkeiten des Outlook-Objektmodells – beispielsweise dem Senden von E-Mails.

13.2 E-Mails senden

Wenn Sie schon Kunden, Artikel und Bestellungen in einer Kundenverwaltung beherbergen, sollten Sie an dieser Stelle auch Zugriff auf die Kommunikation mit dem Kunden haben. Wir gehen an dieser Stelle davon aus, dass die Kommunikation ausschließlich über E-Mail erfolgt. Sollten

Sie oder Ihre Kunden diesen einfachen Weg ebenfalls anstreben, seien Sie nicht allzu großzügig mit der Publikation Ihrer Telefonnummer ...

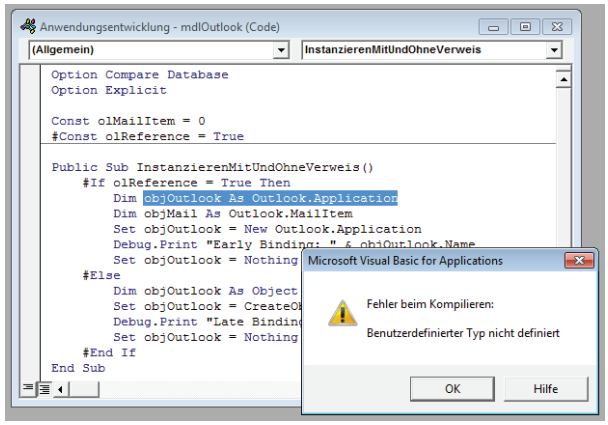


Abbildung 13.4: Kompilierfehler bei Verwendung eines nicht definierten Datentyps

Das Senden einer E-Mail mit Outlook dürfte Ihnen bekannt sein – Sie klicken auf die Schaltfläche zum Erstellen einer neuen E-Mail, geben den Empfänger, den Betreff und den Inhalt ein und hängen gegebenenfalls noch Dateien an. Diesen Vorgang wollen wir im Rahmen dieses Buchs so weit wie möglich übernehmen.

Das bedeutet jedoch nicht, dass wir hier ein eigenes Formular erstellen, das den Outlook-Dialog zum Erstellen einer neuen E-Mail möglichst genau abbildet! Das wäre zwar auch einfach möglich, aber warum sollen wir das Rad neu erfinden?

Outlook 2007/2010 bietet alle Möglichkeiten, direkt den Outlook-Dialog zum Versenden einer E-Mail zu öffnen, diesen automatisch mit Informationen wie Empfänger, Betreff, Inhalt und Anlagen zu füllen und diesen dann zu verschicken. Dabei soll es zwei Möglichkeiten geben:

» Die E-Mail wird direkt nach dem automatischen Erstellen verschickt, ohne sie anzuzeigen.

» Die E-Mail wird vor dem Versenden noch angezeigt, damit der Benutzer den Inhalt manuell anpassen kann.

Wozu diese beiden Varianten? Die erste Variante ist beispielsweise sinnvoll, wenn es um das Versenden einer Rechnung im PDF-Format geht. Sie verwenden dann einen Standardtext, hängen das von der Kundenverwaltung erzeugte Rechnungsdokument an und können die E-Mail versenden, ohne diese vorher noch zu kontrollieren (am Anfang werden Sie dies tun, aber glauben Sie mir: wenn Sie einmal erkannt haben, dass dies reibungslos funktioniert, werden Sie jeden unnötigen Arbeitsschritt vermeiden).

Die zweite Variante ist etwa für individuelle Kundenanschreiben interessant. Dazu soll das Kundenformular eine eigene Schaltfläche erhalten, mit der Sie eine E-Mail öffnen können, die

bereits die richtige E-Mail-Adresse und gegebenenfalls einen Standardtext enthält, den Sie aber nach Belieben anpassen können.

13.2.1 Einfache E-Mails erstellen

Genau wie mit einem Mausklick unter Outlook können Sie mit wenigen VBA-Zeilen eine neue E-Mail kreieren. Wenn Sie einfach nur eine neue E-Mail öffnen möchten, verwenden Sie die folgenden Codezeilen:

```
Dim objOutlook As Outlook.Application
Dim objMail As Outlook.MailItem
Set objOutlook = New Outlook.Application
Set objMail = objOutlook.CreateItem(olMailItem)
objMail.Display
Set objOutlook = Nothing
```

Diese Zeilen erstellen eine neue Outlook-Instanz (oder holen eine bestehende), erzeugen mit der *CreateItem*-Methode ein neues Objekt des Typs *MailItem* und zeigen die E-Mail an (siehe Abbildung 13.5). Damit ist der Fall für den aufrufenden Code erledigt, der Benutzer und die jungfräuliche E-Mail bleiben sich selbst überlassen.

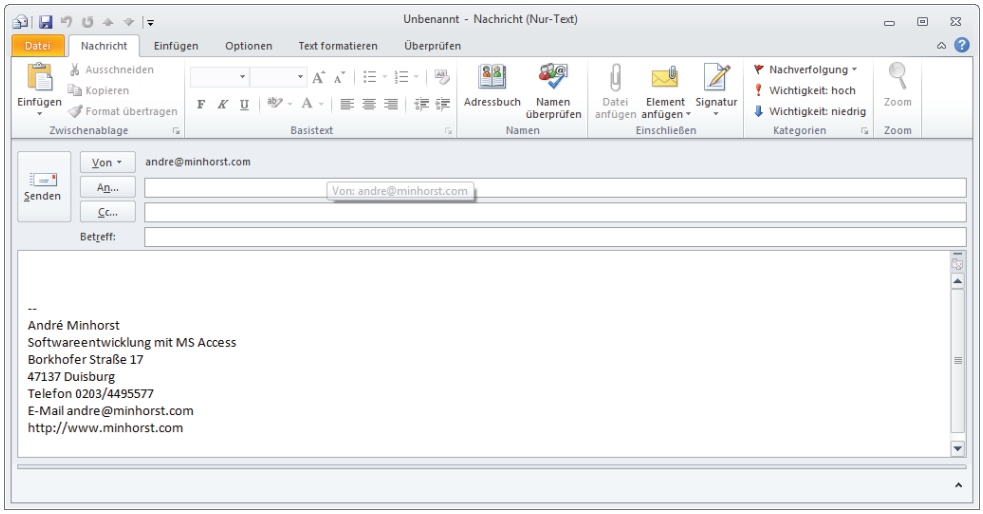


Abbildung 13.5: Eine jungfräuliche E-Mail

Sicherheitsmeldung beim Aufruf per VBA unterbinden

Unter Outlook 2007 und 2010 taucht, sofern Sie noch keine Änderungen an Ihrem System vorgenommen haben, gleich eine Sicherheitsmeldung auf, die erst nach einigen Sekunden das Freigeben des Mailversands ermöglicht (siehe Abbildung 13.6).

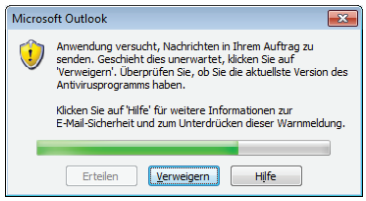


Abbildung 13.6: Sicherheitsmeldung beim Versuch, eine E-Mail per VBA zu versenden

Ab Access 2007 können Sie diese Meldung jedoch deaktivieren. Dazu öffnen Sie Outlook zunächst als Administrator – dies geht am einfachsten, indem Sie Ihre Outlook-Verknüpfung im Startmenü oder an anderer Stelle bei gedrückter Tastenkombination *Strg + Umschalt* anklicken.

Aktivieren Sie dann das Vertrauensstellungscenter, was bei Outlook 2007 und 2010 unterschiedlich funktioniert:

» Unter Outlook 2007 wählen Sie den Menüeintrag *Extras|Vertrauensstellungscenter* aus.

» Unter Outlook 2010 öffnen Sie die Outlook-Optionen, wechseln zum Bereich *Sicherheitscenter* und klicken dort auf die Schaltfläche *Einstellungen für das Sicherheitscenter*.

Im nun erscheinenden Dialog *Vertrauensstellungscenter* (Outlook 2007) beziehungsweise *Sicherheitscenter* (Outlook 2010) wechseln Sie zum Bereich *Programmgesteuerter Zugriff*, wo Sie die Option *Bei verdächtigen Aktivitäten nie Warnhinweis anzeigen (nicht empfohlen)* auswählen (siehe Abbildung 13.7).

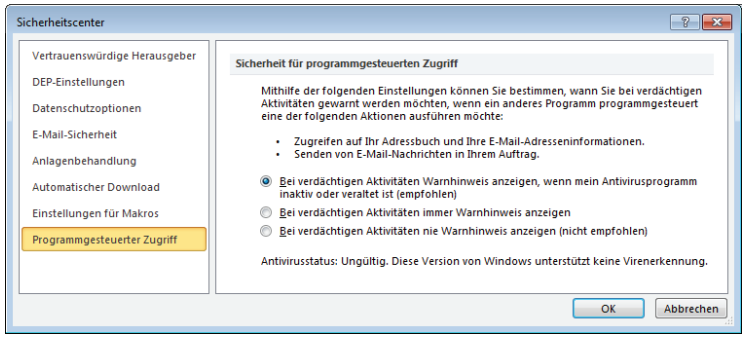


Abbildung 13.7: Deaktivieren der Sicherheitsmeldung beim Versenden von E-Mails

13.2.2 E-Mail-Versand komplett automatisieren

Damit die Anwendung auf Mausklick komplett selbstständig E-Mails versendet, brauchen Sie noch einige weitere Einstellungen. Wir wollen die komplette Funktion zum Versenden von E-Mails in einer Klasse unterbringen. Legen Sie dazu im VBA-Editor mit *Einfügen|Klassenmodul* ein neues Klassenmodul hinzu und speichern Sie es unter dem Namen *clsMail*.

Bevor wir diese Klasse füllen, klären wir noch, welche Features einer E-Mail wir nutzen möchten:

» Empfänger (To)

» Empfänger einer Kopie (CC)

» Empfänger einer blinden Kopie (BCC)

» Absender (From)

» Betreff (Subject)

» Inhalt (Body)

» Format des Inhalts (Plain, HTML)

» Anhänge (Attachment)

» Verschieben in einen speziellen Ordner nach dem Versenden

» Direktes Versenden einer E-Mail

» Erstellen einer E-Mail und Versenden nach manuellen Anpassungen

» Auslesen der Änderungen nach dem Versenden

Die Anforderungen sind nicht unbedingt umfangreich, haben es aber teilweise in sich.

13.2.3 Beschreibung der Klasse clsMail

Die Klasse *clsMail* soll direkt beim Instanzieren Outlook öffnen und eine E-Mail erstellen, die dann von den folgenden Zeilen mit den nötigen Informationen gefüllt und abgesendet wird. Beim Instanzieren einer Klasse wird die folgende Ereignisprozedur ausgelöst, die Sie durch Auswählen des Eintrags *Class* im linken Kombinationsfeld des Code-Fensters der Klasse hinzufügen:

```
Private Sub Class_Initialize()
    Set objOutlook = New Outlook.Application
    Set objMail = objOutlook.CreateItem(olMailItem)
End Sub
```

Die Klasse füllt zunächst die Variable *objOutlook* mit einem Verweis auf eine frisch erstellte Instanz von Outlook beziehungsweise auf eine bereits geöffnete Instanz.

Danach erstellt sie mit der *CreateItem*-Methode und dem Parameter *olMailItem* eine neue E-Mail und verweist mit der Variablen *objMail* darauf. Die beiden Variablen deklarieren Sie wie folgt im Kopf der E-Mail:

```
Private objOutlook As Outlook.Application
Private WithEvents objMail As Outlook.MailItem
```

Warum *objMail* mit *WithEvents* deklariert wird, erfahren Sie weiter unten. Zunächst noch ein Blick auf das Ereignis, das beim Zerstören des Objekts auf Basis der Klasse *clsMail* ausgelöst wird. Dieses leert die Variablen zum Speichern der Outlook- und der MailItem-Referenz:

```
Private Sub Class_Terminate()
    Set objMail = Nothing
    Set objOutlook = Nothing
End Sub
```

Danach schauen wir uns an, wie *Property Let*-Prozeduren aussehen, mit denen Sie die Eigenschaften der frisch erstellten E-Mail einstellen können. Die ersten paar nehmen einfach die Werte entgegen, die von der instanzierenden Prozedur übergeben werden, und weisen diese direkt den entsprechenden Eigenschaften des *MailItem*-Objekts zu – zum Beispiel für den Betreff oder den Inhalt:

```
Public Property Let Betreff(strSubject As String)
    objMail.Subject = strSubject
End Property

Public Property Let Inhalt(strBody As String)
    objMail.Body = strBody
End Property
```

Mit den folgenden beiden *Property Let*-Prozeduren stellen Sie ein, ob der Empfänger beim Öffnen der E-Mail eine Empfangsbestätigung abschicken soll und welche Wichtigkeit die E-Mail hat:

```
Public Property Let Empfangsbestaetigung(bolReadReceiptRequested As Boolean)
    objMail.ReadReceiptRequested = bolReadReceiptRequested
End Property

Public Property Let Wichtigkeit(intImportance As OlImportance)
    objMail.Importance = intImportance
End Property
```

Etwas aufwendiger ist die *Property Let*-Prozedur, die den Absender der E-Mail entgegennimmt. Diese muss zunächst den Account von Outlook finden, welcher der angegebenen E-Mail-Adresse entspricht. Wurde dieser gefunden, stellt die Prozedur die Eigenschaft *SendUsingAccount* auf das gefundene *Account*-Objekt ein. Wird kein *Account*-Objekt gefunden, zeigt die Prozdur eine entsprechende Meldung an:

```
Public Property Let Von(strFrom As String)
    Dim objAccount As Outlook.Account
    For Each objAccount In objOutlook.Session.Accounts
        If objAccount.SmtpAddress = strFrom Then
```

14 Fehlerbehandlung

Die Fehlerbehandlung hat zwei wichtige Funktionen: Erstens soll sie gewährleisten, dass eine Anwendung auch noch nach dem Auftreten eines Laufzeitfehlers stabil weiterläuft und nicht etwa abstürzt. Zweitens soll sie Informationen bereitstellen, um den Fehler zu analysieren und zu beheben. In diesem Buch lernen Sie zwei Möglichkeiten der Fehlerbehandlung kennen. Die klassische Variante implementieren Sie direkt in den Prozeduren der Anwendung. Der Aufwand ist nicht unerheblich: Sie müssen dazu jede einzelne Prozedur mit einigen zusätzlichen Zeilen ausstatten und außerdem alle Zeilen mit Zeilennummern versehen. Auf diese Weise können Sie die Fehlerinformationen um die Angabe der Zeile erweitern, in welcher der Fehler aufgetreten ist.

Die zweite Möglichkeit ist die Fehlerbehandlung mit *vbWatchdog*. Dabei handelt es sich um eine Art DLL, die sich in die Fehlerbehandlung von VBA einklinkt und so Fehler behandelt, ohne dass Sie der entsprechenden Prozedur überhaupt eine Fehlerbehandlung hinzufügen müssen.

14.1 Klassische Fehlerbehandlung

Die klassische Fehlerbehandlung sieht vor, beim Auftreten eines Laufzeitfehlers nicht die in VBA eingebaute Standardfehlermeldung anzuzeigen, sondern eine eigene Fehlerbehandlung zu implementieren. Nehmen wir die folgende, einfache Prozedur als Beispiel, die sich in der Beispieldatenbank im Modul *mdlFehlerbehandlung_Klassisch* befindet:

```
Public Sub Beispielfehler()
    Debug.Print 1 / 0
End Sub
```

Das Ausführen dieser Prozedur löst den Fehler aus Abbildung 14.1 aus.

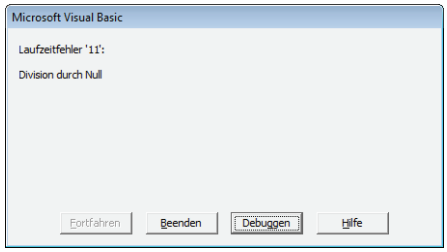


Abbildung 14.1: Fehler beim Dividieren durch die Zahl 0

Mit den eingebauten Fehlermeldungen kann der Benutzer erstens je nach Fehler wenig anfangen. Außerdem wird die Ausführung des Codes unterbrochen und VBA löscht den Inhalt von

Variablen. Dies kann insbesondere kritisch sein, wenn Sie Objekte mit globalen oder modulweit deklarierten Variablen referenzieren – diese sind nach dem Auftreten eines unbehandelten Fehlers anschließend leer.

Das Voranstellen der folgenden Anweisung sorgt dafür, dass die Fehlermeldung ausbleibt und die Variablen ihre Werte behalten:

```
On Error Resume Next
```

Allerdings erfahren weder Nutzer noch Entwickler vom Auftreten des Fehlers. Um es kurz zu machen, finden Sie hier die standardmäßig verwendete Fehlerbehandlung inklusive Zeilennummerierung:

```
Public Sub BeispielfehlerMitFehlerbehandlung()
10        On Error GoTo Fehler
20        Debug.Print 1 / 0
Ende:
30        On Error Resume Next
          'Hier finale Anweisungen
40        Exit Sub
Fehler:
50        ErrNotify Err, "mdlFehlerbehandlung_Klassisch", _
              "BeispielfehlerMitFehlerbehandlung"
60        Resume Ende
End Sub
```

Diese Prozedur enthält gegenüber der vorherigen Variante ohne Fehlerbehandlung die folgenden Erweiterungen:

» Die Anweisung *On Error Goto Fehler* sorgt dafür, dass die Prozedur nach dem Auftreten eines Fehlers weiter unten an der mit *Fehler:* gekennzeichneten Stelle fortgeführt wird.

» Unter *Fehler:* wird die Prozedur *ErrNotify* aufgerufen, wobei ein Verweis auf das *Err*-Objekt mit den Fehlerinformationen sowie der Name des Moduls und der Prozedur mit dem Fehler übergeben werden. Das *Err*-Objekt liefert mit seinen Eigenschaften *Number* und *Description* eine Fehlernummer und eine Beschreibung.

» Nach dem Aufruf von *ErrNotify* wird die Prozedur mit dem Teil hinter der Marke *Ende:* fortgeführt. Hier bringen Sie zum ordnungsgemäßen Abschluss der Prozedur nötige Anweisungen unter, etwa zum Schließen von zuvor geöffneten Dateien oder zum Leeren von Objektvariablen.

» Die hinter der Markierung *Ende:* angeführten Anweisungen werden auch bei fehlerfreiem Verlauf der Prozedur erreicht. Vor der Markierung *Fehler:* befindet sich jedoch eine *Exit Sub*-Anweisung, damit die Prozedur bei fehlerfreier Ausführung nicht mit dem Auslösen der Fehlerbehandlung beendet wird.

» Schließlich finden Sie noch die Zeilennummerierungen vor. Diese werden für alle Zeilen angelegt, die ausgeführt werden. Es gibt ein paar Ausnahmen, zum Beispiel die Prozedurköpfe, Deklarationszeilen und die *Case*-Zeilen in *Select Case*-Anweisungen.

Um die *ErrNotify*-Prozedur kümmern wir uns weiter unten. Zunächst interessiert uns, wie wir eine Fehlerbehandlung ohne allzugroßen Aufwand zu allen Routinen der Anwendung hinzufügen – je nach Umfang der Anwendung kann dies eine ganze Menge sein.

Die gute Nachricht ist: Das Hinzufügen der für die Fehlerbehandlung nötigen Codezeilen und das Nummerieren der Routinen kostet uns schlappe zwei Mausklicks. Dazu müssen Sie jedoch die bereits erwähnte Software *MZ-Tools* installiert haben.

Wenn dies der Fall ist und Sie die im Download befindliche Datei *MZTools3VBA.ini* im entsprechenden Verzeichnis gespeichert haben, brauchen Sie im VBA-Editor nur noch die Einfügemarke in der mit der Fehlerbehandlung auszustattenden Prozedur zu platzieren und auf die Schaltfläche *Fehlerbehandlung hinzufügen* zu klicken (siehe Abbildung 14.2).

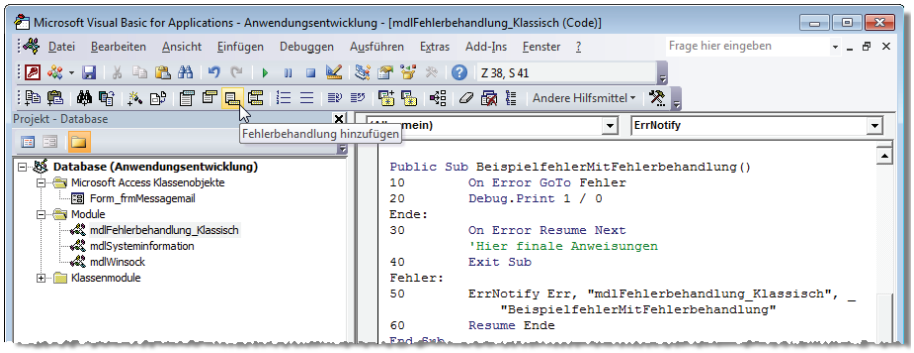


Abbildung 14.2: Hinzufügen der Fehlerbehandlung per Mausklick

Gleich zwei Schaltflächen daneben finden Sie die Schaltfläche *Zeilennummern hinzufügen* – Sie ahnen bereits, welche Aufgabe das Betätigen dieser Schaltfläche für Sie erledigt. Mit der Schaltfläche *Zeilennummern entfernen* werden Sie die Zeilennummern wieder los, wenn Sie beispielsweise den Code ändern oder erweitern möchten.

14.1.1 Fehlermeldung anzeigen

Die folgende einfache Variante der Routine *ErrNotify* gibt die Meldung aus Abbildung 14.3 aus:

```
Sub ErrNotify(AErr As VBA.ErrObject, strModule As String, strProc As String)
    MsgBox "Fehler in Modul " & strModule & ", Routine " & strProc _
        & " in Zeile " & Erl & "." & vbCrLf _
        & "Fehlermeldung: " & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number
End Sub
```

Die Prozedur nimmt einen Verweis auf das durch den Fehler gefüllte *Err*-Objekt entgegen sowie den Namen des Moduls und der Prozedur. Das *Err*-Objekt liefert mit den beiden Eigenschaften *Number* und *Description* die Fehlernummer und die Beschreibung. Die Zeilennummer ermittelt die nicht dokumentierte *Erl*-Funktion. Diese Informationen werden zu einer Zeichenkette zusammengefasst und per *MsgBox*-Anweisung ausgegeben.

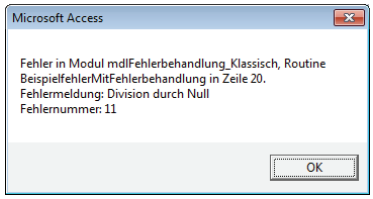


Abbildung 14.3: Beispielfehlermeldung mit Modul, Routine, Zeile, Meldung und Fehlernummer

14.1.2 Fehlermeldung per E-Mail versenden

Wenn die Fehlermeldung beim Entwickler landen soll, damit dieser den Fehler gleich reproduzieren und beheben kann, versenden Sie die notwendigen Informationen am besten gleich automatisch per E-Mail. Allerdings sollten Sie den Benutzer vorab darüber informieren, welche Daten nun an den Entwickler geschickt werden. Dies erledigen Sie, indem Sie diese Daten in einem Formular anzeigen (siehe Abbildung 14.4) und den Benutzer die E-Mail per Mausklick absenden lassen.

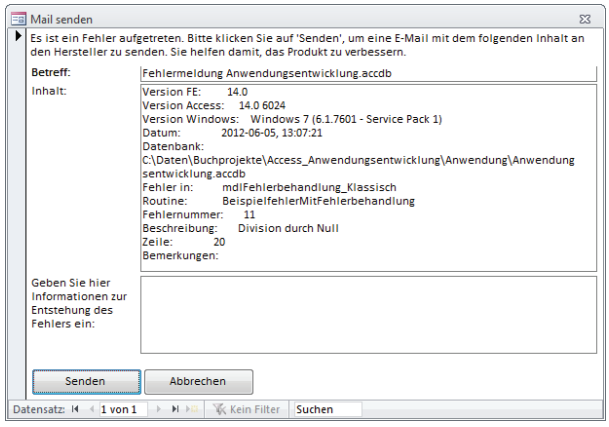


Abbildung 14.4: Formular zum Absenden einer Fehlermeldung an den Entwickler der Software

Das Textfeld des Formulars füllt die *ErrNotify*-Prozedur. Diese ist ähnlich aufgebaut wie die vorherige Variante, die lediglich ein Meldungsfenster mit den Fehlerinformationen anzeigt. Allerdings ermittelt diese Prozedur neben den übergebenen Fehlerinformationen noch einige weitere Daten. Dazu gehört die Version der Anwendung, die mit der *DLookup*-Funktion

aus der Tabelle *tblOptionen* ausgelesen wird. Dahinter finden Sie eine auskommentierte Zeile, welche die Version des Backends hinzufügen würde. Die Access-Version ermittelt die nicht dokumentierte Funktion *SysCmd* mit dem Wert *715* als Parameter. Für die Ermittlung der Windows-Version ist eine benutzerdefinierte Funktion namens *GetWindowsVersion* nötig, die Sie im Modul *mdlSysteminformation* finden. Eine ausführliche Beschreibung sparen wir uns an dieser Stelle. Danach folgen die mit den Parametern der *ErrNotify*-Prozedur übergebenen Informationen:

```
Sub ErrNotify(AErr As VBA.ErrObject, strModule As String, strProc As String, _
        Optional strRemarks As String)
    Dim strMessage As String
    Dim strErrNumber As String
    Dim strErrDescription As String
    Dim strErl As String
    strErrNumber = AErr.Number
    strErrDescription = AErr.Description
    strErl = Erl
    strMessage = strMessage & "Version FE:         " _
        & Nz(DLookup("Version_FE", "tblOptionen"), "") & vbCrLf
'   strMessage = strMessage & "Version BE:         " _
        & Nz(DLookup("Version_BE", "tblOptionen_BE"), "") & vbCrLf
    strMessage = strMessage & "Version Access:     " _
        & Access.Version & " " & SysCmd(715) & vbCrLf
    strMessage = strMessage & "Version Windows:    " & GetWindowsVersion & vbCrLf
    strMessage = strMessage & "Datum:              " _
        & Format(Now, "yyyy-mm-dd, hh:nn:ss") & vbCrLf
    strMessage = strMessage & "Datenbank:          " & CodeDb.name & vbCrLf
    strMessage = strMessage & "Fehler in:          " & strModule & vbCrLf
    strMessage = strMessage & "Routine:            " & strProc & vbCrLf
    strMessage = strMessage & "Fehlernummer:       " & strErrNumber & vbCrLf
    strMessage = strMessage & "Beschreibung:       " & strErrDescription & vbCrLf
    strMessage = strMessage & "Zeile:              " & strErl & vbCrLf
    strMessage = strMessage & "Bemerkungen:        " & strRemarks & vbCrLf
    DoCmd.OpenForm "frmMessageMail", OpenArgs:=strMessage, WindowMode:=acDialog
End Sub
```

14.2 Fehlerbehandlung mit vbWatchdog

Wenn Sie diesen relativ aufwendigen Weg nicht gehen möchten, können Sie ein paar Euro investieren und sich *vbWatchdog* von Wayne Philips zulegen. Damit erhalten Sie ein COM-Add-In, das Sie jedoch nur zum Ausstatten der Zielanwendung mit der Fehlerbehandlungsfunktionalität

benötigen. Die Beispielanwendung ist bereits damit ausgestattet, dafür fallen für den Benutzer keine Kosten an.

Um *vbWatchdog* zu Ihrer Anwendung hinzuzufügen, wählen Sie nach der Installation des COM-Add-Ins einfach den Menüeintrag *Add-Ins|vbWatchdog|Add vbWatchdog to this project* aus (siehe Abbildung 14.5).

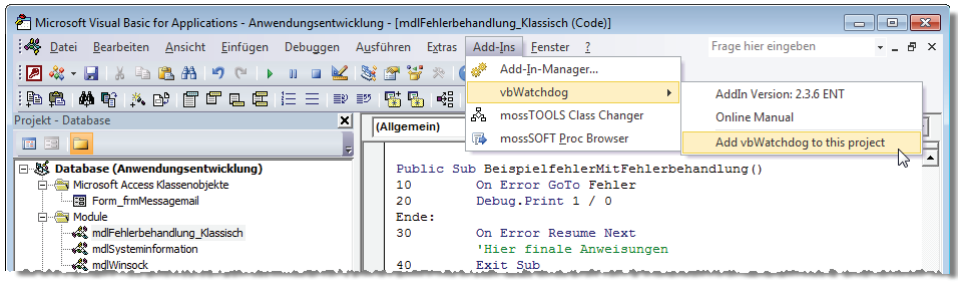


Abbildung 14.5: Hinzufügen von *vbWatchdog* zum aktuellen Projekt

Danach finden Sie im Projekt-Explorer vier neue Klassenmodule vor, welche die komplette Funktion von *vbWatchdog* enthalten (siehe Abbildung 14.6).

Wenn Sie Ihre Anwendung inklusive dieser vier Klassenmodule weitergeben, benötigen Sie keine weiteren Dateien wie beispielsweise eine DLL.

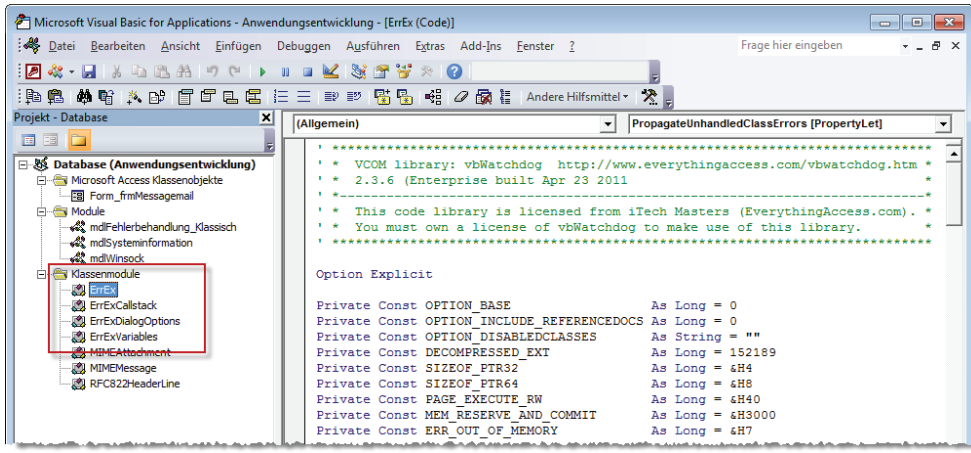


Abbildung 14.6: *vbWatchdog* fügt dem VBA-Projekt vier Klassenmodule hinzu.

Fehlerbehandlung aktivieren

Damit die Fehlerbehandlung funktioniert, müssen Sie diese lediglich zu aktivieren. Dies erledigen Sie im einfachsten Fall mit einem Einzeiler:

```
ErrEx.Enable ""
```

Diese Anweisung bringen Sie beispielsweise in der Ereignisprozedur unter, die durch das Laden des Startformulars *frmStart* ausgelöst wird:

```
Private Sub Form_Load()
    Call ErrEx.Enable("")
End Sub
```

Danach brauchen Sie sich nicht mehr um die Fehlerbehandlung zu kümmern: *vbWatchdog* zeigt beim Auftreten eines jeden Laufzeitfehlers eine entsprechende Meldung an. Setzen Sie beispielsweise einmal die folgende Anweisung im Direktfenster des VBA-Editors ab:

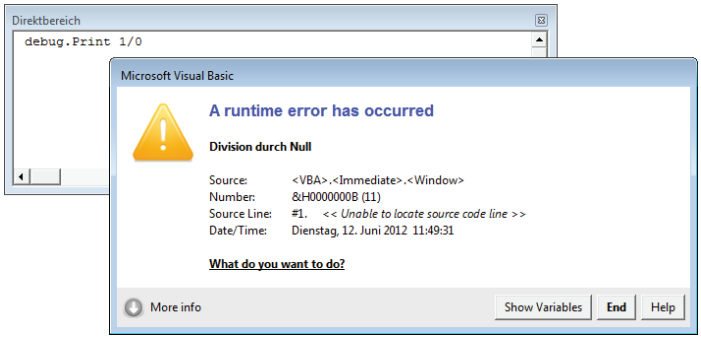


Abbildung 14.7: Eine einfache Fehlermeldung

Mit der folgenden Anweisung deaktivieren Sie *vbWatchdog* (obwohl dies in der Regel nicht wünschenswert ist):

```
ErrEx.Disable
```

14.2.1 Eigene Fehlerbehandlung

Wenn Sie möchten, dass *vbWatchdog* Ihre eigene Fehlerbehandlungsroutine aufruft, geben Sie den Namen dieser Prozedur als Parameter der *Enable*-Methode des *ErrEx*-Objekts an, also beispielsweise so:

```
Private Sub Form_Load()
    Call ErrEx.Enable("")
End Sub
```

Diese Prozedur sieht in einem einfachen Fall so aus:

```
Public Sub ErrNotify_1()
    MsgBox "Error"
End Sub
```

Wenn Sie nun an beliebiger Stelle, also im Code oder auch im Direktfenster, eine fehlerhafte Zeile wie *Debug.Print 1/0* aufrufen, wird zunächst die in Ihrer eigenen Methode angegebene Meldung angezeigt und dann die Meldung von *vbWatchdog*.

vbWatchdog temporär deaktivieren

Wenn Sie eine eigene Fehlerbehandlung einsetzen möchten, um beispielsweise geziehlt auf spezielle Fehler zu reagieren, können Sie *vbWatchdog* genau so umgehen, wie Sie es üblicherweise mit den eingebauten Fehlermeldungen des VBA-Editors tun würden. Dabei gibt es verschiedene Varianten. Bei der ersten aktivieren Sie *vbWatchdog* ohne individuelle Fehlerroutine, wodurch ein Laufzeitfehler die Standardmeldung von *vbWatchdog* anzeigt:

```
ErrEx.Enable ""
Debug.Print 1 / 0
```

Wenn Sie *vbWatchdog* ohne eigene Fehlerroutine aktivieren und dann nach *On Error Resume Next* einen Fehler auslösen, wird der Fehler schlicht übergangen:

```
ErrEx.Enable ""
On Error Resume Next
Debug.Print 1 / 0
```

Wenn Sie *vbWatchdog* mit einer individuellen Fehlermeldung aktivieren, löst der Laufzeitfehler zunächst die benutzerdefinierte Fehlerbehandlung aus und zeigt dann die *vbWatchdog*-Standardmeldung an:

```
ErrEx.Enable "ErrNotify_1"
Debug.Print 1 / 0
```

Schließlich gibt es noch die Möglichkeit, *vbWatchdog* mit einer benutzerdefinierten Fehlerbehandlung zu aktivieren und dann die Fehlerbehandlung mit *On Error Resume Next* außer Kraft zu setzen. In diesem Fall wird die benutzerdefinierte Fehlerbehandlung ausgelöst, aber nicht die Standardmeldung von *vbWatchdog* angezeigt:

```
ErrEx.Enable "ErrNotify_1"
On Error Resume Next
Debug.Print 1 / 0
```

14.2.2 Benutzerdefinierte Fehlermeldung

Das Ziel ist es, mit *vbWatchdog* erstens eine ansprechende und aussagekräftige Fehlermeldung zu generieren und zweitens dem Benutzer die Möglichkeit zu geben, eine E-Mail mit Informationen zum Fehler an den Entwickler der Anwendung zu senden.

Die Fehlermeldung wird im HTML-Format zusammengestellt und verwendet Platzhalter, die zur Laufzeit mit den entsprechenden Fehlerinformationen gefüllt werden. Die Fehlermeldung pas-

sen Sie mit einer Reihe von Eigenschaften an, die das Objekt *DialogOptions* des *ErrEx*-Objekts zur Verfügung stellt. Grundsätzlich beeinflussen Sie mit diesen Methoden das Aussehen der Elemente des Standarddialogs. Wenn Sie beispielsweise alle vorhandenen Schaltflächen entfernen und eine eigene Schaltfläche hinzufügen möchten, welche die fehlerhafte Prozedur beendet, verwenden Sie die folgenden Anweisungen beim Initialisieren des *ErrEx*-Objekts:

```
ErrEx.Enable ""
With ErrEx.DialogOptions
    .RemoveAllButtons
    .AddButton "Prozedur beenden", BUTTONACTION_ONERROREXITPROCEDURE
End With
```

Die vorhandenen HTML-Texte für die einzelnen Bereiche können Sie mit den vier Eigenschaften *HTML_MainBody*, *HTML_MoreInfoBody*, *HTML_CallStackItem* und *HTML_VariableItem* ausgeben. Den HTML-Text für den Haupttext erhalten Sie mit folgender Anweisung, beispielsweise im Direktfenster ausgeführt:

```
Debug.Print ErrEx.DialogOptions.HTML_MainBody
```

Der HTML-Code sieht so aus:

```
<font face=Arial size=13pt color="#4040FF"><b>A runtime error has occurred</b></
font><br><br><b><ERRDESC></b><br><br>Source:|<SOURCEPROJ>.<SOURCEMOD>.<SOURCEPRO
C><br>Number:|&H<ERRNUMBERHEX> (<ERRNUMBER>)<br>Source Line:|#<SOURCELINENUMBER>.
<i><SOURCELINECODE></i><br>Date/Time:|<ERRDATETIME><br><br><b><u>What do you want to
do?</u></b>
```

Interessant hierbei sind die verschiedenen Platzhalter. Diese haben folgende Bedeutung:

- *<ERRDESC>*: Fehlerbeschreibung
- *<ERRNUMBER>*: Fehlernummer
- *<ERRNUMBERHEX>*: Fehlernummer (hexadezimal)
- *<ERRDATETIME>*: Datum und Zeit
- *<SOURCEPROJ>*: Projektname
- *<SOURCEMOD>*: Modulname
- *<SOURCEPROC>*: Prozedurname
- *<SOURCELINENUMBER>*: Zeilennummer
- *<SOURCELINECODE>*: Inhalt der Zeile
- *<CALLSTACK>*: Liste der Prozeduraufrufe

Das Pipe-Zeichen (|) entspricht einem Tabulatorzeichen.

Wenn Sie im einfachsten Fall einfach nur die aktuelle Version des Fehlerdialogs ins Deutsche übersetzen möchten, geben Sie die Werte der Eigenschaften *HTML_MainBody*, *HTML_MoreInfoBody*, *HTML_CallStackItem* und *HTML_VariableItem* aus, ersetzen die englischen Ausdrücke der Ausgabe durch die deutschen Ausdrücke und weisen diese im Code den entsprechenden Eigenschaften wieder zu.

Die englische Originalvariante sieht, von einer VBA-Prozedur aus ausgelöst, wie in Abbildung 14.8 aus.

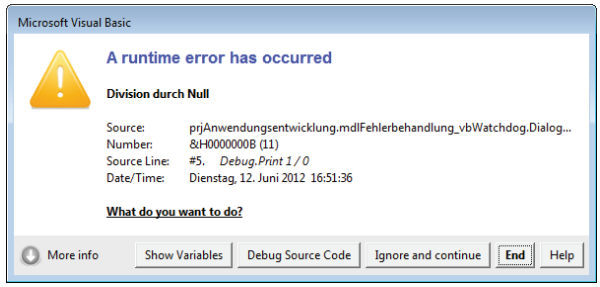


Abbildung 14.8: Englische Originalfehlermeldung

Die folgende Anweisung stellt die Texte der Meldung um. Beachten Sie, dass Sie die im HTML-Code enthaltenen Anführungszeichen beim Zuweisen an die Eigenschaft verdoppeln – anderenfalls würde VBA ein einzelnes Anführungszeichen innerhalb des Ausdrucks als Ende der Zeichenkette interpretieren und der Rest der Zeile würde einen Fehler auslösen.

Die folgende Anweisung teilt die einzelnen Informationen auch noch auf einzelne Zeilen auf, um die Lesbarkeit zu erhöhen:

```
.HTML_MainBody = "<font face=Arial size=13pt color=""#4040FF"">
<b>Es ist ein Laufzeitfehler aufgetreten.</b></font><br><br>
<b><ERRDESC></b><br><br>
Projekt:|<SOURCEPROJ><br>
Modul:|<SOURCEMOD><br>
Prozedur:|<SOURCEPROC><br>
Fehlernummer:|<ERRNUMBER><br>
Zeile:|<SOURCELINENUMBER><br>
Zeileninhalt:|<SOURCELINECODE><br>
Datum und Zeit:|<ERRDATETIME><br><br>
<b><u>Was möchten Sie tun?</u></b>"
```

Dies übersetzt allerdings noch nicht die Texte der Schaltflächen. Deren Beschriftung lässt sich nicht direkt anpassen. Stattdessen entfernen Sie alle Schaltflächen mit der *RemoveAllButtons*-Methode und fügen die Schaltflächen mit benutzerdefinierten Beschriftungen und eingebauten Funktionen wieder hinzu. Das sieht so aus:

15 Bilder

Wenn Sie eine optisch ansprechende Anwendung erstellen wollen, kommen Sie um den Einsatz von Bilddateien und insbesondere von Icons für das Ribbon, Schaltflächen, TreeView et cetera nicht herum. Dieses Kapitel fasst die dazu notwendigen Techniken zusammen.

VBA-Funktionen rund um Bilder

Die Beispieldatenbank enthält zwei Module, die alle für die Arbeit mit Bildern notwendigen Funktionen enthalten. Die Module wurden von Sascha Trowitzsch entwickelt und heißen *mdlOGL0710* sowie *mdlOLE*. Um die in den folgenden Abschnitten beschriebenen Funktionen nutzen zu können, müssen Sie diese beiden Module in Ihre Datenbank importieren.

15.1 Bilder in Access 2007 und 2010

Bilder werden in Access 2007 und Access 2010 etwas unterschiedlich gehandhabt. Wenn Sie beispielsweise einem Access-Formular ein Bild hinzufügen möchten, erledigen Sie das unter Access 2007, indem Sie über den Ribbon-Eintrag *Entwurf|Steuerelemente|Bild* ein Bildsteuerelement zum Formular hinzufügen (siehe Abbildung 15.1).

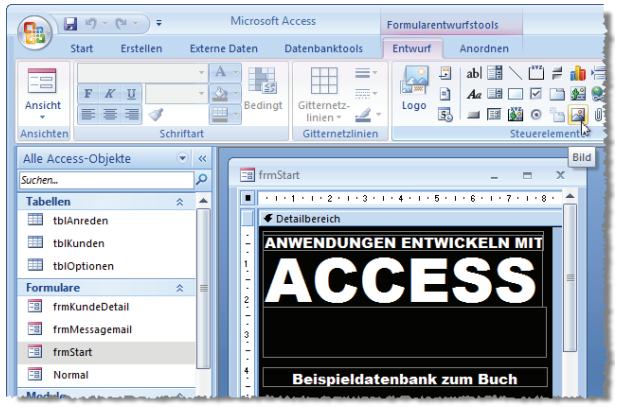


Abbildung 15.1: Hinzufügen eines Bildes unter Access 2007

Access legt dann ein Bildsteuerelement an und bettet die Bilddatei in das Formular ein. Wenn Sie, wie zu Beginn des Buchs beschrieben, die Access-Optionen so eingestellt haben, dass Bilder im Quellbildformat gespeichert werden, nimmt das Bild relativ wenig Platz ein.

Unter Access 2010 fügen Sie einem Formular ein Bild hinzu, indem Sie den Ribbon-Befehl *Entwurf|Steuerelemente|Bild einfügen* betätigen (siehe Abbildung 15.2).

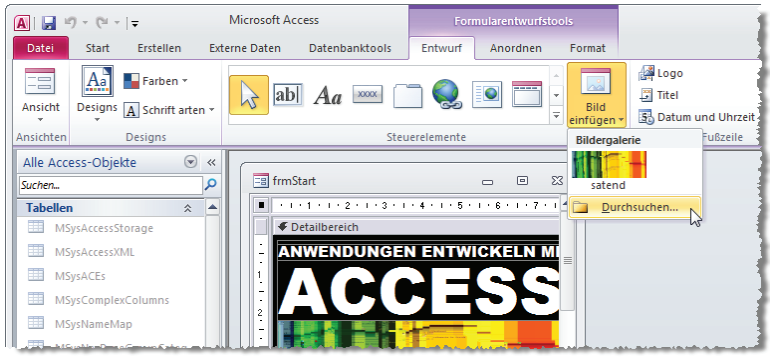


Abbildung 15.2: Hinzufügen eines Bildes unter Access 2010

Standardmäßig speichert Access 2010 die Bilddatei nun in einer Tabelle namens *MSysResources*, die normalerweise nicht sichtbar ist.

Um diese Tabelle sichtbar zu machen, klicken Sie mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs und wählen den Eintrag *Navigationsoptionen* aus dem Kontextmenü aus.

Im nun erscheinenden Dialog aktivieren Sie die beiden Optionen *Ausgeblendete Objekte anzeigen* und *Systemobjekte anzeigen*. Danach erscheinen die entsprechenden Objekte im Navigationsbereich, zum Beispiel die Tabelle *MSysResources* (siehe Abbildung 15.3).

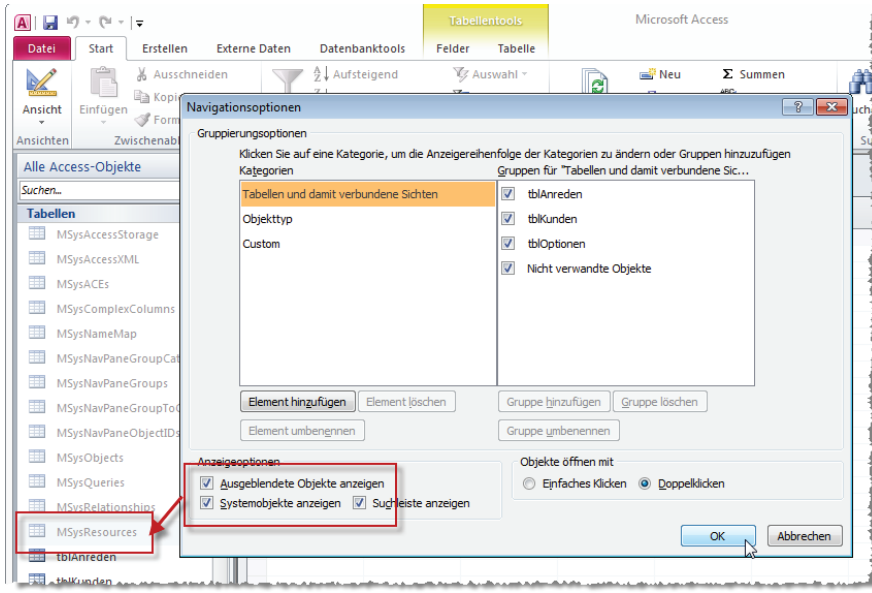


Abbildung 15.3: Aktivieren der Anzeige ausgeblendeter Objekte und von Systemobjekten

Die Anwendung verwendet auf jeden Fall die Tabelle *MSysResources* zum Speichern von Bilddateien, auch unter Access 2007. Es stehen dann zwar einige Funktionen nicht zur Verfügung (zum Beispiel das einfache Zuweisen einer Bilddatei aus *MSysResources* zu einer Schaltfläche), aber das gleichen wir mit zusätzlichen Tools aus.

15.2 Tool zum Hinzufügen von Bildern zu MSysResources

Access 2010 bietet die Möglichkeit, Bilder zur Entwurfsansicht von Formularen oder Berichten hinzuzufügen (siehe Abbildung 15.4).

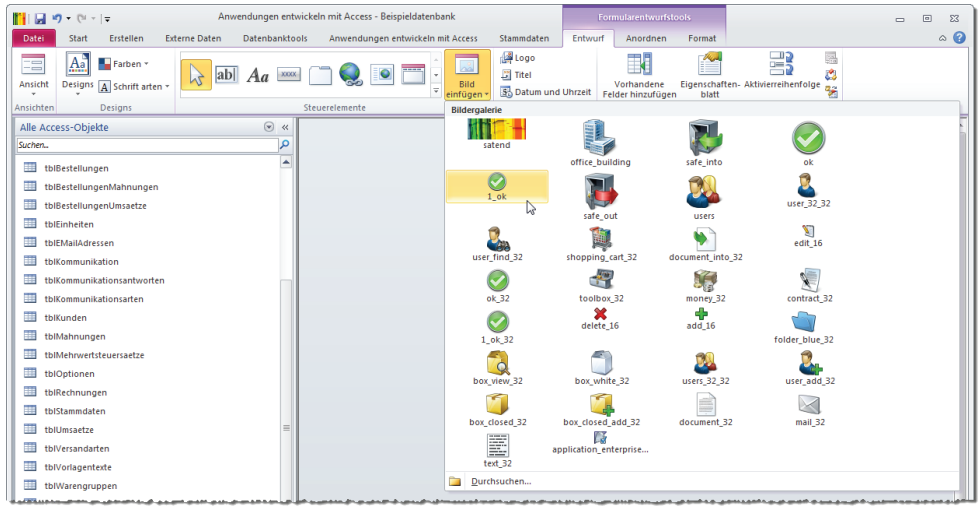


Abbildung 15.4: Auswählen und Hinzufügen von Bildern zur Datenbankanwendung

Die hinzugefügten Bilder werden nicht in die gleichzeitig erzeugten Bildsteuerelemente eingebettet, sondern in der Tabelle *MSysResources* gespeichert und können dann auch anderen Steuerelementen über die Eigenschaft *Bild* zugewiesen werden (siehe Abbildung 15.5).

Dieses Feature ist in Access 2007 gar nicht enthalten. Allerdings ist diese Funktion auch unter Access 2010 nicht besonders komfortabel: Sie können damit beispielsweise immer nur ein Bild gleichzeitig zur Anwendung hinzufügen.

Abhilfe schafft das Access-Add-In *Picture2Attachment*, das ein oder mehrere Bilder zur Tabelle *MSysResources* hinzufügt. Das Tool arbeitet unter Access 2007 und 2010. Da die Tabelle *MSysResources* unter Access 2007 nicht standardmäßig enthalten ist, können Sie diese mit dem Tool per Mausklick anlegen.

Darüber hinaus zeigt es in einer Tabelle alle Datensätze dieser Tabelle an und liefert für Bilddateien auch noch das entsprechende Bild (siehe Abbildung 15.6).

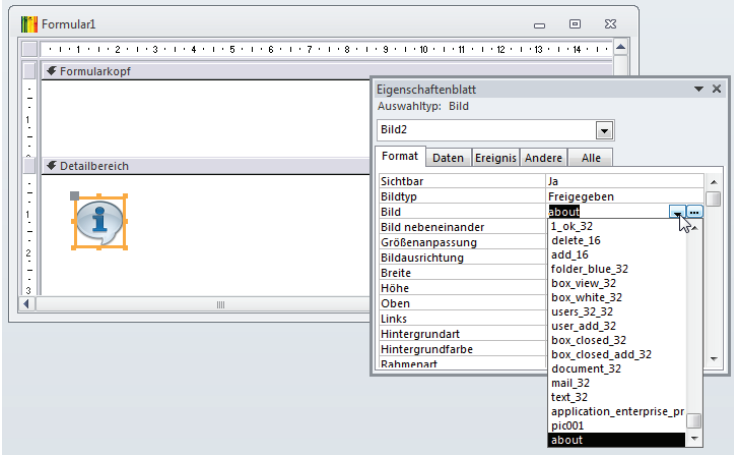


Abbildung 15.5: Bilder werden in einer Tabelle gespeichert und sind dann allgemein zugänglich

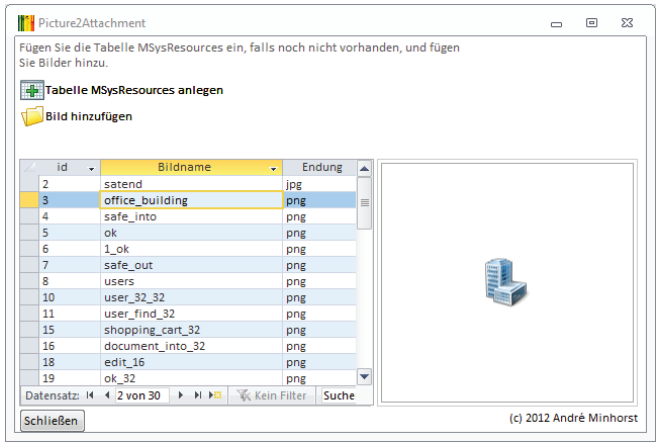


Abbildung 15.6: Tool zum Hinzufügen mehrerer Bilder gleichzeitig – für Access 2007 und 2010

Nun müssen wir nur noch sehen, wie wir die in dieser Tabelle gespeicherten Bilder im Ribbon, in Kontextmenüs, auf Schaltflächen, im Bildsteuerelement und im TreeView-Steuerelement unterbringen.

15.3 Icon-Sammlung

Wenn Sie eine wirklich hübsche Anwendung entwickeln möchten, kommen Sie um den Einsatz von Icons nicht herum. Wie an einigen anderen Stellen in diesem Buch stellt sich die Frage, wie man eine spezielle Aufgabe verrichtet: Soll man selbst viel Arbeit investieren (in diesem Fall in

die Recherche nach kostenlosen Icons im Internet) oder Geld in eine umfangreiche Sammlung fertiger Icons stecken?

Da ich viel lieber programmiere, als stundenlang im Internet zu surfen, habe ich die Entscheidung für mich persönlich bereits vor langer Zeit getroffen: Ich habe mir die Icon-Sammlung von *www.iconexperience.com* gekauft und habe bislang bis auf wenige Ausnahmen immer das passende Icon gefunden.

15.4 Bilder im TreeView-Steuerelement

Wenn Sie Bilder im TreeView-Steuerelement einsetzen möchten, müssen Sie diese zuvor in einem weiteren Steuerelement namens *ImageList* bereitstellen. Dieses Steuerelement weisen Sie dann dem *TreeView*-Steuerelement über eine entsprechende Eigenschaft zu. Beim Anlegen der *TreeView*-Elemente brauchen Sie dann nur noch den Namen des Bildes im *ImageList*-Steuerelement anzugeben.

Das *ImageList*-Steuerelement fügen Sie dem Formular hinzu, das auch das mit Icons auszustattende *TreeView*-Steuerelement enthält. Dazu öffnen Sie das Formular in der Entwurfsansicht, betätigen den Ribbon-Eintrag *Entwurf|Steuerelemente|ActiveX-Steuerelemente* und wählen im nun erscheinenden Dialog den Eintrag *Microsoft ImageList Control, Version 6.0* hinzu (siehe Abbildung 15.7).

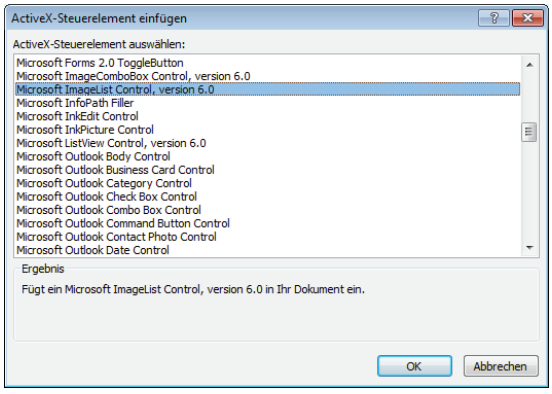


Abbildung 15.7: Hinzufügen eines *ImageList*-Steuerelements

In der Anwendung benötigen wir ein TreeView-Steuerelement und somit auch ein *ImageList*-Steuerelement im Formular *frmKundeDetail*. Während das *TreeView*-Steuerelement gleich in der gewünschten Größe aufgezogen werden kann, gibt sich das *ImageList*-Steuerelement bescheiden und tritt nur in Form eines kleinen Vierecks mit einem Icon in Erscheinung (siehe Abbildung 15.8). In der Formularansicht wird es sogar ausgeblendet – es soll ja auch nur als Container für die im TreeView-Steuerelement anzuzeigenden Icons dienen.

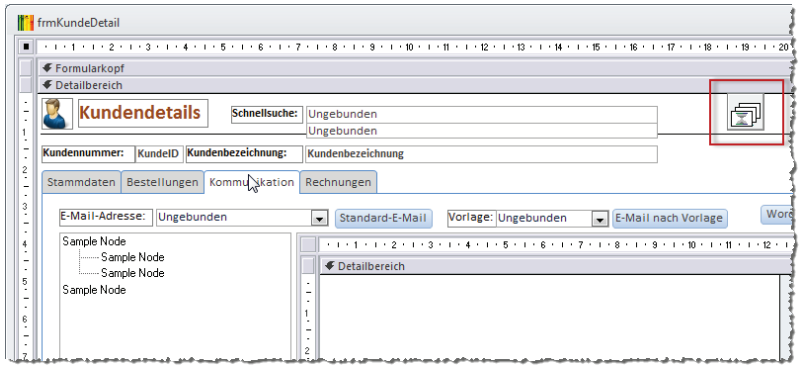


Abbildung 15.8: *ImageList*-Steuerelement im Formular *frmKundeDetail*

Stellen Sie den Namen des *ImageList*-Steuerelements auf *ctlImageList* ein. Normalerweise könnten Sie die Images über den Eigenschaftsdialog dieses Steuerelements hinzufügen – dort stellen Sie die Größe der Icons ein und fügen alle zu verwendenden Bilder hinzu (siehe Abbildung 15.9) – aber in der Praxis war es einfacher, dies per Code zu erledigen und die Bilder direkt aus der Tabelle *MSysResources* einzulesen.

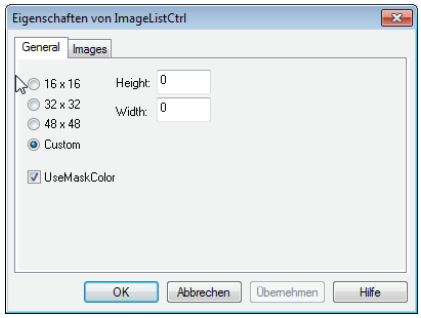


Abbildung 15.9: Eigenschaftsfenster eines *ImageList*-Steuerelements

Um dies gleich beim Öffnen des Formulars zu erledigen, benötigen Sie zunächst eine im Kopf des Klassenmoduls des Formulars deklarierte Variable:

```
Dim objImageList As MSComctlLib.ImageList
```

Diese Variable füllen Sie in einer eigenen Prozedur, die von der Ereignisprozedur *Beim Laden* des Formulars aufgerufen wird – danach folgen weitere Aktionen wie etwa das Füllen des TreeView-Steuerelements, für das wir den ganzen Zauber veranstalten:

```
Private Sub Form_Load()
    ...
    ImageListFuellen
```

```
    TreeViewInstanzieren
End Sub
```

Die Prozedur *ImageListFuellen* setzt zunächst einen Verweis auf das *ImageList*-Steuerelement *ctlImageList*, wobei der Verweis genau genommen auf dessen Eigenschaft *Object* erfolgen muss. Danach leert die Prozedur alle bisher enthaltenen Elemente und stellt die Höhe und Breite für die Elemente auf jeweils 16 Pixel ein – wir wollen ja nur Icons für die TreeView-Elemente verwenden. Schließlich ruft die Prozedur zum Hinzufügen eines jeden Icons eine weitere Prozedur namens *ImageHinzufuegen* auf, welche die eigentliche Arbeit übernimmt.

Dabei übergibt sie einen Verweis auf das *ImageList*-Steuerelement sowie den Namen, unter dem die Bilddatei in der Tabelle *MSysResources* zu finden ist:

```
Private Sub ImageListFuellen()
    Set objImageList = Me.ctlImageList.Object
    objImageList.ListImages.Clear
    objImageList.ImageHeight = 16
    objImageList.ImageWidth = 16
    ImageHinzufuegen objImageList, "mail_16"
    ImageHinzufuegen objImageList, "folder3_mail_16"
    '...
End Sub
```

Die Prozedur *ImageHinzufuegen* deklariert ein neues Objekt des Typs *StdPicture* und ermittelt mit der Funktion *BildIDErmitteln* die ID des Datensatzes der Tabelle *MSysResources*, der im Feld *Name* die entsprechende Bezeichnung aufweist. Die Funktion *GetPicture* liest wiederum das im *Attachment*-Feld dieser Tabelle gespeicherte Bild in das Objekt *objPicture* ein. Die letzte Anweisung schließlich fügt das *StdPicture*-Objekt zum *ImageList*-Steuerelement hinzu:

```
Public Sub ImageHinzufuegen(objImageList As MSComctlLib.ImageList, strImage As String)
    Dim objPicture As StdPicture
    Dim lngID As Long
    lngID = BildIDErmitteln(strImage)
    Set objPicture = GetPicture(lngID)
    objImageList.ListImages.Add , strImage, objPicture
End Sub
```

Die Funktion *BildIDErmitteln* verwendet eine einfache *DLookup*-Anweisung zur Ermittlung der ID des Datensatzes mit dem angegebenen Bildnamen:

```
Public Function BildIDErmitteln(strImage As String) As Long
    Dim lngID As Long
    lngID = DLookup("Id", "MSysResources", "Name='" & strImage & "'")
    BildIDErmitteln = lngID
End Function
```

Fehlt noch die Funktion *GetPicture*. Damit werden zwei verschachtelte Funktionen aufgerufen. Die innere heißt *BLOB2Binary0710* und liest den Inhalt des *Attachment*-Feldes der Tabelle *MSysResources* für die gewünschte Bilddatei aus. Das Ergebnis ist ein Byte-Array, das gleich von der zweiten Funktion *ArrayToPicture* weiterverarbeitet wird.

Diese Funktion wandelt das Byte-Array in ein Objekt des Typs *StdPicture* um und referenziert dieses mit der Objektvariablen *objPicture*.

```
Public Function GetPicture(lngID As Long) As StdPicture
    Dim objPicture As StdPicture
    Set objPicture = ArrayToPicture(mdlBLOBs0710.BLOB2Binary0710(CurrentDb,
"MSysResources", "Data", "Id", lngID, True), &HFFFFFFFF)
    Set GetPicture = objPicture
End Function
```

Wichtig ist an dieser Stelle, dass Sie der Funktion *ArrayToPicture* als zweiten Parameter den Wert *&hFFFFFFFF* übergeben, damit der transparente Hintergrund nicht schwarz angezeigt wird, sondern weiß. Wenn Sie den Namen beziehungsweise den *Key*-Wert eines der so im *ImageList*-Steuerelement gespeicherten Bilder beim Anlegen eines *Node*-Objekts etwa im TreeView angeben, wird das Icon wie in Abbildung 15.10 angezeigt:

```
Private Sub TreeViewInstanzieren()
    '...
    With objTreeView
        '...
        Set .ImageList = Me.ctlImageList.Object
        Set objNode = .Nodes.Add(, , "a0", "E-Mails", "folder3_mail_16")
    End With
End Sub
```

15.5 Bilder in Kontextmenüs

Das Anlegen eines Kontextmenüs per Code sieht beispielsweise wie folgt aus:

```
Set cbr = CommandBars.Add("cbrKommunikation", msoBarPopup, , True)
...
Set cbb = cbr.Controls.Add(msoControlButton)
cbb.Caption = "E-Mail beantworten"
cbb.onAction = "=AntwortErstellen(" & lngKommunikationID & ")"
cbb.Picture = GetPictureByName("mail_forward_16")
```

Dabei erstellt die erste Anweisung das Kontextmenü selbst, die zweite fügt eine Schaltfläche hinzu. Die dritte und vierte Anweisung legen die Beschriftung und die beim Betätigen des Steuerelements auszuführende VBA-Funktion fest.

16 Ribbons

Wenn Sie mal eben eine Anwendung für den Privatgebrauch oder für Kollegen erstellen, die sich mit Access auskennen, werden Sie ein paar Tabellen, Abfragen, Formulare und Berichte lieblos dahinprogrammieren und sich vermutlich wenig um Ergonomie scheren.

Sobald jedoch Access-unerfahrene Mitarbeiter oder Kunden hinzukommen, sollten Sie Ihrer Anwendung deutlich ersichtliche Möglichkeiten hinzufügen, die einzelnen Elemente der Benutzeroberfläche und die Funktionen der Anwendung aufzurufen.

Dabei gilt: Der Navigationsbereich ist kein guter Ersatz für eine Menüleiste beziehungsweise ein Ribbon und sinnvoll eingesetzte Kontextmenüs! Sie selbst wissen zwar, hinter welchem Formularnamen sich welches Formular verbirgt und rufen VBA-Prozeduren schnell durch Öffnen des VBA-Editors und Anwählen der gesuchten Routine auf, aber dem Benutzer einer professionellen Anwendung hilft dies nicht weiter.

Früher (also vor Access 2007) konnten Sie Menüleisten, Symbolleisten und Kontextmenüs über entsprechende Elemente der Benutzeroberfläche erstellen. Heute sieht das anders aus: Die entsprechenden Tools sind nicht mehr in Access verfügbar, zumindest nicht für Datenbanken im Format von Access 2007/2010.

Das Ribbon, so der englische Name für das, was Microsoft in Access 2007 liebevoll mit Multifunktionsleiste und Access 2010 mit Menüband übersetzt hat (wir verwenden im Folgenden den Begriff *Ribbon*), lässt sich nur per Definition über ein XML-Dokument erstellen. Dann wären da noch die Kontextmenüs, die wir aber der Einfachheit halber direkt mit VBA erstellen – das ist generell eine gute Idee, weil wir in der Anwendung auch mal Kontextmenüs benötigen, deren Befehle von den jeweils angelegten Daten abhängen und die somit zur Laufzeit erstellt werden müssen. Beispiele für Kontextmenüs finden Sie in den jeweiligen Kapiteln.

16.1 Menüführung per Ribbon

Vor dem Erstellen des Ribbons stellt sich die Frage, wie die Menüführung überhaupt aussehen soll. Grundsätzlich gibt es folgende Möglichkeiten:

» Sie können ein Anwendungsribbon definieren, das direkt beim Start der Anwendung angezeigt wird. Damit können Sie direkt Formulare und Berichte oder auch VBA-Prozeduren aufrufen.

» Sie können für jedes Formular ein oder mehrere Ribbons definieren, die mit dem Formular angezeigt werden. Damit können Sie im Ribbon Befehle einblenden, die nur für das jeweilige Formular gebraucht werden. Sollten Sie beispielsweise ein Datenblatt als Kundenübersicht verwenden und dieses soll das komplette Access-Fenster ausfüllen, können Sie im Ribbon

Schaltflächen etwa zum Anlegen neuer oder zum Bearbeiten und Löschen vorhandener Datensätze anzeigen.

Ob Sie die Steuerelemente eines Formulars in ein mit dem Formular angezeigtes Ribbon auslagern möchten, hängt von der jeweiligen Konstellation ab. In der Regel sollte dies jedoch nur geschehen, wenn es keine andere Möglichkeit gibt – die Steuerung des Ribbons ist nämlich nicht gerade trivial und lässt sich außerdem äußerst schlecht debuggen.

16.2 Das Ribbon der Beispielanwendung

Die Beispielanwendung verwendet ein Hauptribbon, von dem aus Sie alle wichtigen Anwendungsfunktionen ansteuern können. Dieses sieht wie in Abbildung 16.1 aus.

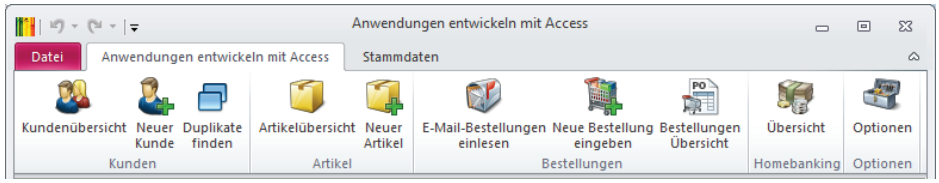


Abbildung 16.1: Das Hauptribbon der Beispielanwendung

Dieses Ribbon besitzt noch eine zweite Registerseite, über die Sie die verschiedenen Dialoge zum Bearbeiten der Vorlagen für verschiedene E-Mails- und Word-Dokumente aufrufen können (siehe Abbildung 16.2).

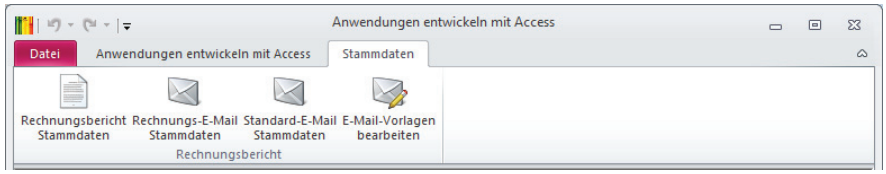


Abbildung 16.2: Ribbon-Tab zum Aufrufen der Stammdaten

16.3 Ribbons von Hand erstellen

Ein solches Ribbon wie oben abgebildet zu erstellen, ohne ein Tool zu verwenden, erfordert die folgenden Schritte:

» Erstellen einer Tabelle namens *USysRibbons*

» Anlegen eines Datensatzes mit einem XML-Dokument, das die gewünschte Ribbon-Definition enthält, also das Aussehen des Ribbons

» Hinzufügen der in dieser Definition referenzierten Bilddateien zur Tabelle *MSysResources*

» Anlegen der Callback-Funktionen, die durch die Steuerelemente ausgelöst werden sollen

» Einstellen der Eigenschaft *Name* des Menübands in den Access-Optionen auf den Wert, der in der Tabelle *USysRibbons* im Feld *Name* angegeben wurde

Diese Schritte schauen wir uns auf den folgenden Seiten im Detail an.

16.3.1 Tabelle zum Speichern der Ribbon-Definition erstellen

Wie ein Ribbon aussieht, legen Sie mit einer entsprechenden Beschreibung im XML-Format fest – wie dies genau funktioniert, schauen wir uns gleich an. Diese Beschreibung können Sie je nach Office-Anwendung an verschiedenen Stellen speichern – unter Access ist der optimale Platz eine Tabelle namens *USysRibbons*.

Warum das? Weil Access beim Starten die Einträge dieser Tabelle durchforstet und die Namen der enthaltenen Ribbons an verschiedenen Stellen zur Auswahl anbietet. Legen wir also zunächst die benötigte Tabelle an.

Diese sieht wie in Abbildung 16.3 aus und enthält drei Felder: eines mit einem eindeutigen Schlüssel, eines mit der Bezeichnung der Ribbon-Definition und eines mit der Ribbon-Definition selbst.

Die Bezeichung dient später zur Auswahl der jeweiligen Definition. Da eine Ribbon-Definition durchaus einige hundert oder gar tausend Zeichen enthalten kann, wird das Feld *RibbonXML* als Memofeld ausgelegt.

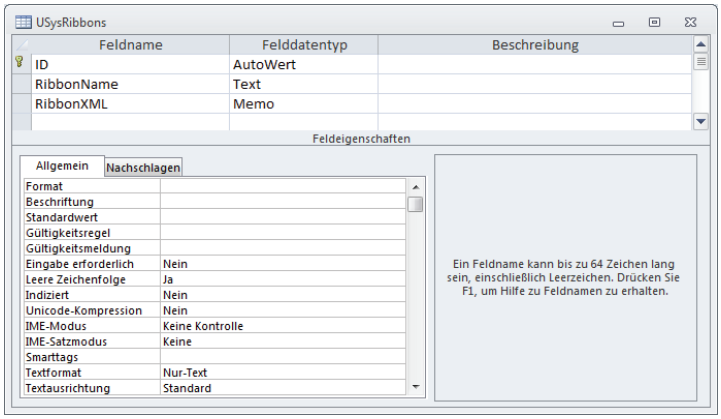


Abbildung 16.3: Die Tabelle *USysRibbons*

In der Datenblattansicht sieht die Tabelle wie in Abbildung 16.4 aus. Dort erkennen Sie den Text einer Ribbon-Definition.

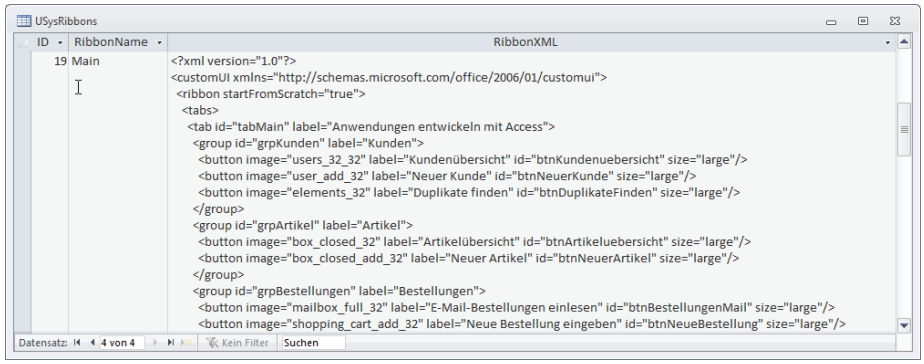


Abbildung 16.4: Die Tabelle *USysRibbons* in der Datenblattansicht

16.3.2 XML-Dokument mit der Ribbon-Definition zusammenstellen

Das Zusammenstellen der XML-Definition eines Ribbons ist grundsätzlich Fleißarbeit, was bedeutet, dass dabei auch gern mal Flüchtigkeitsfehler auftauchen – aber einfache Ribbons lassen sich mit Copy und Paste und einigen manuellen Korrekturen durchaus zusammenstellen. Schauen wir uns zunächst an, wie die Definition der Ribbons für unsere Anwendung für Access 2007 und 2010 im Überblick aussieht – anschließend analysieren wir die einzelnen Elemente.

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabMain" label="Anwendungen entwickeln mit Access">
        <group id="grpKunden" label="Kunden" image="users_32_32">
          <button image="users_32_32" label="Kundenübersicht" id="btnKundenuebersicht"
              onAction="onAction" size="large"/>
          <button image="user_add_32" label="Neuer Kunde" id="btnNeuerKunde"
              onAction="onAction" size="large"/>
          <button image="elements_32" label="Duplikate finden" id="btnDuplikateFinden"
              onAction="onAction" size="large"/>
        </group>
        <group id="grpArtikel" label="Artikel" image="box_closed_32">
          <button image="box_closed_32" label="Artikelübersicht"
              id="btnArtikeluebersicht" onAction="onAction" size="large"/>
          <button image="box_closed_add_32" label="Neuer Artikel" id="btnNeuerArtikel"
              onAction="onAction" size="large"/>
        </group>
        <group id="grpBestellungen" label="Bestellungen" image="purchase_order_cart_32">
```

```
        <button image="mailbox_full_32" label="E-Mail-Bestellungen einlesen"
            id="btnBestellungenMail" onAction="onAction" size="large"/>
        <button image="shopping_cart_add_32" label="Neue Bestellung eingeben"
            id="btnNeueBestellung" onAction="onAction" size="large"/>
        <button image="purchase_order_cart_32" label="Bestellungen Übersicht"
            id="btnBestellungenUebersicht" onAction="onAction" size="large"/>
      </group>
      <group id="grpHomebanking" label="Homebanking">
        <button image="money_32" label="Übersicht" id="btnHomebanking"
            onAction="onAction" size="large"/>
      </group>
      <group id="grpOptionen" label="Optionen">
        <button image="toolbox_32" label="Optionen" id="btnOptionen"
            onAction="onAction" size="large"/>
      </group>
    </tab>
    <tab id="tabStammdaten" label="Stammdaten">
      <group id="grpRechnungsbericht" label="Rechnungsbericht">
        <button image="document_32" label="Rechnungsbericht Stammdaten"
            id="btnRechnungsberichtStammdaten" onAction="onAction" size="large"/>
        <button label="Rechnungs-E-Mail Stammdaten" onAction="onAction" size="large"
            id="btnRechnungsEMailStammdaten" image="mail_32"/>
        <button image="mail_32" label="Standard-E-Mail Stammdaten"
            id="btnStandardEMailStammdaten" onAction="onAction" size="large"/>
        <button image="mail_32_edit" label="E-Mail-Vorlagen bearbeiten"
            id="btnEMailVorlagen" onAction="onAction" size="large"/>
      </group>
    </tab>
  </tabs>
 </ribbon>
</customUI>
```

16.3.3 Ribbon-Definition im Detail

Die Ribbon-Definition wird durch das XML-Tag eingeleitet:

```
<?xml version="1.0"?>
```

Danach folgt, je nach Access-Version, eine der folgenden beiden Zeilen – zunächst die Variante für Access 2007.

In beiden Fällen wird zunächst ein Attribut namens *xmlns* angegeben, gefolgt vom Attribut *loadImage*, dessen Bedeutung wir weiter unten klären:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
loadImage="loadImage">
```

Unter Access 2010 lautet diese Zeile so:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage>
```

Wenn Sie die Anwendung mit dieser Zeile unter Access 2007 öffnen, erscheint der Fehler aus Abbildung 16.5 (gefolgt von einigen weiteren Fehlermeldungen).

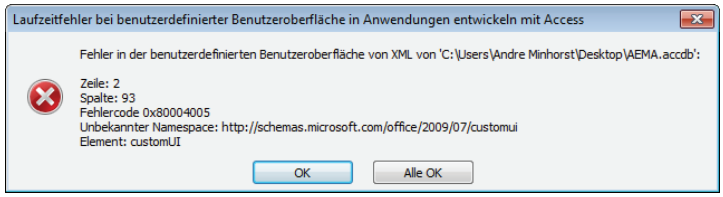


Abbildung 16.5: Fehler beim Öffnen einer Anwendung unter Access 2007 mit einen Ribbon für Access 2010

Andersherum gibt es keinen Fehler – Access 2010 öffnet auch problemlos Ribbons für Access 2007. Das Attribut *xmlns* gibt dabei einen sogenannten Namespace an, der nicht nur festlegt, für welche Access-Versionen die Ribbon-Definition funktioniert, sondern auch, welche Elemente in der XML-Datei verwendet werden dürfen.

Wenn Sie also beispielsweise *http://schemas.microsoft.com/office/2006/01/customui* angeben, was für die 2007er-Version steht, dürfen Sie im Ribbon auch keine Elemente verwenden, die erst mit Access 2010 eingeführt wurden – also beispielsweise solche Elemente, die Anpassungen des Backstage-Bereichs beschreiben.

Andersherum gilt das Gleiche: Wenn Sie für das Attribut *xmlns* den Wert *http://schemas.microsoft.com/office/2009/07/customui* angeben, darf die Ribbon-Definition keine Elemente erhalten, die ausschließlich unter Access 2007 erlaubt waren (dies betrifft insbesondere das *officeMenu*-Element, mit dem Sie das Office-Menü anpassen konnten).

Mehr Informationen zum Backstage-Bereich und zum Office-Menü und wie wir es im Rahmen dieser Anwendung anpassen wollen, erhalten Sie weiter unten unter »Eingebaute officeMenu- und backstage-Elemente ausblenden« ab Seite 503.

Schauen wir uns zunächst den Rest der Ribbon-Definition für unser Ribbon an. Das folgende Element legt fest, dass außer den durch die aktuelle Ribbon-Definition hinzugefügten Elementen alle weiteren Elemente ausgeblendet werden sollen:

```
<ribbon startFromScratch="true">
```

Weiter oben unter »Anwendung für Entwicklung und Test starten« ab Seite 42 haben Sie erfahren, wie Sie die Anwendung mit einer Starter-Datenbank mit verschiedenen Konfigurationen

starten. Dort können Sie unter anderem angeben, welches Ribbon beim Start der Anwendung angezeigt werden soll. Ein Ribbon für den Entwicklungsmodus sollte alle für den Betrieb der Anwendung erforderlichen Elemente liefern, aber die für die Entwicklung benötigten Werkzeuge nicht ausblenden. Deshalb würde eine Ribbon-Definition für den Entwicklungsmodus genau so aussehen wie eines für den Produktivbetrieb – mit einem klitzekleinen Unterschied. Das Attribut *startFromScratch* wird hier auf den Wert *false* eingestellt:

```
<ribbon startFromScratch="false">
```

Auf diese Weise zeigt das Ribbon zwar alle neu hinzugefügten Elemente an, blendet aber die eingebauten Elemente nicht aus, sodass Sie wie gewohnt mit der Entwicklungsumgebung weiterarbeiten können (siehe Abbildung 16.6).

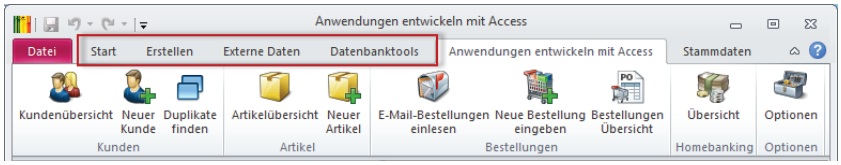


Abbildung 16.6: Ribbon im Entwicklermodus: Alle neuen Elemente sind sichtbar, aber auch die Entwicklungswerkzeuge sind noch verfügbar.

Nun folgen die eigentlichen Elemente, die in genau der gleichen hierarchischen Ordnung definiert werden, wie sie später im Ribbon erscheinen. Zunächst das *tabs*-Element, das alle enthaltenen *tab*-Elemente zusammenfasst:

```
<tabs>
```

Danach folgen in der Hierachie die einzelnen *tab*-Elemente, zunächst das mit der Beschriftung *Anwendungen entwickeln mit Access*:

```
<tab id="tabMain" label="Anwendungen entwickeln mit Access">
```

Hier ist wichtig, dass jedes benutzerdefinierte *tab*-Element mit einem eindeutigen Wert für das Attribut *id* und mit einer Beschriftung ausgestattet wird (für das Attribut *label*). Alle Elemente in der Beispielanwendung erhalten IDs mit entsprechenden Präfixen, also beispielsweise *tab* für das *tab*-Element, *grp* für das *group*-Element oder *btn* für das *button*-Element – also jeweils Abkürzungen mit drei Buchstaben.

Unterhalb eines *tab*-Elements finden sich ein oder mehrere *group*-Elemente. Das erste heißt *grpKunden* und zeigt die Beschriftung *Kunden* an:

```
<group id="grpKunden" label="Kunden">
```

Das ist die Version für Access 2007. Unter Access 2010 gibt es ein neues Element namens image. Dieses nimmt den Namen einer Bilddatei entgegen. Wo aber sollte ein *group*-Element ein Bild anzeigen?

```
<group id="grpKunden" label="Kunden" image="users_32_32">
```

Ganz einfach: Dieses erscheint, wenn das Fenster so schmal wird, dass einzelne Gruppen minimiert werden müssen. Wenn Sie dieses Attribut für alle drei Gruppen mit mehr als einem Steuerelement definieren, sieht das Ribbon wie in Abbildung 16.7 aus.

Abbildung 16.7: Dieses Ribbon zeigt Platzhalter-Bilder für minimierte *group*-Elemente an.

Zu diesem Zeitpunkt liegt noch kein Mechanismus vor, der dafür sorgt, dass die für das *image*-Attribut vorliegenden Bilder tatsächlich angezeigt werden – darum kümmern wir uns weiter unten.

Nun folgen die drei Schaltflächen, definiert durch entsprechende *button*-Elemente:

```
<button image="users_32_32" label="Kundenübersicht" id="btnKundenuebersicht"
    onAction="onAction" size="large"/>
<button image="user_add_32" label="Neuer Kunde" id="btnNeuerKunde"
    onAction="onAction" size="large"/>
<button image="elements_32" label="Duplikate finden" id="btnDuplikateFinden"
    onAction="onAction" size="large"/>
```

Alle drei enthalten einen Wert für das anzuzeigende Bild (*image*), eine Beschriftung (*label*), einen eindeutigen Bezeichner (*id*) und das Attribut *size*, das die Größe der Schaltfläche festlegt.

Außerdem besitzt jedes Element das Attribut *onAction*, das angibt, welche VBA-Funktion beim Anklicken der Schaltlfäche ausgelöst werden soll.

Aber warum lösen alle die gleiche Funktion aus? Ganz einfach: Weil die Kopfzeile der angegebenen Prozedur eine ganz spezielle Syntax haben muss, die einen Parameter entgegennimmt. Die Prozedur *onAction* sieht beispielsweise so aus:

```
Sub onAction(Control As IRibbonControl)


End Sub
```

Der Parameter Control hat den Typ *IRibbonControl*, der wiederum in der Bibliothek *Microsoft Office x.0 Object Library* definiert wird. Um beispielsweise IntelliSense bei der Programmierung der Ribbon-Elemente zu nutzen, benötigen Sie daher einen Verweis auf die entsprechende Bibliothek, die Sie im *Verweise*-Dialog (Menüeintrag *Extras*|*Verweise* im VBA-Editor) hinzufügen (siehe Abbildung 16.8).

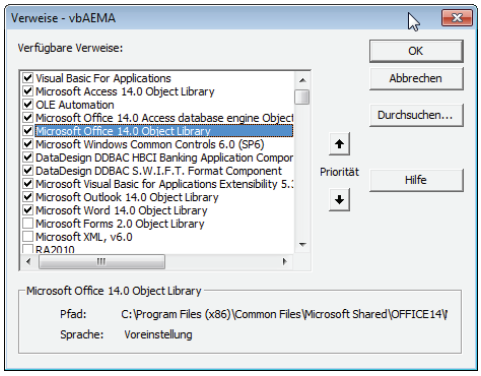


Abbildung 16.8: Für die Programmierung des Ribbons benötigen Sie einen Verweis auf die Bibliothek *Microsoft Office x.0 Object Library*.

Die komplette Prozedur *onAction* schauen wir uns weiter unten an – zunächst betrachten wir die übrigen Elemente der Ribbon-Definition. Dort wird die erste Gruppe mit einem schließenden Element abgeschlossen:

```
        </group>
```

Danach folgen drei weitere *group*-Elemente mit einigen *button*-Elementen, die aber keine neuen Steuerelemente oder Attribute mehr liefern; auch das zweite *tab*-Element begnügt sich mit der Darstellung einer Gruppe mit vier weiteren Steuerelementen. Das Ribbon wird schließlich durch einige schließende Elemente vervollständigt:

```
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

16.3.4 Bilddateien zur Datenbank und zum Ribbon hinzufügen

Bevor wir uns an die eigentliche Funktion der Ribbon-Schaltflächen wagen, schauen wir uns an, wie die Bilder ins Ribbon gelangen.

Wenn Sie Bilder im Ribbon anzeigen möchten, sind zwei Schritte nötig:

» Sie geben einen Ausdruck, typischerweise den Namen des Bildes, als Wert des Attributs *image* des betroffenen Steuerelements an.

» Sie legen eine Callback-Funktion an, die dafür sorgt, dass das Ribbon das Bild beim Anzeigen des betroffenen Steuerelements einliest. Diese Callback-Funktion können Sie wiederum für zwei Callback-Attribute hinterlegen: für das *loadImage*-Attribut des *customUI*-Elements oder für die Eigenschaft *getImage* des betroffenen Steuerelements. Die letztere Methode

kommt vor allem dann zum Einsatz, wenn Sie zur Laufzeit die Bilder einzelner Ribbon-Steuerelemente anpassen möchten – dies kommt in der Beispielanwendung jedoch nicht vor.

Konzentrieren wir uns also auf den ersten Fall. Das *button*-Element *btnKundenuebersicht* etwa soll das Bild *users_32_32* aus der Tabelle *MSysResources* anzeigen.

Dazu stellen Sie das Attribut *image* auf diesen Wert ein. Außerdem weisen Sie dem Attribut *loadImage* den Namen der VBA-Funktion zu, welche das Laden des Bildes übernehmen soll – in diesem Fall *loadImage*:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabMain" label="Anwendungen entwickeln mit Access">
        <group id="grpKunden" label="Kunden">
          <button image="users_32_32" label="Kundenübersicht" id="btnKundenuebersicht"
              size="large"/>
        ....
        </group>
        ...
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Das ist alles, was wir in diesem Kapitel zu diesem Thema erläutern – mehr erfahren Sie unter »Bilder im Ribbon« (Seite 485).

16.3.5 Notwendige Eigenschaften einstellen

Die erste wichtige Eigenschaft sorgt dafür, dass das gewünschte Ribbon beim Start der Anwendung angezeigt wird – und auch immer dann, wenn nicht gerade ein Formular oder ein Bericht dafür sorgt, dass ein anderes Ribbon erscheint.

Sie finden diese Eigenschaft, die unter Access 2007 noch *Name der Multifunktionsleiste* heißt (siehe Abbildung 16.9), unter Access 2010 unter dem Namen *Name des Menübandes* in den Access-Optionen (siehe Abbildung 16.10).

Schließlich sollten Sie noch die Einstellung *Fehler des Benutzeroberflächen-Add-Ins anzeigen* aktivieren (siehe Abbildung 16.11). Nur so werden Fehler in der Ribbon-Definition beim Laden des Ribbons angezeigt.

Und nicht zu vergessen: Wie oben erwähnt, benötigen Sie dringend einen Verweis auf die Bibliothek *Microsoft Office x.0 Object Library*.

17 Datenbank aufteilen

Warum teilt man eine Datenbank auf und was bedeutet dies? Eine Access-Datenbankdatei ist ein Monolith. Sie enthält alle notwendigen Objekte wie Tabellen, Abfragen, Formulare, Berichte, Makros und VBA-Module. Mit den Tabellen enthält sie sogar noch die Daten, die der Benutzer eingibt. Und das ist der entscheidende Nachteil: Da diese eine Datei die Daten enthält, ist sie auch ständigen Änderungen durch den Benutzer unterworfen. Das ist kein Problem, wenn die Datenbank fertig programmiert ist. Das ist selten der Fall: Die meisten Datenbanken offenbaren früher oder später kleinere Fehler, die es zu beheben gilt, oder die Benutzer wünschen Erweiterungen oder Änderungen der Anwendung. Und hier wird es schwierig: Sie können zwar die Änderungen vornehmen, aber während Sie programmieren, gibt der Benutzer natürlich weiter Daten ein. Also teilen Sie die Datenbank in zwei Datenbanken auf: eine, die die Daten enthält, also die Tabellen (das Backend), und eine mit den übrigen Elementen, also der Benutzeroberfläche und der Anwendungslogik in Form von VBA-Code (das Frontend). Es gibt natürlich auch Anwendungslogik in den Tabellen – zum Beispiel Restriktionen wie eindeutige Felder.

Das Aufteilen der Datenbank ist kein Problem, denn Sie können ja in einer Access-Datenbank ohne eigene Tabellen Verknüpfungen zu den Tabellen einer anderen Access-Datenbank herstellen und diese Tabellen genauso nutzen, als ob sich diese in der gleichen Access-Datenbank befinden würden. Welche Vorteile bringt dies nun? Sie erhalten damit die Möglichkeit, das Frontend, das ja nun nur noch Elemente enthält, die der Benutzer nicht verändert, weiterzuentwickeln. Zu gegebener Zeit tauschen Sie dann einfach das alte Frontend gegen das neue Frontend aus und der Benutzer kann von den neuen Funktionen und/oder behobenen Fehlern profitieren.

Natürlich kann es auch einmal Aktualisierungen am Datenmodell geben. Dafür müssen Sie den Benutzer aber auch nicht von der Arbeit abhalten und beispielsweise das Backend mit dem aktuellen Stand der Daten anfordern, die Änderungen durchführen und das Backend wieder zum Benutzer zurücksenden. Sie können nämlich im Frontend Code unterbringen, der beim Start des Frontends die Version des Backends prüft und notwendige Änderungen am Datenmodell auf das Backend anwendet.

Es gibt neben der Wartung der Datenbank natürlich noch einen weiteren, sehr wichtigen Grund für die Aufteilung einer Datenbank: Sie können dann nämlich Frontend-Datenbanken auf verschiedenen Rechnern installieren, damit mehrere Benutzer gleichzeitig auf die im Backend gespeicherten Daten zugreifen können. Dies ist ein Szenario, das üblicherweise mit der Verwaltung eben jener Benutzer einhergeht und zusätzlich nach der Vergabe unterschiedlicher Berechtigungen für Benutzer und/oder Benutzergruppen verlangt.

In diesem Buch werden wir uns allerdings ausschließlich dem Aufteilen einer Datenbank zum Zwecke der einfacheren Wartung widmen. Mit Access 2007 hat Microsoft nämlich entschieden, auf ein integriertes Sicherheitssystem inklusive Benutzerverwaltung zu verzichten. Das ist nicht allzu schlimm, denn das Sicherheitssystem war ohnehin nie wirklich sicher. Ich werde aber

in nicht allzu ferner Zukunft in einer weiteren Publikation auf die Anwendung einer Access-Datenbank in Zusammenarbeit mit einem Datenbank-Management-System wie dem Microsoft SQL Server oder MySQL eingehen. Diese Systeme bringen eine professionelle Benutzer- und Benutzergruppenverwaltung mit, die auch die entsprechende Sicherheit garantiert.

Die eigentliche Aufteilung einer Datenbank in Frontend und Backend ist ein harmloser Vorgang, der sich allein mit den Mitteln der Benutzeroberfläche von Access durchführen lässt. Interessant wird es hingegen, wenn das Frontend ausgetauscht wird. Warum? Weil beim Verknüpfen einer Frontend-Datenbank mit dem Backend der Speicherort des Backends in einer Systemtabelle des Frontends gespeichert wird – zum Beispiel *c:\Datenbank\Backend\AEMA_BE.accda*.

Wenn Sie nun eine neue Version des Frontends programmieren und diese über die alte Version kopieren, weiß die neue Version natürlich noch nicht, an welchem Ort sich die zu verknüpfenden Tabellen im Backend befinden. Sie versucht dann, die Tabellen von dem Ort einzubinden, an dem diese sich zuletzt befunden haben, also irgendwo im Dateisystem Ihres Entwicklungsrechners. Wenn das nicht gelingt, löst dies beim ersten Zugriff auf die enthaltenen Daten einen Fehler aus.

Dies sollten Sie natürlich umgehen, und zwar indem das Frontend beim Öffnen prüft, ob die Datenbank mit den Tabellen sich an Ort und Stelle befindet. Ist dies nicht der Fall, soll die Anwendung einen Dialog zur Auswahl des neuen Speicherortes der Backend-Datenbank anbieten. Nach der Auswahl der Datei erneuert das Frontend dann VBA-gesteuert die Verknüpfung mit den Tabellen im Backend.

Das sollte nun aber nicht mit jeder neuen Version nötig sein, denn erstens kann es ja sein, dass es regelmäßig Updates gibt, und zweitens arbeiten vielleicht viele Benutzer mit der Datenbank. Wenn dann alle paar Tage oder Wochen einige Mitarbeiter einen Dateidialog bemühen müssen, um das Backend neu auszuwählen, kostet das unnötig Zeit und Geld.

Also sorgen wir dafür, dass die neue Frontend-Datenbank den aktuellen Speicherort der Backend-Datenbank anderweitig erfährt. Dazu gibt es verschiedene Möglichkeiten: zum Beispiel einen Eintrag in die Registry oder eine kleine Textdatei, die sich im gleichen Verzeichnis wie die Frontend-Datenbank befindet. Wenn Sie nun eine neue Version des Frontends auf den Rechner des Benutzers kopieren, prüft das Frontend beim Öffnen zunächst, ob der angegebene Speicherort für die verknüpften Tabellen noch stimmt. Falls nicht, liest es den zuletzt verwendeten Speicherort aus der bereits erwähnten Textdatei ein und verknüpft die Tabellen neu. Hier können nun eigentlich nur noch zwei Dinge dazwischenkommen: entweder ist die Backend-Datei nicht mehr am erwarteten Ort oder das Netzwerk ist unterbrochen.

17.1 Datenbank in Frontend und Backend aufteilen

Kommen wir nun zunächst zum einfachen Teil: dem Aufteilen einer Datenbank in Front- und Backend. Dazu erstellen Sie eine neue Datenbank-Datei, die beispielsweise *AEMA_BE.mdb* heißt (*_BE* kennzeichnet das Backend). Aus Gründen der Performance bei verknüpften Access-

Datenbanken sollte der Dateiname nicht länger als acht Zeichen sein und keine Erweiterung von mehr als drei Zeichen besitzen, daher benennen wir die Endung einer neu erstellten *.accdb*-Datenbank in *.mdb* um (dies gilt nur für das Backend!). Öffnen Sie die Datenbank und wählen Sie den Ribbon-Eintrag *Externe Daten|Importieren und Verknüpfen|Access* aus (siehe Abbildung 17.1).

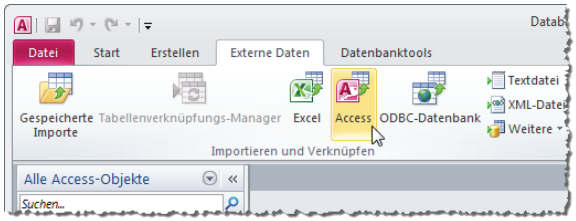


Abbildung 17.1: Importieren von Tabellen einer anderen Access-Datenbank

Wählen Sie im folgenden Dialog die aufzuteilende Datenbank aus, also *<Pfad>\AEMA.accdb*. Behalten Sie die *Importieren...*-Option bei und klicken Sie auf *OK*. Es erscheint der Dialog *Objekte importieren* aus Abbildung 17.2. Klicken Sie hier auf die Schaltfläche *Alle Auswählen*.

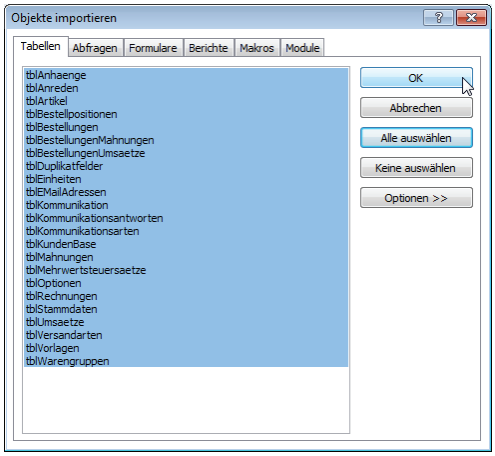


Abbildung 17.2: Importieren aller Tabellen des Frontends in das Backend

Sie sollten sicherheitshalber prüfen, ob die Beziehungen mit importiert werden. Dazu klicken Sie auf die Schaltfläche *Optionen* und aktivieren, sofern noch nicht geschehen, die Option *Importieren|Beziehungen* (siehe Abbildung 17.3). Mit einem Klick auf *OK* importieren Sie nun alle Tabellen der Datenbankdatei *AEMA.accdb* in die Backend-Datenbank *AEMA_BE.mdb*. Aber wollen wir wirklich alle Tabellen der Datenbank in das Backend verschieben? Sollten nicht Tabellen wie *tblOptionen* oder *tblStammdaten* im Frontend verbleiben? Immerhin enthalten diese doch Felder, die auf dem Rechner des jeweiligen Benutzers eingestellt und dort auch wieder abgefragt werden? Die Antwort lautet Nein.

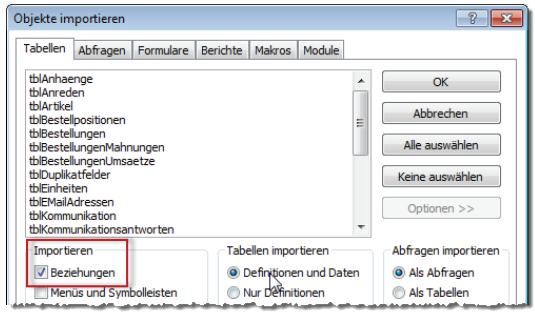


Abbildung 17.3: Stellen Sie sicher, dass auch die Beziehungen mit importiert werden.

Wie oben angemerkt, soll die Aufteilung in diesem Fall zunächst die Möglichkeit eröffnen, das Frontend und somit die Benutzeroberfläche auszutauschen, ohne dass die Daten jeweils in die neue Version übertragen werden sollen – wir gehen also im Rahmen dieses Buchs von einer Einzelplatzanwendung aus. Und selbst wenn wir eine Mehrbenutzer-Anwendung planen, so würden doch die benutzerabhängigen Daten unter Angabe des jeweiligen Benutzers im Backend gespeichert werden. Anders könnte der Benutzer ja gar nicht von verschiedenen Arbeitsplätzen aus auf seine Daten zugreifen, sondern müsste immer vom gleichen Rechner aus arbeiten. Also verschieben wir alle Tabellen in das Backend.

Nun haben wir von jeder Tabelle zwei Exemplare: eine im Frontend und eine im Backend. Die Tabellen im Frontend können Sie löschen: Das Frontend soll schließlich auf die Tabellen im Backend zugreifen, und das realisieren wir durch die Herstellung entsprechender Verknüpfungen. Um die Tabellen zu löschen, markieren Sie diese einzeln im Navigationsbereich und betätigen die *Entf*-Taste. Es erscheinen Meldungen wie die aus Abbildung 17.4, die Sie einfach bestätigen.

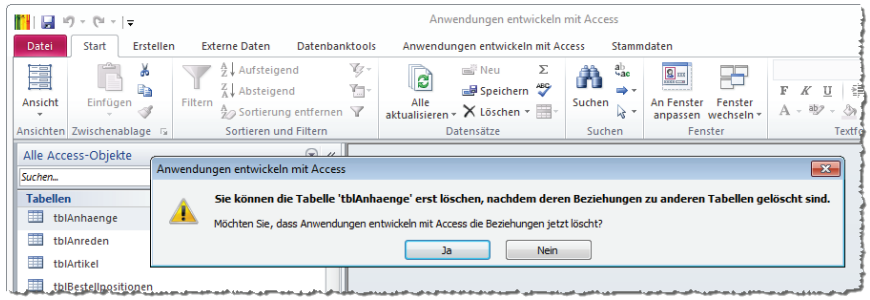


Abbildung 17.4: Löschen einer Tabelle per Navigationsbereich

Die Anwendung funktioniert zu diesem Zeitpunkt natürlich gar nicht mehr, denn es fehlen sämtliche Tabellen. Um dies wieder in Ordnung zu bringen, fügen Sie nun Verknüpfungen auf alle Tabellen in der Backend-Datenbank *AEMA_BE.mdb* hinzu. Dazu betätigen Sie – diesmal vom Frontend aus – den Ribbon-Eintrag *Externe Daten|Importieren und Verknüpfen|Access*. Wählen

Sie im Dialog *Externe Daten* diesmal die Backend-Datenbank als Datenquelle aus. Außerdem aktivieren Sie die zweite Option *Erstellen Sie eine Verknüpfung zur Datenquelle, indem Sie eine verknüpfte Tabelle erstellen* und klicken Sie dann auf *OK* (siehe Abbildung 17.5).

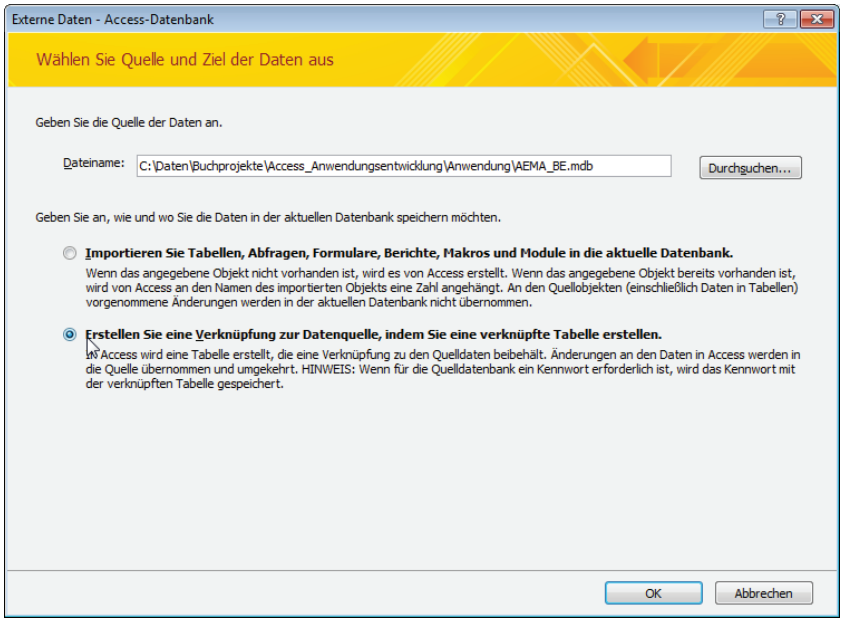


Abbildung 17.5: Vorbereiten der Tabellenverknüpfungen

Wählen Sie im Dialog *Tabellen verknüpfen* aus Abbildung 17.6 mit einem Klick auf *Alle auswählen* alle Tabellen aus und klicken Sie auf *OK*.

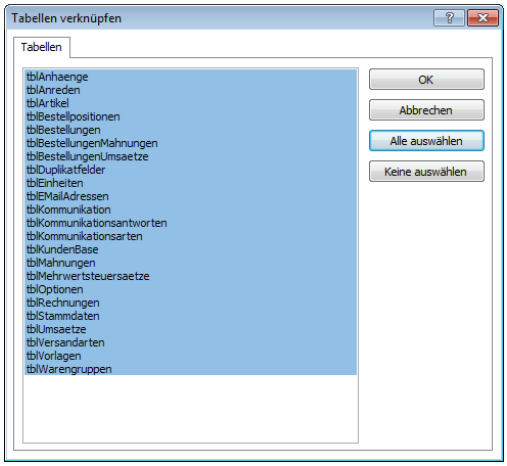


Abbildung 17.6: Auswahl der zu verknüpfenden Tabellen

Dies stellt Verknüpfungen zu allen Tabellen der Backend-Datenbank her, die wie in Abbildung 17.7 dargestellt werden. Sie können nun doppelt auf die verknüpften Tabellen klicken und erhalten genauso Zugriff auf die Daten, als wenn sich die Tabellen in der gleichen Datenbank befinden würden. Das Gleiche gilt für Abfragen, Formulare, Steuerelemente, Berichte und VBA-Code, der auf die in den Tabellen enthaltenen Daten zugreift.

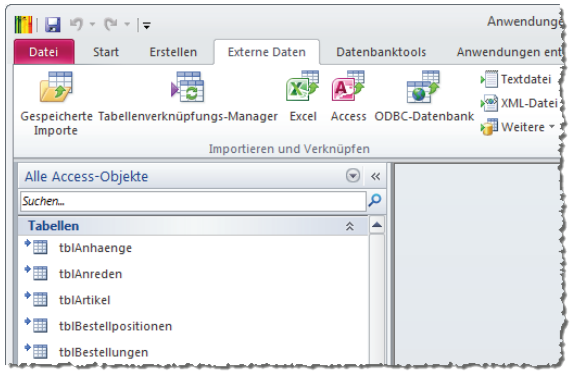


Abbildung 17.7: Verknüpfte Tabellen im Frontend

17.2 Prüfen und Wiederherstellen der Verknüpfung beim Start

Wenn Sie mit einer Frontend/Backend-Lösung arbeiten, speichert die Tabelle *MSysObjects* der Frontend-Datenbank zu jeder Tabelle Name und Pfad der Backend-Datenbank. Diese Tabelle blenden Sie im Navigationsbereich ein, indem Sie mit der rechten Maustaste auf den Titel dieses Bereichs klicken, den Eintrag *Navigationsoptionen...* auswählen und im nun erscheinenden Dialog die Option *Systemobjekte anzeigen* aktivieren. Öffnen Sie die Tabelle *MSysObjects* nun per Doppelklick auf den Eintrag im Navigationsbereich, zeigt diese ihre Informationen wie in Abbildung 17.8 an.

Sollte der Benutzer nun den Speicherort der Backend-Datenbank ändern oder diese gar löschen, findet Access die verknüpften Tabellen nicht mehr unter dem im Feld *Database* der Tabelle *MSysObjects* angegebenen Namen.

Sie können dies testen, indem Sie einfach den Dateinamen der Backend-Datenbank ändern. Wenn Sie nun versuchen, auf eine der Tabellen zuzugreifen, erscheint die Meldung aus Abbildung 17.9.

Um solche Probleme zu verhindern, soll die Anwendung beim Starten prüfen, ob die Tabellen ordnungsgemäß verknüpft sind. Dies untersuchen wir der Einfachheit halber nur anhand einer einzigen Tabelle, und zwar mit der Tabelle *tblOptionen*.

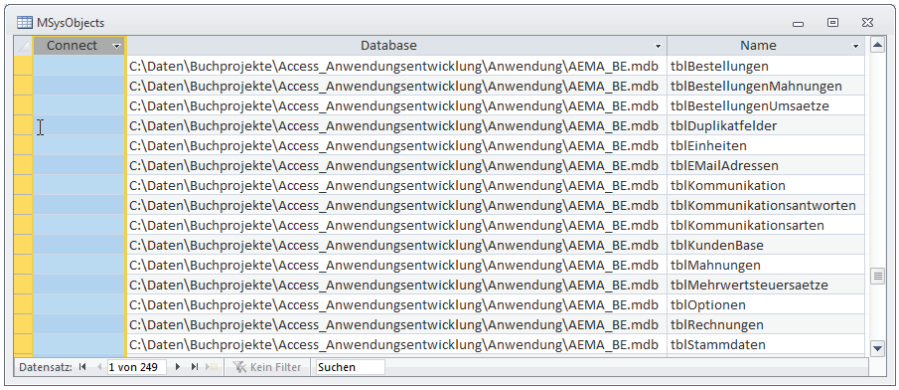


Abbildung 17.8: Pfad und Name der Quelldatenbank verknüpfter Tabellen

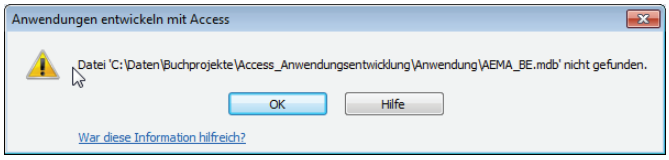


Abbildung 17.9: Meldung beim Fehlen einer verknüpften Tabelle

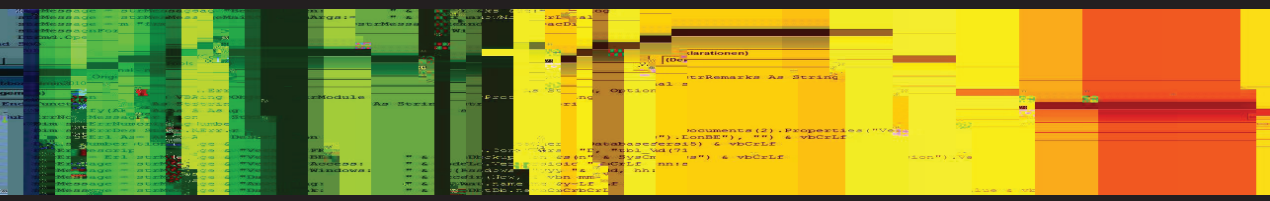
Die Untersuchung erfolgt komplett in der Prozedur *StartDB*, die beim Start der Anwendung durch das automatisch ausgelöste Makro *AutoExec* aufgerufen wird. Dort legen wir zunächst in zwei Konstanten den Namen der Tabelle und des Feldes an, anhand derer wir die Verknüpfung prüfen wollen:

```
Const strTesttabelle As String = "tblAnreden"
Const strTestfeld As String = "AnredeID"
```

Der Teil des Makros *AutoExec*, der diese Funktion auslöst, erledigt noch eine weitere Aufgabe. In einer *Wenn*-Bedingung fragt das Makro den Wert einer temporären Variablen ab, die unter dem Namen *Verknuepft* in der *TempVars*-Auflistung gespeichert wird. Dieser Wert wird von der Funktion *StartDB* auf *True* gesetzt, wenn das Frontend ordnungsgemäß mit dem Backend verknüpft ist, anderenfalls auf *False*. Hat die Variable den Wert *False*, soll die Anwendung gleich wieder geschlossen werden. Diese Aufgabe könnte man theoretisch auch durch einen Aufruf der Methode

```
DoCmd.Quit
```

erledigen. Allerdings führt dies zu einem Fehler, wenn dies in einer von einem Makro aus aufgerufenen VBA-Routine geschieht. Also übergeben wir im Falle der gescheiterten Verknüpfung mit einer temporären Variablen einen Wert an das aufrufende Makro, damit das Makro die Anwendung beenden kann. Doch zunächst zum Ablauf der Funktion *StartDB*, die wir zeilen-



Das Buch: Anwendungen entwickeln mit Access zeigt die Entwicklung einer kompletten Anwendung auf Basis von Microsoft Access. Mit der Anwendung verwalten Sie Kunden, Artikel, Bestellungen, Rechnungen, dokumentieren die Kommunikation mit dem Kunden und enttarnen säumige Zahler. Das Buch beschreibt alle zur Entwicklung der Anwendung notwendigen Schritte und ist daher für Einsteiger wie auch für fortgeschrittene Entwickler geeignet. Die üblichen Grundlagen à la "Was ist eine Tabelle" entfallen, hier wird kurz und knackig erläutert, wie Sie die benötigten Access-Objekte und den VBA-Code erstellen.

Der Autor: Dipl.-Ing. André Minhorst.
Herausgeber und Chefredakteur von Access [basics] – www.access-basics.de.
Ext. Chefredakteur von Access im Unternehmen – www.access-im-unternehmen.de.
Entwickler des Ribbon-Admin 2010 – www.ribbon-admin.de.
Softwareentwickler und Buchautor. Arbeitet mit Access seit 1998.
Verleger. André Minhorst Verlag – www.minhorst.com.

Die Techniken – Best Of:
Erstellen von Tabellen. Definieren des Datenmodells. Aufbau von Formularen. Interaktion zwischen den Formularen. Formulare mit Unterformularen. Übersichtsformulare. Detailformulare. Suchfunktionen für Formulare. Formular-Tuning. Implementieren von Ereignisprozeduren für Formulare und Steuerelemente. Entwerfen eines Rechnungsberichts. Einbinden einer HBCI-Schnittstelle. Automatisches Erstellen von Word-Anschreiben. Erstellen und Einlesen von E-Mails per VBA über Outlook. Professionelle Fehlerbehandlung. Anzeigen von Bildern im Ribbon, in Menüs, im TreeView und auf Schaltflächen. Erstellen von Ribbons und Kontextmenüs. Aufteilen der Datenbank in Frontend und Backend.

Die Anwendung:
Verwalten von Kunden und Artikeln in Übersichts- und Detailformularen. Verwalten von Bestellungen und Bestelldetails. Schreiben von Rechnungen. Rechnungsdruck. Rechnungsversand als PDF per E-Mail. Verfassen von E-Mails an Kunden. Importieren und Zuordnen von Kundenanfragen aus Outlook. Einlesen von Umsätzen und Kontoständen per Onlinebanking. Abgleichen offener Rechnungen mit Zahlungseingängen. Erstellung individueller Anschreiben in Word mit eigenem Template-System.

Der Download:
eBook, Beispieldatenbanken und Tools unter www.minhorst.com.

ANDRÉ MINHORST VERLAG
WWW.MINHORST.COM

Anwendungen entwickeln mit Access
ISBN 978-3-944216-00-3
9 783944 216003
€ 60,00