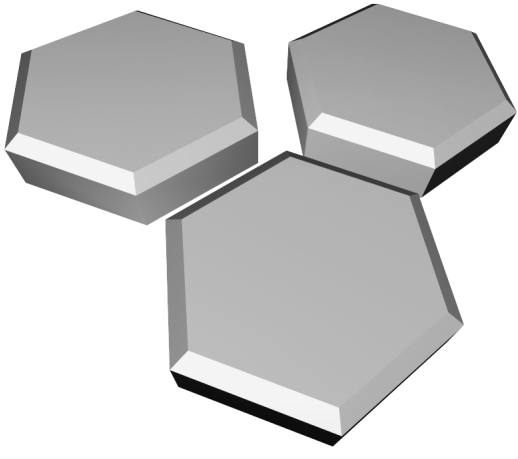
Das Access-VBA Codebook



PEARSON Education

ADDISON-WESLEY

Bernd Held

Das Access-VBA Codebook

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

10 9 8 7 6 5 4 3 2 1
07 06 05

ISBN 3-8273-2177-8

Korrektorat: G & U Technische Dokumentation GmbH, Flensburg
Lektorat: Frank Eller, feller@pearson.de
Herstellung: Monika Weiher, mweiher@pearson.de
Satz: reemers publishing services gmbh, Krefeld – gesetzt aus der Minion
Umschlaggestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Druck und Verarbeitung: Kösel, Krugzell (www.Koeselbuch.de)

Printed in Germany

Inhaltsverzeichnis

Vorwort

Dieses Access-VBA-Codebook ist in erster Linie für fortgeschrittene Anwender und Programmierer gedacht, die noch mehr über VBA in Access erfahren möchten. Die im Buch vorgestellten Quellcodes sind auf der mitausgelieferten CD-ROM in den einzelnen Demodateien sowie dem Code-Repository abrufbar.

Vorab ein paar Worte, was Sie in den einzelnen Kapiteln erwartet:

Im ersten Kapitel können Sie typische Lösungen rund um das Thema Funktionen nachschlagen. Dabei gibt es Access-spezifische sowie VBA-Standardfunktionen.

Im zweiten Kapitel finden Sie interessante API-Funktionen und benutzerdefinierte, eigene Funktionen.

Das dritte Kapitel bildet den Einstieg in die Welt der wichtigsten Access-Objekte. Hier setzen Sie Objekte von der Applikation selbst über Menü- und Symbolleisten bis hin zu Objekten für die Datei- und Verzeichnisverwaltung ein.

Im vierten Kapitel liegen Lösungen zur Tabellenprogrammierung vor. Dabei gibt es in diesem Kapitel jeweils Lösungen zu ADO, DAO und ADOX.

Das fünfte Kapitel gibt Antworten auf Fragen, wie Sie Abfragen erstellen, programmieren und speichern können. Unter anderem werden Aktionsabfragen, Aktualisierungsabfragen und Parameterabfragen in praxisnahen Aufgaben vorgestellt. Auch hier kommen beide Zugriffsmethoden ADO und DAO zum Einsatz.

Kapitel 6 beschäftigt sich mit dem Thema der Berichtsprogrammierung. Sie finden in diesem Kapitel Lösungen, wie Sie Berichte beispielsweise ansprechen, formatieren und erstellen können.

In Kapitel 7 geht es um die Erstellung und Programmierung von Formularen. Dabei werden Lösungen zu den einzelnen Steuerelementen wie Listboxen, Kombinationsfelder und Textboxen präsentiert. Daneben können Sie Beispiele zu weiteren Steuerelementen wie beispielsweise den Fortschrittsbalken, das Kalendersteuerelement und das Office-Chart-Steuerelement vorfinden. Auch Standarddialoge, Meldungsboxen und Eingabeboxen sind Bestandteil dieses Kapitels.

In Kapitel 8 werden Ereignisse in erster Linie auf Formularbasis vorgestellt. Dabei wird aufgezeigt, wie Sie auf bestimmte Ereignisse reagieren können. Die Palette reicht hier von Ereignissen auf Formularebene über Ereignisse auf Steuerelementsebene bis hin zu Tasten- und Mausereignissen.

Kapitel 9 geht der Frage nach, wie man Quellcode und Datenbanken schützen kann. Des Weiteren können Sie in diesem Kapitel nachschlagen, wie Sie Arbeitsgruppen anlegen und Benutzer einfügen oder löschen. Ein weiterer Bestandteil dieses Kapitels ist die VBE-Programmierung. Dabei werden beispielsweise Module, Makros und Ereignisse eingefügt bzw. gelöscht, Quellcodes importiert und exportiert und vieles mehr.

Im letzten Kapitel dieses Buchs wird das Zusammenspiel von Access und anderen Office-Programmen demonstriert. Dabei werden Daten zwischen Access und Outlook, Word, PowerPoint, dem Internet Explorer und Excel ausgetauscht. Ein weiterer Bestandteil in diesem Kapitel ist die Bearbeitung von Textdateien in Access.

Bei Nachfragen und allgemeinem Feedback zu meinem Buch erreichen Sie mich über meine Excel-Homepage http://held-Office.de oder über Bernd.Held@mut.de. So erreichen Sie gleichzeitig auch den Verlag. Auf der Homepage von Markt+Technik finden Sie auch ein eigenes VBA-Forum http://www.mut.de/main/main.asp?page=vbaforum, wo Sie Ihre VBA-Fragen loswerden können. Ich moderiere dieses Forum und antworte dort nahezu täglich. Auch Sie sind natürlich herzlich eingeladen, in diesem VBA-Forum zu antworten.

Vielen Dank an Thomas Tai von SmartTools-Publishing, der mir für die CD-ROM wertvolle Access-Tools zur Verfügung gestellt hat.

Ich wünsche Ihnen während des Lesens des Buchs und beim späteren Anwenden der Lösungen viel Spaß!

Bernd Held

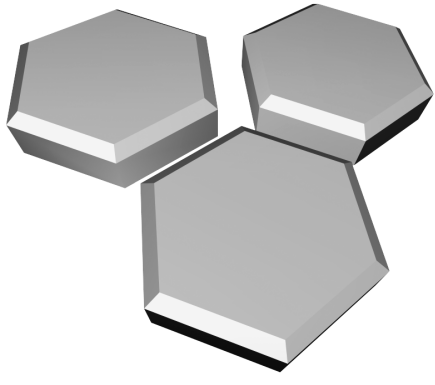
Über den Autor

Bernd Held, MVP für Microsoft Excel

Bernd Held ist gelernter Informatiker und programmierte drei Jahre lang bei einer Firma der Automobilbranche Warenwirtschafts- und Suchsysteme für den Kfz-Bereich. Danach arbeitete er sechs Jahre beim debis Systemhaus im Controlling. Dort war er verantwortlich für das Berichtswesen, die Leistungsverrechnung, das Erstellen von betrieblichen Auswertungen und Wirtschaftlichkeitsrechnungen sowie für die Entwicklung neuer Controlling-Tools auf der Basis von Microsoft Office. Seit dem 1. Januar 2002 ist Herr Held selbstständig. Er schreibt Fachartikel in renommierten Zeitschriften, verfasst Computerbücher, führt Software-Schulungen durch und programmiert im Auftrag von Kunden. Sein Spezialgebiet ist Microsoft Office. Dort hat er sich auf den Bereich Excel und die Office-VBA-Programmierung spezialisiert. Aber auch über Microsoft Works, FrontPage, Windows und diverse andere Themen hat er schon viele Bücher geschrieben. Auch zu Access veröffentlichte er bereits zahlreiche Artikel in Fachzeitschriften.

Vor fünf Jahren wurde Bernd Held als MVP (Most Valuable Professional) von der Firma Microsoft ausgezeichnet. Dieser Titel wird für besondere fachliche Kompetenz, überdurchschnittlichen Einsatz in den Diskussionsforen und für außergewöhnliches Kommunikationstalent verliehen.

Rezepte



Allgemeine VBA-Funktionen

In VBA gibt es Hunderte von Funktionen. Die meisten davon können im kompletten Office-Paket eingesetzt werden, andere sind Access-spezifisch. Funktionen lassen sich in verschiedene Gruppen einteilen. So gibt es beispielsweise Text-, Datums- und Zeitfunktionen, mathematische Funktionen und vieles mehr. Die folgenden Funktionen werden in diesem Kapitel vorgestellt:

- Datums- und Zeitfunktionen
- Textfunktionen
- Dateifunktionen und -anweisungen
- Mathematische Funktionen
- Prüffunktionen
- Umwandlungsfunktionen

1 Zeichenfolge in gültiges Datum umwandeln

Mithilfe der Funktion `CDate` können Zeichenfolgen in einen gültigen Datumswert konvertiert werden. Im folgenden Beispiel aus Listing 1 wird eine Zeichenfolge in ein Datumsformat umgewandelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub DatumWandeln()
   Dim strDatum As String
   strDatum = "17. Mai 2004"
   strDatum = CDate(strDatum)
End Sub
```

Listing 1: Mit der Funktion CDate Zeichenfolgen in gültige Datumswerte umwandeln

Das Datum liegt in einer Variablen vom Typ `String` vor. Übergeben Sie diesen String der Funktion `CDate`, die daraus ein verwertbares Datum macht. Die Funktion `CDate` erkennt sowohl Datumsliterale und Zeitliterale als auch bestimmte Zahlen, die im zulässigen Bereich für ein Datum liegen. Beim Umwandeln einer Zahl in ein Datumsformat wird der ganzzahlige Teil für das Datum verwendet. Nachkommastellen der Zahl werden in eine Zeitangabe (beginnend bei 0:00) umgewandelt.

Selbstverständlich funktioniert dasselbe auch mit Zeitwerten:

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=================================================

Sub ZeitWandeln()
   Dim strZeit As String
   strZeit = "18:45:00"
   strZeit = CDate(strZeit)
End Sub
```

Listing 2: Mit der Funktion CDate Zeichenfolgen in gültige Zeitwerte umwandeln

2 Systemdatum abrufen

Zugriff auf das Systemdatum, welches in Windows hinterlegt ist, erhalten Sie über die Funktion `Date`. Die Funktion `Time` gibt die aktuelle Systemzeit aus.

Ein typisches Beispiel für den Gebrauch dieser Funktionen ist das Anzeigen des aktuellen Tagesdatums sowie der Uhrzeit auf dem Bildschirm.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=================================================

Sub SystemdatumAbrufen()
   MsgBox "Heute ist der " & Date & vbLf & "um " & Time & " Uhr"
End Sub
```

Listing 3: Das aktuelle Tagesdatum anzeigen

Abbildung 1: Das aktuelle Datum sowie die Uhrzeit ermitteln

Hinweis

Sollten das Datum bzw. die Uhrzeit nicht stimmen, dann kontrollieren Sie diese Einstellungen in der Systemsteuerung von Windows.

3 Endtermine errechnen

Die Funktion DateAdd liefert einen Wert vom Typ Variant (Date) zurück, der ein Datum enthält, zu dem ein bestimmtes Zeitintervall addiert wurde. Die Syntax dieser Funktion lautet:

```
DateAdd(interval, number, date)
```

Die Syntax für die DateAdd-Funktion besteht aus den folgenden benannten Argumenten:

Teil	Beschreibung
interval	Zeichenfolgenausdruck, der das zu addierende Zeitintervall ergibt.
Number	Numerischer Ausdruck, der die Anzahl der zu addierenden Intervalle ergibt. Er kann positiv (für ein zukünftiges Datum) oder negativ (für ein vergangenes Datum) sein.
date	Ein Wert vom Typ Variant (Date) oder ein als Literal dargestelltes Datum, zu dem das Intervall hinzuaddiert wird.

Tabelle 1: Die Argumente der Funktion DateAdd

Im folgenden Makro aus Listing 4 wird ausgehend von einem Bestelldatum der Liefertermin errechnet und im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub EndTerminErrechnen()
   Dim DateBestellung As Date
   Dim strIntervall As String
   Dim intZahl As Integer

   DateBestellung = "20.07.2004"
   Debug.Print "Bestelldatum: " & DateBestellung
   strIntervall = "m"
   intZahl = 3
   Debug.Print "Lieferdatum: " & _
   DateAdd(strIntervall, intZahl, DateBestellung)
End Sub
```

Listing 4: Das Enddatum mit der Funktion DateAdd ausrechnen

Über die Variable strIntervall geben Sie bekannt, in welcher Einheit Sie rechnen möchten. Die dafür in Frage kommenden Einheiten können Sie der Tabelle 2 entnehmen.

Einheit	Beschreibung
yyyy	Jahr
q	Quartal
m	Monat
y	Tag des Jahres
d	Tag
w	Wochentag
ww	Woche
h	Stunde
n	Minute
s	Sekunde

Tabelle 2: Die verfügbaren Intervalle für das Argument Interval

Über die Variable `intZahl` geben Sie den Datumsversatz bekannt, der entweder aus einer positiven (Zukunft) oder einer negativen Zahl (Vergangenheit) bestehen kann.

Als letzte Variable übergeben Sie der Funktion das Ausgangsdatum, von dem Sie die Berechnung starten möchten.

Das Ergebnis wird über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung ausgegeben.

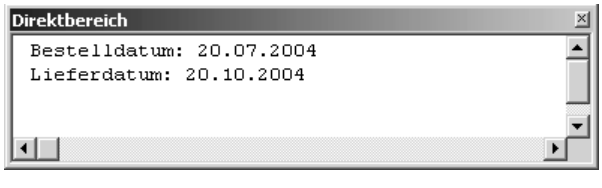


Abbildung 2: Ein Lieferdatum in der Zukunft errechnen

4 Kalenderwoche ermitteln

Mithilfe der Funktion `DatePart` können Sie einen bestimmten Teil des Datums extrahieren, indem Sie auf die Konstanten dieser Funktion zurückgreifen. Unter anderem können Sie dabei auch die Kalenderwoche zu einem Datum ermitteln.

Die Syntax der Funktion `DatePart` lautet:

```
DatePart(interval,date[,firstdayofweek[, firstweekofyear]])
```

Im Argument `Interval` können Sie angeben, welchen Teil des Datums Sie extrahieren möchten. Sehen Sie sich dazu die Tabelle 2 an.

Im Argument `Date` geben Sie den Wert an, den Sie berechnen möchten.

Im Argument `firstdayofweek` müssen Sie den ersten Tag der Woche angeben. Denken Sie beispielsweise daran, dass der jüdische Kalender mit dem Sonntag als ersten Tag der Woche beginnt. Für unseren europäischen Bereich müssen Sie daher den Wert 2 bzw. die Konstante `vbMonday` einsetzen. Wenn Sie die ganze Sache etwas variabler halten möchten, dann setzen Sie die Konstante `vbUseSystem` ein. Damit wird die Einstellung des ersten Tags der Woche direkt aus den Einstellungen Ihrer Windows-Systemsteuerung herausgelesen. Sehen Sie die einzelnen Belegungen der Konstanten in Tabelle 4.

Konstante	Wert	Beschreibung
`VbUseSystem`	0	Die NLS API-Einstellung wird verwendet.
`VbSunday`	1	Sonntag (Voreinstellung)
`VbMonday`	2	Montag
`vbTuesday`	3	Dienstag
`vbWednesday`	4	Mittwoch
`vbThursday`	5	Donnerstag
`vbFriday`	6	Freitag
`vbSaturday`	7	Samstag

Tabelle 3: Die FirstDayOfWeek-Konstanten der Funktion DatePart

Im letzten Argument `firstweekofyear` legen Sie die erste Woche eines Jahres fest. Danach richtet sich auch jeweils die Nummerierung der Kalenderwoche. Dabei können Sie folgende Einstellungen treffen:

Konstante	Wert	Beschreibung
`VbUseSystem`	0	Die NLS API-Einstellung aus der Systemsteuerung von Windows wird verwendet.
`vbFirstJan1`	1	Anfang in der Woche mit dem 1. Januar (Voreinstellung)
`vbFirstFourDays`	2	Anfang in der ersten Woche, die mindestens vier Tage im neuen Jahr enthält.
`VbFirstFullWeek`	3	Anfang in der ersten vollen Woche des Jahres.

Tabelle 4: Die FirstWeekOfYear-Konstanten der Funktion DatePart

Im nächsten Beispiel aus Listing 5 soll anhand des aktuellen Tagesdatums die dazugehörige Wochennummer ermittelt und im Direktfenster der Entwicklungsumgebung ausgegeben werden.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Sub KalenderwocheErmitteln()
   Debug.Print "Heute ist der " & Date
   Debug.Print "Wir befinden uns in der Kalenderwoche " & _
   DatePart("ww", Date)
End Sub
```

Listing 5: Die Kalenderwoche ermitteln

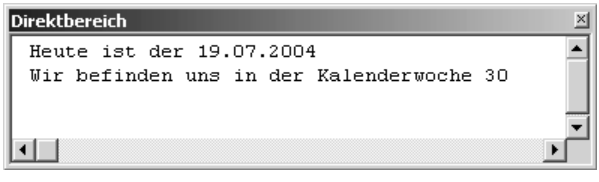


Abbildung 3: Aus einem Datum die dazugehörige KW errechnen

Möchten Sie diese Lösung über eine Funktion aufrufen, dann sehen Sie sich einmal das Listing 6 an.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Function DINKw(Datum)
   DINKw = DatePart("ww", Datum, vbMonday, vbFirstFourDays)
End Function

Sub KW()
   MsgBox "Wir gefinden uns in der KW " & DINKw(Date)
End Sub
```

Listing 6: Die Kalenderwoche über eine Funktion abrufen

Übergeben Sie der Funktion `DINKw` das aktuelle Datum, welches Sie über die Funktion `Date` abrufen. In der Funktion `DINKw` wird die Funktion `DatePart` eingesetzt, die das übergebene Datum untersucht. Den Rückgabewert geben Sie über die Methode `Msgbox` am Bildschirm aus.

5 Datumsdifferenzen errechnen

Mithilfe der Funktion `DateDiff` können Sie die Anzahl der Zeitintervalle angeben, die zwischen zwei bestimmten Terminen liegen. Die Syntax für diese Funktion lautet:

```
DateDiff(interval, date1, date2[, firstdayofweek[, firstweekofyear]])
```

Da die Argumente der Funktion `DateDiff` fast gleich sind wie bei der Funktion `DatePart`, lesen Sie weiter oben im Kapitel nach.

Im folgenden Makro aus Listing 7 werden die Tage bis zum Jahreswechsel errechnet und im Direktfenster ausgegeben.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Sub TageBisZumJahreswechsel()
   Dim DateStart As Date
   Dim DateEnde As Date

   DateStart = Date
   DateEnde = "31.12.2004"
   Debug.Print "Anzahl Tage von heute " & _
   DateStart & vbLf & " bis zum Jahresende: " & _
   DateDiff("d", DateStart, DateEnde)
End Sub
```

Listing 7: Verbleibende Tage bis zum Jahreswechsel ermitteln

Deklarieren Sie zu Beginn des Makros zwei Variablen vom Typ `Date`. Danach füllen Sie die Variable `DateStart` mit dem aktuellen Systemdatum. Die Variable `DateEnde` füllen Sie mit dem letzten Tag des Jahres 2004. Über die Funktion `DateDiff` ermitteln Sie den Zeitraum, der zwischen diesen beiden Zeitpunkten liegt. Damit die Funktion weiß, in welcher Einheit es die Berechnung durchführen soll, übergeben Sie als erstes Argument das Kürzel `d`, das für die Einheit »Tage« steht.

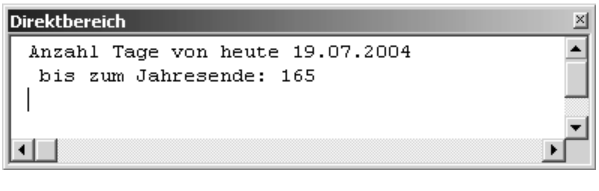


Abbildung 4: Es sind noch genau 165 Tage bis zum Jahreswechsel.

6 Einen Kalender erstellen

Über den Einsatz der Funktion `DateSerial` können Sie einen Wert zurückliefern, der die angegebene Jahres-, Monats- und Tageszahl enthält.

Die Syntax dieser Funktion sieht dabei wie folgt aus:

```
DateSerial(year, month, day)
```

Teil	Beschreibung
`year`	Wert vom Typ Integer. Zahl im Bereich von 100 bis 9999 (einschließlich) oder ein numerischer Ausdruck.
`month`	Wert vom Typ Integer. Beliebiger numerischer Ausdruck.
`day`	Wert vom Typ Integer. Beliebiger numerischer Ausdruck.

Tabelle 5: Die Argumente der Funktion DateSerial

Im folgenden Beispiel aus Listing 8 soll für einen Monat für jeden Tag genau ein Eintrag im Direktfenster in umgekehrter Reihenfolge eingefügt werden.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Sub TageSchreiben()
   Dim intZ As Integer

   For intZ = 29 To 1 Step -1
      Debug.Print Format(DateSerial(2004, 2, intZ), "dd.mm.yy")
   Next intZ
End Sub
```

Listing 8: Einzelne Tage formatiert in den Direktbereich schreiben

Übergeben Sie der Funktion `DateSerial` einzeln und nacheinander die drei benötigten Argumente `Jahr`, `Monat` und `Tag`. Der Tag wird in diesem Beispiel über eine dynamische Variable mit dem Namen `intz` übergeben. Über den Einsatz der Funktion `Format` bringen Sie das Ergebnis noch in das richtige Format. Dabei stehen die Kürzel `dd` für die zweistellige, numerische Tagesangabe, die Kürzel `mm` für eine zweistellige, numerische Monatsangabe sowie die Kürzel `yy` für eine zweistellige, numerische Jahresangabe.

Fangen Sie bei dieser Lösung beim letzten Tag des Monats Januar an und subtrahieren Sie bei jedem Schleifendurchlauf den Wert 1. Die Schrittweite müssen Sie bei diesem Makro über das Schlüsselwort `Step` und das negative Vorzeichen angeben.

Abbildung 5: Die Tage eines Monats rückwärts schreiben

Sollen die Tage in der Reihenfolge vorwärts geschrieben werden, dann starten Sie das Makro aus Listing 9.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub TageSchreibenVorwärts()
   Dim intZ As Integer

   For intZ = 1 To 29
      Debug.Print Format(DateSerial(2004, 2, intZ), "dd.mm.yy")
   Next intZ
End Sub
```

Listing 9: Die Tage eines Monats vorwärts schreiben

In einer Schleife durchlaufen Sie alle Tage des Monats Februar. Dabei fangen Sie beim ersten Februar an und enden beim 29. Februar. Die Schrittweite müssen Sie bei dieser Schleife nicht angeben, da diese Schleife standardmäßig in Einerschritten arbeitet.

Hinweis

In beiden Makros aus Listing 8 und Listing 9 fällt auf, dass der 29. Februar ein konstanter Wert ist, der eben nur für ein Schaltjahr gilt. Im folgenden Kapitel 2 können Sie Funktionen nachschlagen, die den letzten Tag eines Monats ermitteln sowie ein Jahr auf Schaltjahr prüfen.

7 Datumswerte erkennen und umsetzen

Mithilfe der Funktion `DateValue` können Sie eine Zeichenfolge in einen gültigen Datumswert umwandeln. Die Syntax dieser Funktion lautet:

```
DateValue(Datum)
```

Das erforderliche Argument `Datum` ist normalerweise ein Zeichenfolgenausdruck, der ein Datum aus dem Bereich vom 1. Januar 100 bis zum 31. Dezember 9999 repräsentiert. Alter-

nativ können Sie für `Datum` aber auch einen beliebigen Ausdruck angeben, der ein Datum, eine Zeit oder beides aus diesem Bereich darstellen kann.

Im folgenden Beispiel aus Listing 10 wird ein etwas ungewöhnliches Datum in ein gängiges Datumsformat umgesetzt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub EingabeWandeln()
   Dim Date1 As Date

   Date1 = DateValue("1/1/04")
   MsgBox Date1
End Sub
```

Listing 10: Einen String-Wert in ein verwertbares Datum umwandeln

Deklarieren Sie eine Variable vom Typ `Date`. Danach übergeben Sie der Funktion `DateValue` ein Datum, das als Text vorliegt. Die Funktion `DateValue` macht daraus ein Access-verwertbares Datum.

8 Zeitwerte erkennen und umsetzen

Die Funktion `TimeValue` wandelt eine Zeichenfolge in einen gültigen Zeitwert um. Die Syntax dieser Funktion lautet:

```
TimeValue(Uhrzeit)
```

Das erforderliche Argument `Uhrzeit` ist normalerweise ein Zeichenfolgenausdruck, der eine Zeit im Bereich von 0:00:00 bis 23:59:59 darstellt. `Uhrzeit` kann aber auch ein beliebiger Ausdruck sein, der eine Zeit aus diesem Bereich angibt.

Im folgenden Beispiel aus Listing 11 wird der Funktion `TimeValue` ein Zeitwert übergeben, der als Text vorliegt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================
```

Listing 11: Eine Uhrzeit erkennen und umwandeln

```
Sub ZeitenErkennenUndWandeln()
   Dim DateZeit As Date

   Debug.Print "Vorher: " & "2:49:59 PM"
   DateZeit = TimeValue("2:49:59 PM")
   Debug.Print "Nachher:" & DateZeit
End Sub
```

Listing 11: Eine Uhrzeit erkennen und umwandeln (Forts.)

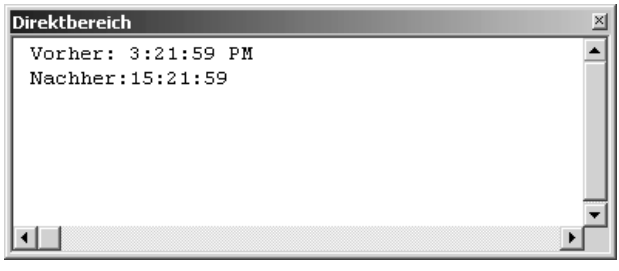


Abbildung 6: Die Uhrzeit wurde erkannt und umgesetzt.

9 Datum zerlegen

Über die folgenden drei Funktionen können Sie ein Datum in seine Einzelteile zerlegen. Dabei kann aus einem Datum der Tagesteil, der Monatsteil sowie der Jahresteil extrahiert werden. Dabei gibt es folgende Funktionen:

Mit der Funktion `Day` können Sie den Tag als ganzzahligen Wert (1–31) aus einem Datumswert extrahieren.

Mit der Funktion `Month` können Sie den Monat als ganzzahligen Wert (1–12) aus einem Datumswert extrahieren.

Mit der Funktion `Year` können Sie das Jahr als ganzzahligen Wert aus einem Datumswert extrahieren.

Die Syntax dieser Funktionen lautet:

```
Day(Datum)
Month (Datum)
Year (Datum)
```

Das erforderliche Argument `Datum` ist ein beliebiger Wert vom Typ `Variant`, ein numerischer Ausdruck, ein Zeichenfolgenausdruck oder eine beliebige Kombination, die ein Datum darstellen kann.

Im folgenden Beispiel aus Listing 12 wird ein Datum zerlegt und anschließend wieder in etwas anderer Form zusammengesetzt:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub DatumZerlegen()
   Dim DateBeginn As Date
   Dim intTag As Integer
   Dim intMonat As Integer
   Dim intJahr As Integer

   DateBeginn = Date
   intTag = Day(DateBeginn)
   intMonat = Month(DateBeginn)
   intJahr = Year(DateBeginn)

   Debug.Print "Aus:  " & DateBeginn
   Debug.Print "wird: " & Format(intTag, "00") & _
      "-" & Format(intMonat, "00") & _
      "-" & Format(intJahr, "0000")
End Sub
```

Listing 12: Datum zerlegen und wieder zusammensetzen

Deklarieren Sie zu Beginn des Makros vier Variablen. In der ersten Variable `DateBeginn` vom Typ `Date` speichern Sie das aktuelle Systemdatum, das Sie über die Funktion `Date` ermitteln. Danach wird diese Variable in die drei Variablen `intTag`, `intMonat` und `intJahr`, die Sie mithilfe der Funktionen `Day`, `Month` und `Year` füllen, zerlegt.

Geben Sie danach das Datum zuerst unverändert im Direktfenster der Entwicklungsumgebung über die Anweisung `Debug.Print` aus. Anschließend setzen Sie die Einzelteile des Datums, die nun in den Integer-Variablen vorliegen, in einer anderen Reihenfolge über den Einsatz des Verkettungsoperators `&` zusammen. Dabei geben Sie mithilfe der Funktion `Format` an, dass die Tages- und Monatsangeben zweistellig sein müssen, da sonst Access die Nullen bei einstelligen Tagen sowie Monaten verschlucken würde. So wird der erste Januar nicht als 1.1.2004, sondern als 01.01.2004 ausgegeben.

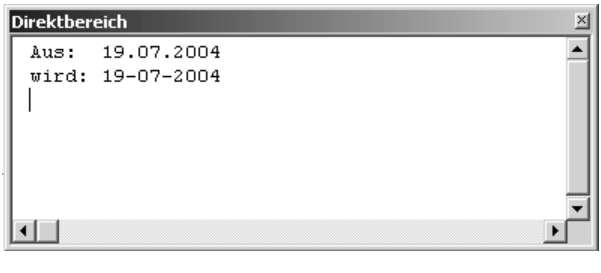


Abbildung 7: Die Funktion Day extrahiert den Tagesanteil aus einem Datum.

10 Die richtige Datumstabelle öffnen

Im nächsten Beispiel aus Listing 13 wird das Öffnen einer Tabelle abhängig vom Tagesdatum vorgenommen. Dazu existieren in der Datenbank insgesamt zwölf Tabellen, von denen die jeweilige Monatstabelle geöffnet wird. Jede Monatstabelle wird mit dem Namen »Monat« und der angehängten Monatsnummer identifiziert. Die Tabelle für den Monat Mai heißt somit `MONAT5`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub MonatsTabelleStarten()
   Dim intMonat As Integer

   On Error GoTo Fehler

   intMonat = Month(Now())
   DoCmd.OpenTable "Monat" & intMonat
   Exit Sub

   Fehler:
      MsgBox "Tabelle konnte nicht gefunden werden!", vbCritical
End Sub
```

Listing 13: Die entsprechende Monatstabelle öffnen

Extrahieren Sie zunächst den Monatsanteil direkt aus der Funktion `Now`, die sowohl das aktuelle Tagesdatum als auch die aktuelle Uhrzeit zurückgibt. Danach wenden Sie die Methode `OpenTable` an, um die entsprechende Tabelle zu öffnen. Sollte die Tabelle nicht gefunden werden können, dann verhindert die Anweisung `On Error GoTo Fehler` einen Makroabsturz.

11 Erstellungsdatum einer Datenbank ermitteln

Die Funktion `FileDatTime` liefert das Erstellungsdatum einer Datei, ohne dass die Datei geöffnet sein muss. Die Syntax dieser Funktion lautet:

```
FileDateTime(Pfadname)
```

Das erforderliche Argument `Pfadname` ist ein Zeichenfolgenausdruck, der einen Dateinamen angibt. Pfadname kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten.

Im folgenden Beispiel aus Listing 14 wird das Erstellungsdatum der aktiven Datenbank ermittelt und am Bildschirm ausgegeben.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Sub ErstellungsdatumAbfragen()
   MsgBox "Datei erstellt am " & vbLf & _
   FileDateTime(Application.CurrentDb.Name), vbInformation
End Sub
```

Listing 14: Das Erstellungsdatum einer Datenbank ermitteln und ausgeben

Über die Eigenschaft `Name` können Sie den kompletten Pfadnamen der Datenbank ermitteln und an die Funktion `FileDateTime` übergeben. Die Methode `CurrentDb` gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Abbildung 8: Das Erstellungsdatum der aktuellen Datenbank abfragen

Im folgenden Beispiel aus Listing 15 wird auf eine andere, nicht geöffnete Datenbank zugegriffen und das Erstellungsdatum der Datei ermittelt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Sub ErstellungsdatumAbfragenAndereDB()

   On Error GoTo fehler

   MsgBox "Datei erstellt am " & vbLf & _
   FileDateTime("C:\Eigene Dateien\Kundendienst1.mdb"), vbInformation
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 15: Das Erstellungsdatum einer nicht geöffneten Datenbank ermitteln

Übergeben Sie der Methode `FileDateTime` den vollen Pfad der Datenbank, deren Erstellungsdatum Sie abfragen möchten. Die `On Error`-Anweisung tritt hier in Kraft, falls die Datenbank bzw. das Verzeichnis nicht gefunden werden können. In diesem Fall wird in den Paragraphen `fehler` verzweigt und über das Objekt `Err` die eindeutige Fehlernummer sowie die Fehlerbeschreibung ausgegeben.

Hinweis

Das Erstellungsdatum über die Methode `FileDateTime` können Sie nicht nur für Datenbankdateien, sondern auch für alle anderen Dateien anwenden.

Im folgenden Beispiel aus Listing 16 wird ein komplettes Verzeichnis mit allen Dateien ausgelesen, gezählt und das Erstellungsdatum einer jeden Datei in das Direktfenster der Entwicklungsumgebung geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub ErstellungsdatumAlleDateienErmitteln()
   Dim strOrdner As String
   Dim intDatei As Integer
   Dim strDatei As String

   strOrdner = "C:\Eigene Dateien\"
   strDatei = Dir(strOrdner & "*.*")
   intDatei = 1

   Do Until strDatei = ""
      Debug.Print "Datei-Nr " & intDatei
      Debug.Print "Dateiname: " & strDatei
      Debug.Print "Erstellt am: " & FileDateTime(strOrdner & strDatei) & vbLf
      strDatei = Dir
      intDatei = intDatei + 1
   Loop
End Sub
```

Listing 16: Das Erstellungsdatum aller Dateien eines Ordners ermitteln und ausgeben

Deklarieren Sie zu Beginn des Makros aus Listing 16 einige Variablen. In der Variablen `strOrdner` geben Sie das Verzeichnis an, aus dem Sie die Dateien auslesen möchten.

Diese starre Angabe können Sie beispielsweise auch dynamisch gestalten – über den Einsatz einer API-Funktion, die einen Verzeichnisbaum in einem Dialogfenster anzeigt, aus dem Sie dann das gewünschte Verzeichnis zur Laufzeit des Makros elegant auswählen können. Dazu aber mehr im nächsten Kapitel.

In der Variabeln `strDatei` speichern Sie die erste Datei, die über die Funktion `Dir` gefunden wird. Übergeben Sie der Funktion dabei den Ordner sowie einen Dateifilter. Das Kürzel `*.*` bedeutet, dass alle Dateien in diesem Verzeichnis gesucht werden sollen.

In einer `Do until`-Schleife suchen Sie nun so lange nach Dateien, bis keine mehr zu finden sind. Innerhalb der Schleife geben Sie die jeweilige Nummer der Datei über die Variable `intDatei` aus, die Sie bei jedem Schleifendurchlauf um den Wert 1 erhöhen. Auch den Namen jeder gefundenen Datei geben Sie im Direktfenster aus, dabei steht der Name jeweils in der Variablen `StrDatei`, die bei jedem Schleifendurchlauf ebenfalls aktualisiert wird.

Bei der Ermittlung des Erstellungsdatums über die Funktion `FileDateTime` müssen Sie darauf achten, dass Sie den kompletten Pfad der gefundenen Datei an die Funktion übergeben.

Über den erneuten Einsatz der Funktion `Dir` innerhalb der Schleife füllen Sie die Variable `StrDatei` mit dem Namen der nächsten gefundenen Datei.

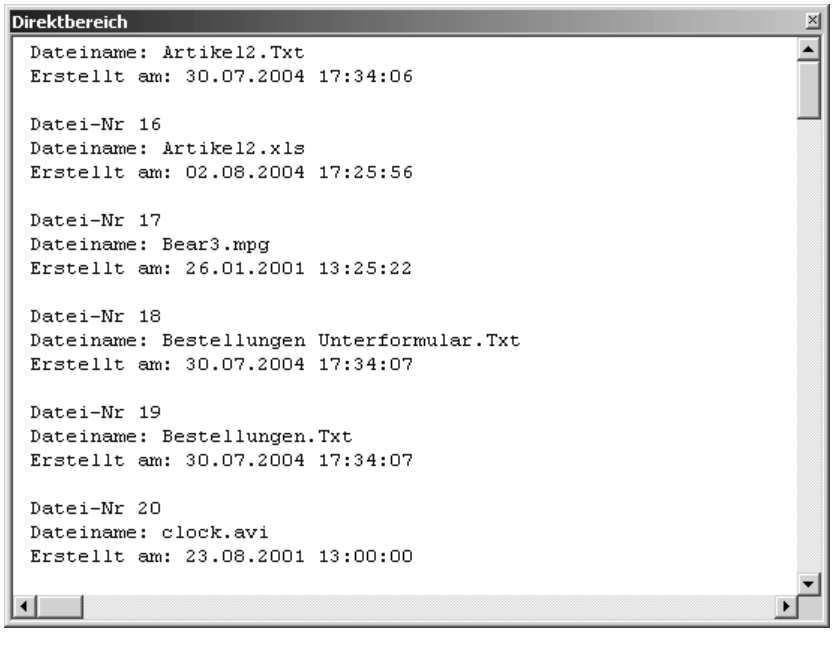


Abbildung 9: Die Namen und das Erstellungsdatum der Dateien wurde dokumentiert.

12 Datumsangaben formatiert ausgeben

Für das Anzeigen von Datums- und Zeitangaben stehen Ihnen fertige Systemkonstanten zur Verfügung, die die Formatierung des Datums bzw. des Zeitwerts für Sie übernehmen.

Diese Datums-/Zeitkonstanten werden im Zusammenspiel mit der Funktion `FormatDateTime` verwendet. Die Syntax dieser Funktion lautet:

```
FormatDateTime(Datum[,BenanntesFormat])
```

Im Argument `Datum` übergeben Sie der Funktion einen Datumswert.

Im Argument `BenanntesFormat` wählen Sie eine der in der Tabelle folgenden Datums-/Zeitkonstanten.

Konstante	Wert	Beschreibung
`vbGeneralDate`	0	Zeigt ein Datum und/oder eine Uhrzeit an. Wenn es ein Datum gibt, wird es in Kurzform angezeigt. Wenn es eine Uhrzeit gibt, wird sie im langen Format angezeigt. Falls vorhanden, werden beide Teile angezeigt.
`vbLongDate`	1	Zeigt ein Datum im langen Datumsformat an, das in den Ländereinstellungen des Computers festgelegt ist.
`VbShortDate`	2	Zeigt ein Datum im kurzen Datumsformat an, das in den Ländereinstellungen des Computers festgelegt ist.
`vbLongTime`	3	Zeigt eine Uhrzeit in dem Zeitformat an, das in den Ländereinstellungen des Computers festgelegt ist.
`vbShortTime`	4	Zeigt eine Uhrzeit im 24-Stunden-Format (hh:mm) an.

Tabelle 6: Die Datumskonstanten

Im folgenden Beispiel aus Listing 17 wird ein Datum auf unterschiedlichste Art und Weise im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=================================================

Sub DatumFormatiertAusgeben()
   Dim DateBeginn As Date

   DateBeginn = Now
   Debug.Print FormatDateTime(DateBeginn, vbGeneralDate)
   Debug.Print FormatDateTime(DateBeginn, vbLongDate)
   Debug.Print FormatDateTime(DateBeginn, vbShortDate)
   Debug.Print FormatDateTime(DateBeginn, vbLongTime)
   Debug.Print FormatDateTime(DateBeginn, vbShortTime)
End Sub
```

Listing 17: Tagesdatum verschiedenartig formatieren

Je nach Einsatz der Konstanten, die Sie bei der Funktion anwenden, sieht das Ergebnis anders aus. Sehen Sie sich dazu die Ergebnisse in Abbildung 10 an.

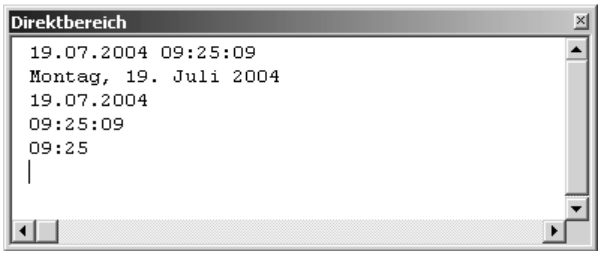


Abbildung 10: Verschiedene Möglichkeiten, ein Datum auszugeben

13 Eine Pause einlegen ...

Neben den Datumsfunktionen `Year`, `Month` und `Day` gibt es auch Funktionen, um einen Zeitwert in seine Einzelteile zu zerlegen. Dabei gibt es folgende Zeitfunktionen:

Die Funktion `Hour` liefert die Stunde aus einem Datums-/Zeitwert als ganzzahligen Wert (0–23).

Die Funktion `Minute` liefert die Minute aus einem Datums-/Zeitwert als ganzzahligen Wert (0–59).

Die Funktion `Second` liefert die Sekunde aus einem Datums-/Zeitwert als ganzzahligen Wert (0–59).

Die Syntax dieser Funktionen lautet:

```
Hour(Uhrzeit)
Minute(Uhrzeit)
Second(Uhrzeit)
```

Das erforderliche Argument `Uhrzeit` ist ein beliebiger Wert vom Typ `Variant`, ein numerischer Ausdruck, ein Zeichenfolgenausdruck oder eine beliebige Kombination, die eine Uhrzeit darstellen kann.

Im folgenden Beispiel aus Listing 18 wird eine Zeitverzögerung in Access eingebaut. Dabei soll die Verarbeitung für genau 10 Sekunden unterbrochen werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================
```

Listing 18: Die Anwendung für 10 Sekunden anhalten

```
Sub PauseEinlegen()
   Dim dateStunde As Date
   Dim dateMinute As Date
   Dim dateSekunde As Date
   Dim datePause As Date

   MsgBox "Bitte OK klicken, dann 10 Sekunden warten!"

   dateStunde = Hour(Now())
   dateMinute = Minute(Now())
   dateSekunde = Second(Now()) + 10

   datePause = TimeSerial(dateStunde, dateMinute, dateSekunde)
   Application.Wait datePause
   MsgBox "Die 10 Sekunden sind abgelaufen", vbInformation
End Sub
```

Listing 18: Die Anwendung für 10 Sekunden anhalten (Forts.)

Speichern Sie zunächst die einzelnen Zeitwerte, indem Sie die Funktionen `Hour`, `Minute` und `Second` einsetzen. Addieren Sie zur letzten Zeitansage den Wert 10, um die Makroverarbeitung um 10 Sekunden zu unterbrechen, und definieren Sie somit die Pausenzeit mithilfe der Funktion `TimeSerial`. Übergeben Sie diese Pausenzeit der Methode `Wait`.

14 Monatsnamen ermitteln

Anhand eines Datums können Sie recht leicht den Monatsnamen ermitteln. Wenn diese Aufgabe automatisiert erledigt werden soll, dann gibt es hierfür in VBA eine Funktion.

Die Funktion `MonthName` gibt eine Zeichenfolge zurück, die den festgelegten Monat angibt. Die Syntax dieser Funktion lautet:

```
MonthName(Monat[, abkürzen])
```

Die Syntax der `MonthName`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
`Monat`	Die numerische Bezeichnung des Monats, z. B. 1 für Januar, 2 für Februar usw.
`Abkürzen`	Optionales Argument. Boolescher Wert, der angibt, ob der Monatsname abgekürzt wird. Wird er ausgelassen, ist die Standardeinstellung `False`, d.h., der Monatsname wird nicht abgekürzt.

Tabelle 7: Die Argumente der Funktion MonthName

Im folgenden Beispiel aus Listing 19 wird der Monatsname aus dem aktuellen Tagesdatum ermittelt und im Direktbereich der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub MonatsnameErmitteln()
   Debug.Print "Heute ist der " & Date
   Debug.Print "verkürzt: " & MonthName(Month(Date), True)
   Debug.Print "normal: " &  MonthName(Month(Date), False)
End Sub
```

Listing 19: Den Monatsnamen ermitteln

Die Funktionsweise der Zeilen können Sie im Prinzip von rechts nach links lesen. Zuerst wird der Monatsteil über die Funktion `Month` aus dem aktuellen Tagesdatum ermittelt. Danach wird diese Nummer, die zwischen 1 und 12 liegt, an die Funktion `MonthName` übergeben. Die Funktion `MonthName` macht daraus automatisch den dazugehörigen Monatsnamen. In gekürzter Form lauten diese Namen Jan bis Dez und in normaler Form Januar bis Dezember.

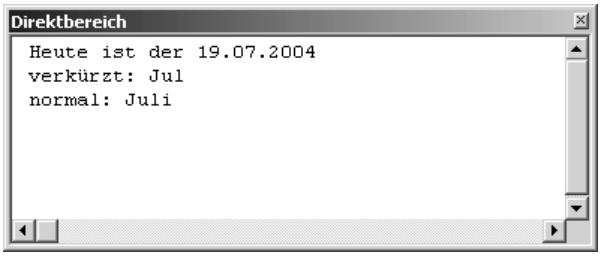


Abbildung 11: Den Monatsnamen aus einem Datum extrahieren

15 Wochentag ermitteln

Auf den ersten Blick etwas schwieriger ist es, aus einem Datum, das in der Zukunft oder der Vergangenheit liegt, den exakten Tagesnamen zu ermitteln, sofern kein Kalender zur Hand ist. Mithilfe von VBA ist diese Aufgabe ein Kinderspiel. Dabei wird die Aufgabe im Zusammenspiel zweier Funktionen gelöst: `WeekDay` und `WeekDayName`.

Soll aus einem Tagesdatum der dazugehörige Wochentag ermittelt werden, dann können Sie die Funktion `Weekday` einsetzen. Diese Funktion gibt den Wochentag aus einem Datumswert zurück (1–7). Die Syntax lautet:

```
Weekday(date, [firstdayofweek])
```

Die Syntax für die `Weekday`-Funktion besteht aus folgenden benannten Argumenten:

Teil	Beschreibung
Date	Erforderlich. Wert vom Typ Variant, numerischer Ausdruck, Zeichenfolgenausdruck oder eine beliebige Kombination, die ein Datum darstellen kann. Wenn Date den Wert Null enthält, wird Null zurückgegeben.
Firstdayofweek	Optional. Eine Konstante, die den ersten Wochentag angibt. Wird die Konstante nicht angegeben, so wird vbSunday angenommen.

Tabelle 8: Die Argumente der Funktion Weekday

Das Argument firstdayofweek hat folgende Einstellungen:

Konstante	Wert	Beschreibung
vbUseSystem	0	NLS API-Einstellung wird verwendet.
VbSunday	1	Sonntag (Voreinstellung)
vbMonday	2	Montag
vbTuesday	3	Dienstag
vbWednesday	4	Mittwoch
vbThursday	5	Donnerstag
vbFriday	6	Freitag
vbSaturday	7	Samstag

Tabelle 9: Die Argumente von Firstdayofweek

Die Funktion WeekDayName gibt eine Zeichenfolge zurück, die den festgelegten Wochentag angibt. So erwartet diese Funktion einen Wert zwischen 1 und 7, um daraus einen lesbaren Wochentag in der Form Montag bis Sonntag zu machen. Die Syntax dieser Funktion lautet:

```
WeekdayName(Wochentag, abkürzen, ErsterWochentag)
```

Die Syntax der WeekdayName-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
Wochentag	Erforderlich. Die numerische Bezeichnung des Wochentags. Der numerische Wert der einzelnen Tage hängt von der Einstellung für ErsterWochentag ab.
Abkürzen	Optional. Boolescher Wert, der angibt, ob der Name des Wochentags abgekürzt wird. Wird er ausgelassen, ist die Standardeinstellung False, d.h., der Name des Wochentags wird nicht abgekürzt.
ErsterWochentag	Optional. Numerischer Wert, der den ersten Tag der Woche angibt.

Tabelle 10: Die Argumente der Funktion WeekdayName

Im folgenden Listing 20 werden ausgehend von einem vorgegebenen Datum der dazugehörige Wochentag sowie das Quartal ermittelt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'====================================================

Function Quartal(DatAngabe)
   Quartal = DatePart("q", DatAngabe)
End Function


Function Wochentag(DatAngabe)
   Wochentag = WeekdayName(Weekday(DatAngabe, vbUseSystemDayOfWeek), False)
End Function


Sub WochenTagUndQuartal()
   Debug.Print " Dieses Datum liegt im " & Quartal("20.07.2004") & " .Quartal!"
   Debug.Print " Der Wochentag ist ein " & Wochentag("20.07.2004")
End Sub
```

Listing 20: Quartal- und Wochentagnamen ermitteln

Die Funktion `Weekday` gibt einen Wert zwischen 1 und 7 zurück. Diesen Wert übergeben Sie an die Funktion `WeekDayName`, die dann in ausgeschriebener Form den Tagesnamen liefert.

Das Quartal eines Datums können Sie über die Funktion `DatePart` ermitteln. Wählen Sie dabei als erstes Argument das Kürzel `q`, damit die Funktion weiß, dass das Quartal des übergebenen Datums ermittelt werden soll.

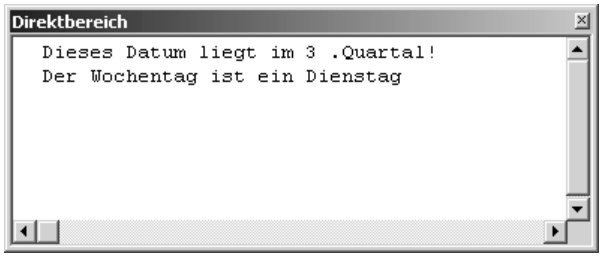


Abbildung 12: Der Wochentag sowie das Quartal wurden ausgelesen.

16 Buchstaben aus Zeichenfolgen entfernen

Um festzustellen, ob es sich bei einem Zeichen um ein alphanumerisches oder ein numerisches Zeichen handelt, können Sie die Funktion `ASC` einsetzen. Diese Funktion gibt einen

Wert vom Typ `Integer` zurück, der den Zeichencode entsprechend dem ersten Buchstaben in einer Zeichenfolge darstellt. Die Syntax dieser Funktion lautet:

```
Asc(Zeichenfolge)
```

Das erforderliche Argument `Zeichenfolge` ist ein beliebiger gültiger Zeichenfolgenausdruck. Wenn `Zeichenfolge` keine Zeichen enthält, tritt ein Laufzeitfehler auf.

Im folgenden Beispiel aus Listing 21 werden über eine benutzerdefinierte Funktion alle Buchstaben aus einem Text entfernt, so dass nur noch Zahlenwerte zurückbleiben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Function BuchstRaus(s As String) As String
   Dim intZ As Integer

   For intZ = 1 To Len(s)

      Select Case Asc(Mid(s, intZ, 1))
         Case 0 To 64, 123 To 197
            BuchstRaus = BuchstRaus & Mid(s, intZ, 1)
     End Select

   Next intZ
End Function


Sub BuchstabenUndZahlen()
   MsgBox "Aus 'Wert0815' " & vbLf & "bleibt übrig" & _
      vbLf & BuchstRaus("Wert0815"), vbInformation
End Sub
```

Listing 21: Buchstaben aus Zeichenfolgen entfernen

Übergeben Sie der Funktion `BuchstRaus` einen Text, der sowohl Buchstaben als auch Zahlenwerte beinhaltet.

Setzen Sie in der Funktion eine Schleife auf, die Zeichen für Zeichen abarbeitet. Prüfen Sie mithilfe der Funktion `Asc` das jeweils aktuelle Zeichen des übergebenen Textes aus dem Makro `BuchstabenUndZahlen`, indem Sie dieses in einen Integer-Wert umwandeln. Mit der Funktion `Mid` extrahieren Sie dabei jeweils das nächste Zeichen aus dem übergebenen Text. Dabei entsprechen die Werte 65 bis 90 Kleinbuchstaben, die Werte 97 bis 122 den Großbuchstaben und die restlichen Werte den Sonderzeichen.

Als Rückgabe der Funktion werden nur noch numerische Zeichen an das aufrufende Makro zurückgegeben.

Abbildung 13: Alle Buchstaben in der Zeichenfolge wurden entfernt.

17 Zahlen aus Zeichenfolgen entfernen

Analog zur Lösung aus Listing 21 können Sie ebenso numerische Werte aus einer Zeichenfolge entfernen, indem Sie das Makro aus Listing 22 starten.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=================================================

Function ZahlenRaus(s As String) As String
   Dim intZ As Integer
   For intZ = 1 To Len(s)
      Select Case Asc(Mid(s, intZ, 1))
         Case 0 To 64, 123 To 197
         Case Else
            ZahlenRaus = ZahlenRaus & Mid(s, intZ, 1)
      End Select
   Next intZ
End Function

Sub ZahlenUndBuchstaben()
   MsgBox "Aus 'Wert0815' " & vbLf & "bleibt übrig" & _
      vbLf & ZahlenRaus("Wert0815"), vbInformation
End Sub
```

Listing 22: Zahlen aus Zeichenfolge entfernen

Als Erweiterung zum Makro aus Listing 21 wurde im Makro aus Listing 22 ein `Case Else`-Zweig eingefügt, in dem die »Zerstückelung« durchgeführt wird.

Abbildung 14: Alle Zahlen wurden aus der Zeichenfolge entfernt.

18 Zahlungsart definieren

Um Schreibarbeit zu sparen, können Sie beispielsweise eine Zahlungsart über eine Nummer definieren und ansprechen, dann können Sie die Funktion `Choose` einsetzen. Diese Funktion wählt einen Wert aus einer Liste von Argumenten aus und gibt ihn zurück. Die Syntax für diese Funktion lautet:

```
Choose(Index, Auswahl-1[, Auswahl-2, ... [, Auswahl-n]])
```

Teil	Beschreibung
`Index`	Erforderlich. Numerischer Ausdruck oder Feld, der bzw. das einen Wert von 1 bis zur Anzahl der möglichen Auswahlwerte ergibt.
`Auswahl`	Erforderlich. Ein `Variant`-Ausdruck, der einen der möglichen Auswahlwerte enthält.

Tabelle 11: Die Argumente der Funktion Choose

Im folgenden Beispiel aus Listing 23 wird über eine Funktion die Zahlungsart ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Function Auswahl(intz As Integer)
   Auswahl = Choose(intz, "Bar", "EC", "Rechnung")
End Function

Sub AutoTexte()
   Debug.Print Auswahl(1)
   Debug.Print Auswahl(2)
   Debug.Print Auswahl(3)
End Sub
```

Listing 23: Die Zahlungsart über eine Nummer auswählen

Das erste Wort in der Argumentfolge ist das Wort `Bar`. Es kann über die Nummer 1 angesprochen werden. Auf diese Art und Weise können beispielsweise auch ganze Sätze über kurze Nummern angesprochen werden.

19 Aus Zahlen Buchstaben machen

Die Funktion `Chr` gibt einen Wert vom Typ `String` zurück, der das Zeichen enthält, das dem angegebenen Zeichencode zugeordnet ist. Die Syntax dieser Funktion lautet:

```
Chr(Zeichencode)
```

Das erforderliche Argument `Zeichencode` ist ein Wert vom Typ `Long`, der ein Zeichen festlegt.

Abbildung 15: Autotexte wie z.B. die Zahlungsart über eine Nummer ansprechen

Im folgenden Beispiel aus Listing 24 werden mithilfe der Funktion `Chr` aus einzelnen Zahlen Buchstaben gewonnen und in den Direktbereich geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub AusZahlenBuchstabenErzeugen()
   Dim intz As Integer

   For intz = 65 To 90
      Debug.Print Chr(intz)
   Next intz
End Sub
```

Listing 24: Die Zahlen 65 bis 90 stehen für die Großbuchstaben.

Übergeben Sie der Funktion `Chr` einen Wert zwischen 65 und 90. Damit können Großbuchstaben erzeugt werden.

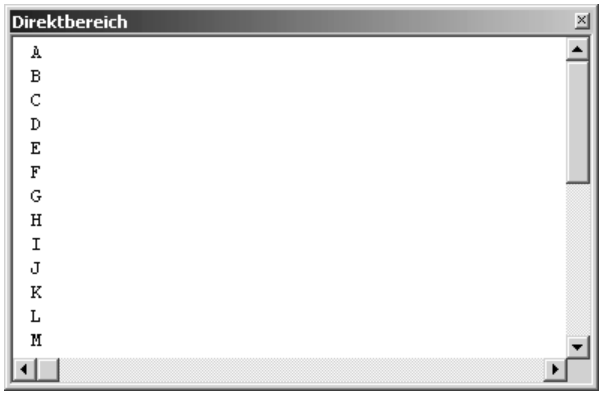


Abbildung 16: Zahlen wurden in Buchstaben umgewandelt.

20 Zeichenfolgen vergleichen

Um zwei Zeichenfolgen miteinander zu vergleichen bzw. um festzustellen, ob eine Zeichenfolge in einer anderen Zeichenfolge auftaucht, setzen Sie die Funktion `InStr` ein. Über diese Funktion können Sie einen Wert vom Typ `Variant (Long)` zurückgeben, der die Position des ersten Auftretens einer Zeichenfolge innerhalb einer anderen Zeichenfolge angibt.

```
InStr([Start, ]Zeichenfolge1, Zeichenfolge2[, Vergleich])
```

Die Syntax der `InStr`-Funktion verwendet die folgenden Argumente:

Teil	Beschreibung
`Start`	Optional. Numerischer Ausdruck, der die Startposition für die Suche festlegt. Wird `Start` nicht angegeben, so beginnt die Suche mit dem ersten Zeichen in der Zeichenfolge. Wenn `Start` den Wert Null enthält, tritt ein Fehler auf. Bei Angabe von `Vergleich` muss auch das Argument `Start` angegeben werden.
`Zeichenfolge1`	Erforderlich. Durchsuchter Zeichenfolgenausdruck.
`Zeichenfolge2`	Erforderlich. Gesuchter Zeichenfolgenausdruck.
`Vergleich`	Optional. Legt die Art des Zeichenfolgenvergleichs fest. Der Wert Null für `Vergleich` führt zu einem Fehler.

Tabelle 12: Die Argumente der Funktion InStr

Konstante	Wert	Beschreibung
`vbUseCompareOption`	`-1`	Führt einen Vergleich mithilfe der Option `Compare`-Anweisung durch.
`vbBinaryCompare`	`0`	Führt einen binären Vergleich durch.
`vbTextCompare`	`1`	Führt einen textbasierten Vergleich durch.
`vbDatabaseCompare`	`2`	Nur Microsoft Access. Führt einen Vergleich durch, der auf Informationen in Ihrer Datenbank basiert.

Tabelle 13: Die Vergleichstypen der Funktion InStr

Als Rückgabewerte der Funktion `InStr` gelten folgende Argumente.

Fall	Rückgabewerte von InStr
`Zeichenfolge1` hat die Länge Null.	`0`
`Zeichenfolge1` ist Null.	`Null`
`Zeichenfolge2` hat die Länge Null.	`start`
`Zeichenfolge2` ist Null.	`Null`

Tabelle 14: Die Rückgabewerte der Funktion InSt

Fall	Rückgabewerte von InStr
`Zeichenfolge2` ist nicht vorhanden	0
`Zeichenfolge2` ist in `Zeichenfolge1` enthalten.	Position, an der die Übereinstimmung beginnt
`Start` > `Zeichenfolge2`	0

Tabelle 14: Die Rückgabewerte der Funktion InSt (Forts.)

Im folgenden Beispiel aus Listing 25 wird geprüft, ob es sich um eine gültige E-Mail-Adresse handelt. Dabei wird von der Prämisse ausgegangen, dass eine E-Mail-Adresse dann gültig ist, wenn das Zeichen @ in der Mail-Adresse vorkommt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub EMailCheck()
   Dim strUrl As String

   strUrl = "Held-office@t-online.de"

   If InStr(strUrl, "@") > 1 Then
      MsgBox "E-Mail gültig!"
   Else
      MsgBox "E-Mail ungültig!"
   End If
End Sub
```

Listing 25: Über die Funktion Instr nach dem Zeichen @ suchen

Wird die Zeichenfolge @ in der Variablen `s` gefunden, dann meldet die Funktion einen Rückgabewert größer 1 zurück. Dabei wird die genaue Position des gesuchten Zeichens zurückgegeben. Fängt die E-Mail-Adresse mit der Zeichenfolge @ an, so handelt es sich um eine nicht gültige E-Mail-Adresse.

Wollen Sie den E-Mail-Adressencheck ein wenig erweitern und beispielsweise sicherstellen, dass keine Umlaute wie ä, ö und ü eingesetzt werden, dann starten Sie das Makro aus Listing 26.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================
```

Listing 26: Die erweiterte E-Mail-Adressenprüfung

```
Sub EMailCheckErW()
   Dim strUrl As String

   strUrl = "Held-öffice@t-online.de"

   If InStr(strUrl, "@") > 1 Then
      If InStr(strUrl, "ä") > 0 Or InStr(strUrl, "ö") > 0 _
         Or InStr(strUrl, "ü") > 0 Then
         MsgBox "E-Mail ungültig! Enthält Umlaute!"
      Else
         MsgBox "E-Mail gültig!"
      End If
   Else
      MsgBox "E-Mail ungültig! das Zeichen @ fehlt"
   End If
End Sub
```

Listing 26: Die erweiterte E-Mail-Adressenprüfung (Forts.)

Bei der Lösung aus Listing 26 wird neben der Existenz des Zeichens @ noch nach Umlauten geprüft. Wenn Umlaute in der E-Mail-Adresse enthalten sind, dann meldet die Funktion `InStr` einen Wert ungleich 0.

21 Den Pfad- und Dateinamen einer Datenbank zerlegen

Im vorherigen Beispiel haben Sie das Zeichen @ in einer Zeichenfolge gesucht. Dabei suchte diese Funktion beginnend vom linken Rand der Zeichenfolge.

Muss nun vom rechten Rand einer Zeichenfolge gesucht werden, dann setzen Sie die Funktion `InStrRev` ein. Bei dieser Funktion wird die Suche beginnend vom rechten Rand einer Zeichenfolge gestartet.

```
InstrRev(stringcheck, stringmatch [, start[, compare]])
```

Die Syntax der `InstrRev`-Funktion besteht aus folgenden benannten Argumenten:

Teil	Beschreibung
`Stringcheck`	Erforderlich. Der zu durchsuchende Zeichenfolgenausdruck.
`Stringmatch`	Erforderlich. Zeichenfolgenausdruck, nach dem gesucht wird.
`Start`	Optional. Numerischer Ausdruck, der die Anfangsposition für jede Suche festlegt. Wird er ausgelassen, wird –1 verwendet, d.h., die Suche beginnt an der letzten Zeichenposition.

Tabelle 15: Die Argumente der Funktion InStrRev

Teil	Beschreibung
Compare	Optional. Numerischer Wert, der die Art des Vergleichs angibt, der beim Beurteilen von untergeordneten Zeichenfolgen verwendet werden soll. Wird er ausgelassen, wird ein binärer Vergleich ausgeführt.

Tabelle 15: Die Argumente der Funktion InStrRev (Forts.)

Im folgenden Beispiel aus Listing 27 wird der komplette Pfad sowie der Pfad und der Dateinamen der aktiven Datenbank ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub PfadUndDateiNamen()
   Dim intPos As Integer
   Dim strPfad As String
   Dim strName As String

   intPos = InStrRev(Application.CurrentDb.Name, "\")
   strPfad = Left(Application.CurrentDb.Name, intPos)
   strName = Mid(Application.CurrentDb.Name, intPos + 1)

   MsgBox "Komplett: " & Application.CurrentDb.Name & _
      vbLf & "Pfad: " & strPfad & vbLf & "Datei: " & strName
End Sub
```

Listing 27: Den Pfad- und Dateinamen separieren (ab Access 2000)

Aus der Eigenschaft `Name`, in der der Name der Datei sowie der komplette Speicherpfad verzeichnet sind, werden die einzelnen Informationen herausgezogen. Dabei wird über die Funktion `InStrRev` nach dem Zeichen »\« gesucht, und zwar von der rechten Seite des kompletten Namens aus. Die Position wird hierbei in der Variablen `intPos` zwischengespeichert.

So bekommen Sie den Pfad, indem Sie vom linken Rand so viele Zeichen übertragen, wie in der Variablen `intPos` gespeichert wurden.

Soll nur der Name der Datenbank extrahiert werden, dann wenden Sie die Funktion `Mid` an, um in der Mitte des kompletten Namenstextes aufzusetzen. Dabei soll das Zeichen »\« nicht mehr mit übertragen werden. Daher addieren Sie zur Variablen `intPos` noch den Wert 1.

Anwender der Version Access 97 haben beim Makro aus Listing 27 Pech. Leider gibt es diese Funktion erst ab Access 2000. Für Access 97-Anwender muss daher folgende Syntax aus Listing 28 gelten.

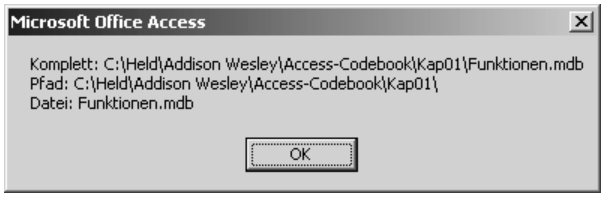


Abbildung 17: Alle Informationen wurden aus einer Angabe separiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub PfadUndDateiAccess97()
   Dim intPos As Integer
   Dim AltPos As Integer
   Dim strPfad As String
   Dim strName As String

   intPos = 1

   Do Until intPos = 0
      AltPos = intPos
      intPos = InStr(intPos + 1, Application.CurrentDb.Name, "\")
   Loop

   strPfad = Left(Application.CurrentDb.Name, AltPos)
   strName = Mid(Application.CurrentDb.Name, AltPos + 1)

   MsgBox "Komplett: " & Application.CurrentDb.Name & _
      vbLf & "Pfad: " & strPfad & vbLf & "Datei: " & strName
End Sub
```

Listing 28: Pfad- und Dateinamen separieren (Access 97)

In einer `Do until`-Schleife arbeiten Sie den kompletten Datei- und Pfadamen ab, indem Sie jeweils über die Funktion `Instr` vom linken Rand aus nach dem Zeichen »\« suchen, bis Sie am rechten »Rand« angekommen sind.

Den Pfad können Sie danach über den Einsatz der Funktion `Left` ermitteln, indem Sie vom linken Textrand so viele Zeichen übertragen, wie in der Variablen `AltPos` verzeichnet sind.

Den Namen der Datei bekommen Sie, indem Sie ausgehend von der letzten Fundstelle des Zeichens »\«, welche in der Variablen `AltPos` verzeichnet ist, den Wert 1 hinzuaddieren und danach genau von dieser Stelle beginnen, mithilfe der Funktion `Mid` die restlichen Zeichen zu übertragen.

22 Zeichenfolgen splitten und zusammenführen

Die Funktion `Join` gibt eine Zeichenfolge zurück, die sich aus der Kombination einer Reihe von untergeordneten Zeichenfolgen ergibt, die in einem Datenfeld enthalten sind. Die Syntax dieser Funktion lautet:

```
Join(sourcearray [, delimiter])
```

Die Syntax der `Join`-Funktion besteht aus folgenden benannten Argumenten:

Teil	Beschreibung
`Sourcearray`	Erforderlich. Eindimensionales Datenfeld, das die zu kombinierenden, untergeordneten Zeichenfolgen enthält.
`Delimiter`	Optional. Zeichen einer Zeichenfolge, mit dem die untergeordneten Zeichenfolgen in der zurückgegebenen Zeichenfolge getrennt werden. Wird es ausgelassen, wird das Leerstellenzeichen (" ") verwendet. Wenn `delimiter` eine Zeichenfolge der Länge Null ("") ist, werden alle Elemente in der Liste ohne Trennzeichen verkettet.

Tabelle 16: Die Argumente der Funktion Join

Die Funktion `Split` gibt ein nullbasiertes, eindimensionales Datenfeld zurück, das eine festgelegte Anzahl an untergeordneten Zeichenfolgen enthält.

```
Split(expression[, delimiter[, limit[, compare]]])
```

Teil	Beschreibung
`Expression`	Erforderlich. Zeichenfolgenausdruck, der untergeordnete Zeichenfolgen und Trennzeichen enthält. Wenn `expression` eine Zeichenfolge der Länge Null ("") ist, gibt `Split` ein leeres Datenfeld zurück, d.h. ein Datenfeld ohne Elemente und ohne Daten.
`Delimiter`	Optional. Zeichen einer Zeichenfolge, mit der die Grenzen von untergeordneten Zeichenfolgen identifiziert werden. Wird es ausgelassen, wird das Leerstellenzeichen (" ") als Trennzeichen verwendet. Wenn `delimiter` eine Zeichenfolge der Länge Null ist, wird ein aus einem Element bestehendes Datenfeld, das die gesamte Zeichenfolge von `expression` enthält, zurückgegeben.
`Limit`	Optional. Anzahl der zurückzugebenden, untergeordneten Zeichenfolgen; –1 gibt an, dass alle untergeordneten Zeichenfolgen zurückgegeben werden.
`Compare`	Optional. Numerischer Wert, der die Art des Vergleichs angibt, der beim Beurteilen von untergeordneten Zeichenketten verwendet werden soll.

Tabelle 17: Die Argumente der Funktion Split

Im folgenden Beispiel aus Listing 29 wird ein Text über die Funktion `Split` zerlegt, aufbereitet und anschließend mithilfe der Funktion `Join` wieder zusammengebastelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub JoinSlpitt()
   Dim strText As String
   Dim Varray As Variant
   Dim intZ As Integer
   Dim intGesamt As Integer

   strText = "Vorname;Nachame;Straße;PLZ;Ort"
   Debug.Print "Vor Konvertierung: " & strText
   Varray = Split(strText, ";")
   intGesamt = UBound(Varray)

   For intz = 0 To iGesamt
      Varray(intz) = UCase(Varray(intz))
   Next intz

   Varray = Join(Varray, "-")
   Debug.Print "Nach Konvertierung: " & Varray
End Sub
```

Listing 29: Texte zerlegen, aufbereiten und wieder zusammensetzen

Über die Funktion `Split` zerlegen Sie zuerst einmal den Inhalt der Variablen `strText` anhand des Trennzeichens Semikolon in seine Einzelbestandteile. Dabei schichten Sie die einzelnen Inhalte in ein Datenfeld um. Über die Funktion `UBound` können Sie die Größe des Datenfelds ermitteln. Diese Funktion liefert den letzten Wert im Datenfeld und gibt diesen als Zählvariable zurück. Danach setzen Sie eine Schleife auf, die beginnend vom ersten Eintrag im Datenfeld (=0) bis zum letzten Eintrag im Datenfeld läuft und über die Funktion `UCase` den Text in Großbuchstaben umwandelt. Im Anschluss an die Schleife wenden Sie die Funktion `Join` an, um das Datenfeld wieder in ein »Gesamt-Feld« auszulesen. Geben Sie dabei beispielsweise ein anderes Trennzeichen an.

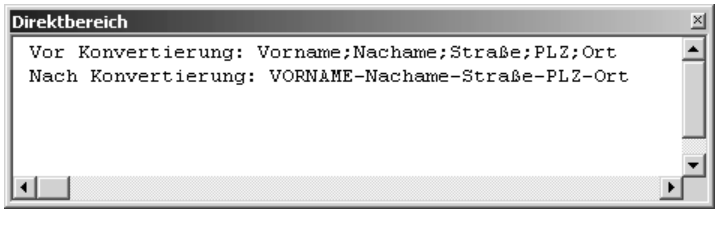


Abbildung 18: Text wurde zerlegt, aufbereitet und wieder zusammengesetzt.

23 In Groß- und Kleinschreibung umwandeln

Für die Konvertierung von Texten in die Groß- bzw. Kleinschreibung stehen Ihnen die folgenden beiden Funktionen zur Verfügung.

Die Funktion `LCase` wandelt Großbuchstaben in Kleinbuchstaben um. Die Syntax dieser Funktion lautet:

```
LCase(Zeichenfolge)
```

Die Funktion `UCase` wandelt Kleinbuchstaben in Großbuchstaben um. Die Syntax dieser Funktion lautet:

```
UCase(Zeichenfolge)
```

Das erforderliche Argument `Zeichenfolge` ist ein beliebiger gültiger Zeichenfolgenausdruck.

Im folgenden Beispiel aus Listing 30 wird ein Text in Kleinbuchstaben konvertiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub TextKleinMachen()
   Dim strText As String
   strText = "TEST"
   MsgBox LCase(strText)
End Sub
```

Listing 30: Text in Kleinbuchstaben umwandeln

Eine weitere Möglichkeit, Texte in Groß- bzw. Kleinbuchstaben umzuwandeln, bietet die Funktion `StrConv`. Diese Funktion gibt einen Wert vom Typ `Variant` (`String`) zurück, der wie angegeben umgewandelt wurde. Die Syntax der Funktion sieht wie folgt aus:

```
StrConv(string, conversion, LCID)
```

Die Syntax der `StrConv`-Funktion besteht aus folgenden benannten Argumenten:

Teil	Beschreibung
`String`	Erforderlich. Zeichenfolgenausdruck, der umgewandelt werden soll.
`Conversion`	Erforderlich. Wert vom Typ Integer. Die Summe der Werte, die den Typ der auszuführenden Umwandlung angibt.
`LCID`	Optional. Die Gebietsschema-ID, wenn diese sich von der des Systems unterscheidet. (Die Gebietsschema-ID des Systems ist die Voreinstellung.)

Tabelle 18: Die Argumente der Funktion StrConv

Konstante	Wert	Beschreibung
vbUpperCase	1	Wandelt die Zeichenfolge in Großbuchstaben um.
VbLowerCase	2	Wandelt die Zeichenfolge in Kleinbuchstaben um.
VbProperCase	3	Wandelt den ersten Buchstaben jedes Wortes innerhalb der Zeichenfolge in einen Großbuchstaben um.
vbWide*	4	Wandelt schmale (Einzel-Byte) Zeichen innerhalb der Zeichenfolge in breite (Doppel-Byte) Zeichen um.
vbNarrow	8	Wandelt breite (Doppel-Byte) Zeichen innerhalb der Zeichenfolge in schmale (Einzel-Byte) Zeichen um.
vbKatakana	16	Wandelt Hiragana-Zeichen innerhalb der Zeichenfolge in Katakana-Zeichen um.
vbHiragana	32	Wandelt Katakana-Zeichen innerhalb der Zeichenfolge in Hiragana-Zeichen um.
vbUnicode	64	Wandelt die Zeichenfolge in Unicode um und verwendet dabei die Voreinstellungen aus der Zeichenumsetzungstabelle des Systems (nicht verfügbar auf dem Macintosh).
VbFromUnicode	128	Wandelt die Zeichenfolge von Unicode in die Voreinstellungen aus der Zeichenumsetzungstabelle des Systems um (nicht verfügbar auf dem Macintosh).

Tabelle 19: Einstellungen für das Argument conversion

Im folgenden Beispiel aus Listing 31 wird jeweils der erste Buchstaben eines jeden neuen Wortes in Großbuchstaben umgewandelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub KonvertierungText()
   Dim strText As String

   strText = "access codebook"
   MsgBox StrConv(strText, vbProperCase)
End Sub
```

Listing 31: Texte schnell konvertieren über die Funktion StrConv

Übergeben Sie der Funktion `StrConv` den String aus der Variablen `StrText` und geben Sie als Argument die Konstante `vbProperCase` an, um jeweils den ersten Buchstaben eines Wortes groß auszugeben.

24 Access-Version feststellen

Um die im Einsatz befindliche Access-Version festzustellen, können Sie die Eigenschaft `Version` des Objekts `Application` abfragen. Diese Eigenschaft liefert die Versionsnummer, die Sie auswerten können. So liefert die Funktion beispielsweise folgende Werte:

Access-95: 7

Access-97: 8

Access 2000: 9

Access 2002: 10.0

Access 2003: 11.0

Über die Funktion `Left` können Sie jetzt die erste Ziffer der Versionsnummer prüfen. Wenn diese beispielsweise den Wert 1 zurückliefert, dann haben Sie eine neuere Version im Einsatz.

Die Funktion `Left` gibt einen Wert vom Typ `Variant (String)` zurück, der eine bestimmte Anzahl von Zeichen ab dem ersten (linken) Zeichen einer Zeichenfolge enthält. Die Syntax dieser Funktion lautet:

```
Left(string, length)
```

Die `Left`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
`String`	Erforderlich. Zeichenfolgenausdruck, aus dem die ersten (linken) Zeichen zurückgegeben werden. Wenn `string` den Wert Null enthält, wird Null zurückgegeben.
`Length`	Erforderlich; Wert vom Typ `Variant (Long)`. Numerischer Ausdruck, der die Anzahl der zurückzugebenden Zeichen angibt. Der Wert 0 führt zur Rückgabe einer Null-Zeichenfolge (""). Ist `length` größer oder gleich der Zeichenanzahl in `string`, so wird die gesamte Zeichenfolge zurückgegeben.

Tabelle 20: Die Argumente der Funktion Left

Im folgenden Beispiel aus Listing 32 wird eine Versionsprüfung in Access durchgeführt. Manchmal bereiten gerade ältere Access-Versionen bei der Programmierung Kummer, weil bestimmte VBA-Befehle noch nicht im Befehlswortschatz sind und daher Makroabstürze verursachen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub AccessVersionErmitteln()

   MsgBox Application.Version

   Select Case Left(Application.Version, 1)
      Case "5"
         MsgBox "Sie haben Access 5 im Einsatz"
      Case "7"
         MsgBox "Sie haben Access 95 im Einsatz"
      Case "8"
         MsgBox "Sie haben Access 97 im Einsatz"
      Case "9"
         MsgBox "Sie haben Access 2000 im Einsatz"
      Case Else
         MsgBox "Access 2002 und höher!"
   End Select
End Sub
```

Listing 32: Die Access-Version ermitteln und auswerten

In einer `Select Case`-Anweisung werten Sie die Eigenschaft `Version` aus. Diese Eigenschaft gibt eine Nummer zurück, über die Sie die Access-Version identifizieren können. Dabei wird im Beispiel aus Listing 32 nur die erste Ziffer der Versionsnummer über die Funktion `Left` extrahiert. Access-Versionen ab Access 2002, die die Versionsnummern 10 und 11 haben, werden dann automatisch über den `Case Else`-Zweig bedient.

25 Text Zeichen für Zeichen zerlegen

Möchten Sie einen Text Zeichen für Zeichen zerlegen, dann können Sie für diese Aufgabe die Funktion `Mid` einsetzen. Diese Funktion gibt einen Wert vom Typ `Variant` (`String`) zurück, der eine bestimmte Anzahl von Zeichen aus einer Zeichenfolge enthält. Die Syntax der Funktion lautet:

```
Mid(string, start[, length])
```

Die Syntax der `Mid`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
string	Erforderlich. Zeichenfolgenausdruck, aus dem Zeichen zurückgegeben werden. Wenn string den Wert Null enthält, wird Null zurückgegeben.
start	Erforderlich. Wert vom Typ Long. Position in string, an der die zurückzugebende Zeichenfolge beginnt. Ist start größer als die Anzahl der Zeichen in string, so gibt Mid eine leere Zeichenfolge ("") zurück.
length	Optional. Wert vom Typ Variant (Long). Anzahl der zurückzugebenden Zeichen. Wird length nicht angegeben oder befinden sich weniger Zeichen im Text (das Zeichen an der Stelle start eingeschlossen), als durch length angegeben, so werden alle Zeichen ab start bis zum Ende der Zeichenfolge zurückgegeben.

Tabelle 21: Die Argumente der Funktion Mid

Im nächsten Beispiel aus Listing 33 wird der Text Buchstabe für Buchstabe zerlegt und in einzelnen Zeilen im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub TextZerlegen()
   Dim intZ As Integer
   Dim StrText As String

   StrText = "Access-Codebook"
   For intZ = 1 To Len(StrText)
      Debug.Print Mid(StrText, intZ, 1)
   Next intZ
End Sub
```

Listing 33: Einen Text Zeichen für Zeichen zerlegen und ausgeben

In einer Schleife arbeiten Sie sich zeichenweise durch den Vorgabetext durch. Dabei ermitteln Sie über die Funktion Len zuerst einmal die Gesamtlänge des Vorgabetextes in der Variablen strText. Diese Gesamtlänge wird als Endekriterium für die Schleife verwendet. Innerhalb der Schleife zerlegen Sie den Text Zeichen für Zeichen mithilfe der Funktion Mid und geben die einzelnen Buchstaben im Direktfenster der Entwicklungsumgebung aus.

26 Dateiendung prüfen

Soll die Endung einer Datei überprüft werden, dann können Sie so vorgehen, dass Sie vom Gesamtnamen, beispielsweise *Funktionen.mdb* die letzten drei Zeichen extrahieren und auswerten.

Abbildung 19: Text zerlegen und ausgeben

Diese Aufgabe können Sie über die Funktion `Right` erledigen. Diese Funktion gibt einen Wert vom Typ `Variant (String)` zurück, der eine bestimmte Anzahl von Zeichen von der rechten Seite (dem Ende) einer Zeichenfolge enthält. Die Syntax der Funktion lautet:

```
Right(string, length)
```

Die Syntax der `Right`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
`string`	Erforderlich. Zeichenfolgenausdruck, von dem die letzten (rechtsstehenden) Zeichen zurückgegeben werden. Wenn `string` den Wert Null enthält, wird Null zurückgegeben.
`length`	Erforderlich. Wert vom Typ `Variant (Long)`. Numerischer Ausdruck, der die Anzahl der zurückzugebenden Zeichen angibt. Der Wert 0 führt zur Rückgabe einer Null-Zeichenfolge (""). Ist `Length` größer oder gleich der Zeichenanzahl in `String`, so wird die gesamte Zeichenfolge zurückgegeben.

Tabelle 22: Die Argumente der Funktion Right

Im folgenden Beispiel aus Listing 34 soll ein Dateinamen in eine Inputbox eingegeben werden. Danach wird geprüft, ob es sich um einen Access-Dateinamen handelt. Diese Information können Sie an der Dateierweiterung *mdb* sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================
```

Listing 34: Den Dateityp über die Endung prüfen

```
Sub DateiTypChecken()
   Dim strDatei As String

   strDatei = InputBox("Bitte Dateinamen eingeben", "Dateinamen:")

   If Len(strDatei) <> 0 Then
      If UCase(Right(strDatei, 3)) = "MDB" Then
         MsgBox "Es handelt sich um eine Access-Datei!"
      Else
         MsgBox "Keine Access-Datenbank!"
      End If
   End If
End Sub
```

Listing 34: Den Dateityp über die Endung prüfen (Forts.)

Schneiden Sie am Dateinamen die letzten drei Zeichen über die Funktion `Right` ab und werten Sie diese Information dann aus.

27 Zeichen ersetzen

Sollen einzelne Zeichen bzw. Zeichenfolgen ausgetauscht werden, dann setzen Sie für diese Aufgabe die Funktion `Replace` ein.

Die Funktion `Replace` gibt eine Zeichenfolge zurück, in der eine festgelegte, untergeordnete Zeichenfolge mit einer festgelegten Häufigkeit durch eine andere untergeordnete Zeichenfolge ersetzt wurde. Die Syntax dieser Funktion lautet:

```
Replace(expression, find, replace[, start[, count[, compare]]])
```

Die Syntax der `Replace`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
`Expression`	Erforderlich. Zeichenfolgenausdruck, der die zu ersetzende, untergeordnete Zeichenfolge enthält.
`Find`	Erforderlich. Die untergeordnete Zeichenfolge, nach der gesucht wird.
`Replace`	Erforderlich. Die untergeordnete Ersatzzeichenfolge.
`Start`	Optional. Position in `expression`, an der die Suche nach der untergeordneten Zeichenfolge beginnt. Wird diese Angabe ausgelassen, wird bei 1 begonnen.
`Count`	Optional. Anzahl der durchzuführenden Ersetzungen der untergeordneten Zeichenfolge. Wird diese Angabe ausgelassen, ist die Standardeinstellung –1, d.h., alle möglichen Zeichenfolgen werden ersetzt.
`Compare`	Optional. Numerischer Wert, der die Art des Vergleichs angibt, der beim Beurteilen von untergeordneten Zeichenketten verwendet werden soll.

Tabelle 23: Die Argumente der Funktion Replace

Das Argument `compare` kann folgende Werte haben:

Konstante	Wert	Beschreibung
`vbUseCompareOption`	`-1`	Führt einen Vergleich unter Verwendung der Option Compare-Anweisung durch.
`vbBinaryCompare`	`0`	Führt einen binären Vergleich durch.
`vbTextCompare`	`1`	Führt einen Textvergleich durch.
`vbDatabaseCompare`	`2`	Nur Microsoft Access. Führt einen Vergleich anhand der Informationen in Ihrer Datenbank durch.

Tabelle 24: Die möglichen Vergleichskonstanten

Im folgenden Beispiel aus Listing 35 wird ein Satz mit Umlauten ausgetauscht. Dabei sollen folgende Regeln gelten:

Aus ä wird ae.

Aus ö wird oe.

Aus ü wird ue.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub UmlauteErsetzen()
   Dim strText As String

   strText = "Sie können sich darüber ärgern oder nicht!"
   strText = Replace(strText, "ü", "ue")
   strText = Replace(strText, "ä", "ae")
   strText = Replace(strText, "ö", "oe")
   MsgBox strText
End Sub
```

Listing 35: Umlaute ersetzen

Übergeben Sie der Funktion nacheinander zuerst den ursprünglichen Text, der in der String-Variablen `StrText` steht. Als zweites Argument geben Sie die Zeichenfolge an, die im ersten Argument gesucht werden soll. Das dritte Argument beinhaltet die Zeichenfolge, die die Zeichenfolge aus dem zweiten Argument ersetzen soll.

28 Zeichenfolge mit Leerzeichen auffüllen

Um Leerzeichen in eine bestehende Zeichenfolge aufzunehmen, können Sie mit der Funktion `Space` arbeiten.

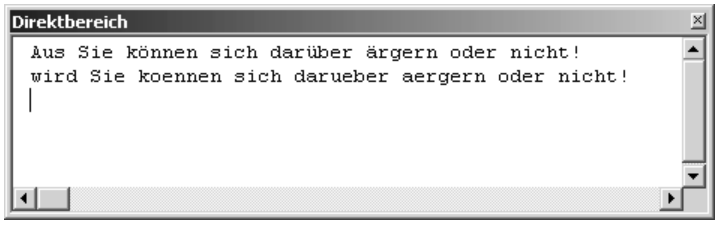


Abbildung 20: Zeichenfolgen schnell austauschen

Die Funktion `Space` gibt eine Zeichenfolge vom Typ `Variant (String)` zurück, die aus einer angegebenen Anzahl von Leerzeichen besteht. Die Syntax dieser Funktion lautet:

```
Space(Zahl)
```

Das erforderliche Argument `Zahl` entspricht der Anzahl der gewünschten Leerzeichen in der Zeichenfolge.

Im folgenden Beispiel aus Listing 36 wird ein Text mit Leerzeichenfolgen aufgefüllt. Dabei soll als Ergebnis die Textlänge immer aus zehn Zeichen bestehen. Liegen im Text also beispielsweise nur vier Zeichen vor, dann werden noch sechs Leerzeichen im Text integriert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub MitLeerzeichenFüllen()
   Dim strText As String

   strText = "0815"
   If Len(strText) < 10 Then
      strText = strText + Space(10 - Len(strText))
   End If
End Sub
```

Listing 36: Leerzeichen in eine Zeichenfolge pumpen

Im theoretischen Beispiel aus Listing 36 wird zunächst ein Text in einer Variablen vom Typ String gespeichert. Danach wird die Funktion `Len` eingesetzt, um die Länge des Textes zu ermitteln. Ergibt die Prüfung eine Länge kleiner als 10, werden über die Funktion `Space` die restlichen Zeichen mit Leerzeichen aufgefüllt.

29 Zahlen in Texte umwandeln

Die Funktion `Str` gibt einen Wert vom Typ `Variant (String)` zurück, der eine Zahl darstellt. Die Syntax dieser Funktion lautet:

```
Str(Zahl)
```

Das erforderliche Argument `Zahl` ist ein Wert vom Typ `Long`, der einen beliebigen numerischen Ausdruck enthält. Beim Konvertieren von Zahlen in Texte wird ein führendes Leerzeichen für das Vorzeichen der Zahl reserviert. Ist die Zahl positiv, so enthält der zurückgegebene Text ein führendes Leerzeichen.

Im folgenden Beispiel aus Listing 37 wird eine Zahl in einen Textwert umgewandelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub ZahlInTextWandeln()
   Dim lzahl As Long

   lzahl = 70469

   Debug.Print Str(lzahl)
End Sub
```

Listing 37: Zahlenwert in einen Textwert umwandeln

30 Zeichenfolgen miteinander vergleichen

Sollen Zeichenfolgen miteinander verglichen werden, dann können Sie dazu die Funktion `StrComp` einsetzen. Die Syntax dieser Funktion lautet:

```
StrComp(string1, string2[, compare])
```

Teil	Beschreibung
`string1`	Erforderlich. Ein beliebiger gültiger Zeichenfolgenausdruck.
`string2`	Erforderlich. Ein beliebiger gültiger Zeichenfolgenausdruck.
`Compare`	Optional. Legt die Art des Zeichenfolgenvergleichs fest. Der Wert Null für `compare` führt zu einem Fehler. Die möglichen Vergleichstypen können Sie aus der folgenden Tabelle entnehmen.

Tabelle 25: Die Argumente der Funktion StrComp

Konstante	Wert	Beschreibung
`vbUseCompareOption`	`-1`	Führt einen Vergleich mithilfe der Option `Compare`-Anweisung durch.
`vbBinaryCompare`	`0`	Führt einen binären Vergleich durch.
`vbTextCompare`	`1`	Führt einen textbasierten Vergleich durch.

Tabelle 26: Die Vergleichstypen der Funktion StrComp

Konstante	Wert	Beschreibung
vbDatabaseCompare	2	Nur Microsoft Access. Führt einen Vergleich durch, der auf Informationen in Ihrer Datenbank basiert.

Tabelle 26: Die Vergleichstypen der Funktion StrComp (Forts.)

Die Funktion `StrComp` liefert folgende Rückgabewerte.

Fall	Rückgabewert
`string1` liegt im Alphabet vor `string2`	-1
`string1` entspricht `string2`	0
`string1` liegt im Alphabet hinter `string2`	1
`string1` oder `string2` ist Null	Null

Tabelle 27: Die Rückgabewerte der Funktion StrComp

Im folgenden Beispiel aus Listing 38 werden zwei Zeichenfolgen miteinander verglichen und überprüft, welche der beiden Zeichenfolgen im Alphabet weiter vorne angeordnet ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub ZeichenfolgenVergleichen()
   Dim strText1 As String
   Dim strText2 As String

   strText1 = "RTS"
   strText2 = "STS"

   If StrComp(strText1, strText2, 1) = -1 Then
      MsgBox "Text1 liegt vor Text2!"
   Else
      MsgBox "Text2 liegt vor Text1"
   End If
End Sub
```

Listing 38: Einen Textvergleich durchführen

Im Beispiel aus Listing 38 werden zwei Zeichenfolgen in String-Variablen gefüllt. Über die Funktion `StrComp` wird geprüft, welche der beiden Zeichenfolgen im Alphabet weiter vorne liegt. Der Rückgabewert –1 bedeutet, dass die erste Zeichenfolge m Alphabet weiter vorne liegt als die zweite Zeichenfolge.

31 Zeichenfolgen spiegeln

Über die Funktion `StrReverse` können Sie eine Zeichenfolge spiegeln. Dabei gibt die Funktion eine Zeichenfolge zurück, in der die Reihenfolge der Zeichen einer bestimmten Zeichenfolge umgekehrt wurde. Die Syntax der Funktion lautet:

```
StrReverse(expression)
```

Das Argument `expression` ist die Zeichenfolge, deren Zeichen umgekehrt werden sollen. Wenn es sich bei `expression` um eine Zeichenfolge der Länge Null ("") handelt, wird eine Zeichenfolge der Länge Null zurückgegeben. Wenn `expression` Null ist, kommt es zu einem Fehler.

Im folgenden Beispiel aus Listing 39 wird ein Text gespiegelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub ZeichenfolgeSpiegeln()
   Dim strText As String

   strText = "Esel"
   Debug.Print "Vorher: " & strText
   Debug.Print "Nachher: " & LCase(StrReverse(strText))
End Sub
```

Listing 39: Texte spiegeln mithilfe der Funktion StrReverse

Vor dem eigentlichen Drehen des Textes durch die Funktion `StrReverse` wenden Sie die Funktion `LCase` an, um die Zeichenfolge zusätzlich in Kleinbuchstaben zu konvertieren.

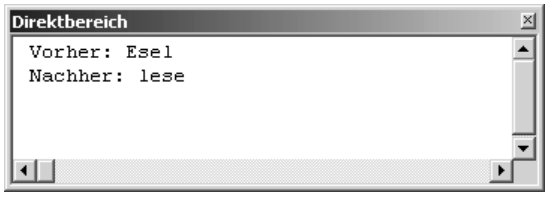


Abbildung 21: Texte drehen über die Funktion StrReverse.

32 Texte vervollständigen

Soll ein Text mit einem bestimmten Zeichen vervollständigt werden, dann können Sie diese Aufgabe über die Funktion `String` lösen. Diese Funktion gibt eine Zeichenfolge vom Typ `Variant (String)` zurück, die ein sich wiederholendes Zeichen der angegebenen Länge enthält. Die Syntax der Funktion lautet:

```
String(number, character)
```

Die Syntax der `String`-Funktion besteht aus folgenden benannten Argumenten:

Teil	Beschreibung
`Number`	Erforderlich. Wert vom Typ `Long`. Länge der zurückgegebenen Zeichenfolge. Wenn `number` den Wert Null enthält, wird Null zurückgegeben.
`Character`	Erforderlich. Wert vom Typ `Variant`. Ein Zeichencode, der ein Zeichen oder einen Zeichenfolgenausdruck angibt, dessen erstes Zeichen verwendet wird, um die zurückzugebende Zeichenfolge zu erstellen. Wenn `character` den Wert Null enthält, wird Null zurückgegeben.

Tabelle 28: Die Argumente der Funktion String

Im folgenden Beispiel aus Listing 40 wird die Länge eines Textes überprüft. Hat der Text eine Länge kleiner 10, dann werden die restlichen Zeichen durch einen Stern vervollständigt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=================================================

Sub MitSternAuffuellen()
   Dim StrText As String

   StrText = "Gesamt"
   If Len(StrText) < 10 Then
      StrText = _
      StrText & String(10 - Len(StrText), "*")
   End If
   MsgBox StrText
End Sub
```

Listing 40: Das Stern-Symbol als Füllzeichen verwenden

Füllen Sie zunächst die String-Variable mit einem Text Ihrer Wahl. Danach prüfen Sie über die Funktion `Len` die Länge der Zeichenfolge. Mithilfe der Funktion `String` füllen Sie dann die noch fehlenden Zeichen mit dem Zeichen *.

33 Zuordnungen treffen

Eine Art von Abfrage stellt die Funktion `Switch` dar. Diese Funktion wertet eine Liste von Ausdrücken aus und gibt einen Wert vom Typ `Variant` oder einen Ausdruck zurück, der dem ersten Ausdruck in der Liste zugeordnet ist, der `True` ergibt. Die Syntax der Funktion lautet:

```
Switch(Ausdr-1, Wert-1[, Ausdr-2, Wert-2 … [, Ausdr-n,Wert-n]])
```

Die Syntax der `Switch`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
Ausdr	Erforderlich. Variant-Ausdruck, der ausgewertet werden soll.
Wert	Erforderlich. Wert oder Ausdruck, der zurückgegeben werden soll, wenn der entsprechende Ausdruck True ergibt.

Tabelle 29: Die Argumente der Funktion Switch

Im folgenden Beispiel aus Listing 41 soll über eine Inputbox ein Land eingegeben werden. Über die Funktion Switch wird dann die dazugehörige Sprache des Landes ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub LandUndSprache()
   Dim StrLand as String

   strLand = InputBox("Bitte ein Land eingeben!")
   If Len(strLand) <> 0 Then
      On Error GoTo Fehler
      strLand = Switch _
         (strLand = "Deutschland", "Deutsch", _
          strLand = "Frankreich", "Französisch", _
          strLand = "Italien", "Italienisch", _
          strLand = "Spanien", "Spanisch")
       MsgBox "Sie sprechen " & strLand
       Exit Sub


      Fehler:
         MsgBox "Das Land ist nicht eingepflegt!"
   End If
End Sub
```

Listing 41: Zuordnung von Land zur Sprache vornehmen

Fragen Sie über die Methode Inputbox ein Land ab. Über die Funktion Len können Sie abfragen, ob überhaupt eine Eingabe vorgenommen wurde. Wenn nicht, dann gibt diese Funktion den Wert 0 zurück. Im anderen Fall weisen Sie mithilfe der Funktion Switch jedem Land die dazugehörige Sprache zu.

34 Führende Leerzeichen entfernen

Oft machen führende oder nachgestellte Leerzeichen Probleme bei der Programmierung, weil man sie unter anderem ja gar nicht sieht.

Die Funktion `Trim` gibt einen Wert vom Typ `Variant` (`String`) zurück, der eine Kopie einer bestimmten Zeichenfolge enthält, die keine führenden Leerzeichen (`LTrim`), keine nachgestellten Leerzeichen (`RTrim`) sowie keine Kombination aus führenden und nachgestellten Leerzeichen (`Trim`) enthält. Die Syntax dieser Funktionen lautet:

```
LTrim(Zeichenfolge)
RTrim(Zeichenfolge)
Trim(Zeichenfolge)
```

Das erforderliche Argument `Zeichenfolge` ist ein beliebiger gültiger Zeichenfolgenausdruck. Wenn `Zeichenfolge` den Wert Null enthält, wird Null zurückgegeben.

Im folgenden Beispiel aus Listing 42 werden aus einem Text alle führenden sowie nachfolgenden Leerzeichen entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Sub LeerzeichenEntfernen()
   Dim StrText As String
   StrText = " Das ist ein Test   "
   StrText = Trim(StrText)
End Sub
```

Listing 42: Leerzeichen am linken und rechten Rand entfernen

Müssen aus einer Zeichenfolge wirklich alle Leerzeichen raus, also auch in der Mitte, dann schreiben Sie eine Funktion, die beispielsweise wie in Listing 43 aussehen kann.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================
Function LeereRaus(s As String) As String
   Dim intZ As Integer

   For intZ = 1 To Len(s)
      Select Case Asc(Mid(s, intZ, 1))
         Case 32
            'hier handelt es sich um das Zeichen Leer
         Case Else
            LeereRaus = LeereRaus & Mid(s, intZ, 1)
      End Select
   Next intZ
End Function
```

Listing 43: Alle Leerzeichen aus einer Zeichenfolge eliminieren

```
Sub AlleLeerzeichenEntfernen()
   Dim StrText As String

   StrText = " Das ist ein Test   "
   StrText = LeereRaus(StrText)
   MsgBox StrText
End Sub
```

Listing 43: Alle Leerzeichen aus einer Zeichenfolge eliminieren (Forts.)

In der Funktion aus Listing 43 wird über eine `For Next`-Schleife Zeichen für Zeichen abgearbeitet. Dabei ermitteln Sie über die Funktion `Len` die Gesamtlänge des übergebenen Textes. Innerhalb der Schleife wandeln Sie in einer `Select Case`-Anweisung über die Funktion `ASC` den jeweiligen Buchstaben in einen Zahlenwert um, der dem Zeichencode entspricht. Dabei wird ein Leerzeichen über den numerischen Wert 32 dargestellt. Daher wird im ersten Zweig der `Case`-Anweisung nichts weiter gemacht. Im `Case Else`-Zweig werden die übrigen Zeichen wieder zu einem String zusammengesetzt und dem aufrufenden Makro als Rückgabewert übergeben.

35 Standardverzeichnis neu setzen

Mithilfe der Anweisung `ChDir` können Sie in das aktuelle Verzeichnis wechseln.

```
ChDir Pfad
```

Das erforderliche Argument `Pfad` ist ein Zeichenfolgenausdruck, der angibt, welches Verzeichnis zum neuen Standardverzeichnis wird.

Im folgenden Beispiel aus Listing 44 wird das Standardverzeichnis neu gesetzt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================
Sub VerzeichnisWechseln()
   On Error GoTo Fehler
   ChDir "C:\Windows\Temp"
   Exit Sub

   Fehler:
      MsgBox "Dieses Verzeichnis gibt es nicht!"
End Sub
```

Listing 44: Das Verzeichnis über ChDir neu einstellen

Übergeben Sie der Anweisung `ChDir` den kompletten Pfad inklusive des Laufwerks.

36 Laufwerkswechsel durchführen

Mithilfe der Anweisung `ChDrive` stellen Sie das aktuelle Laufwerk ein. Die Syntax für diese Funktion lautet:

```
ChDrive Laufwerk
```

Das erforderliche Argument `Laufwerk` ist ein Zeichenfolgenausdruck, der ein existierendes Laufwerk angibt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub LaufwerkWechseln()
   On Error GoTo Fehler
   ChDrive "d:"
   Exit Sub

   Fehler:
      MsgBox "Dieses Laufwerk gibt es nicht!"
End Sub
```

Listing 45: Laufwerkswechsel über die Anweisung ChDrive durchführen

Übergeben Sie der Funktion `ChDrive` den Laufwerksbuchstaben inklusive des Doppelpunkts.

Hinweis

In Kapitel 2 dieses Buchs können Sie nachschlagen, wie Sie ein Laufwerk bzw. ein Verzeichnis über den Einsatz einer API-Funktion, die einen Verzeichnisbaum zur Verfügung stellt, einsetzen können.

37 Aktuelles Verzeichnis auslesen

Über die Anweisung `CurDir` können Sie das aktuell eingestellte Verzeichnis ermitteln. Die Syntax dieser Funktion lautet:

```
CurDir[(Laufwerk)]
```

Das optionale Argument `Laufwerk` ist ein Zeichenfolgenausdruck, der ein existierendes Laufwerk angibt. Ist kein Laufwerk angegeben oder enthält das Laufwerk eine Null-Zeichenfolge (""), so gibt die `CurDir`-Funktion den Pfad des aktuellen Laufwerks zurück.

Im folgenden Beispiel aus Listing 46 wird das momentan eingestellte Verzeichnis am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub AktuellesVerzeichnisAusgeben()
   MsgBox "Das aktuelle Verzeichnis lautet: " & CurDir("C")
End Sub
```

Listing 46: Das aktuelle Verzeichnis abfragen

Über die Anweisung `CurDir` wird das aktuell eingestellte Verzeichnis ausgegeben. Übergeben Sie dieser Anweisung als Argument den Laufwerksbuchstaben, in dem das aktuelle Verzeichnis ermittelt werden soll.

38 Dateien und Verzeichnisse auslesen

Um die Dateien aus einem Verzeichnis auszulesen, können Sie die noch aus DOS-Tagen bekannte Funktion `Dir` einsetzen.

Die Funktion `Dir` gibt eine Zeichenfolge (`String`) zurück, die den Namen einer Datei, eines Verzeichnisses oder eines Ordners darstellt, der mit einem bestimmten Suchmuster, einem Dateiattribut oder mit der angegebenen Datenträger- bzw. Laufwerksbezeichnung übereinstimmt. Die Syntax der Funktion lautet:

```
Dir[(Pfadname[, Attribute])]
```

Die Syntax der `Dir`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
`Pfadname`	Optional. Zeichenfolgenausdruck, der einen Dateinamen angibt. Der Dateiname kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten. Eine Null-Zeichenfolge ("") wird zurückgegeben, wenn der Pfad aus `Pfadname` nicht gefunden werden kann
`Attribute`	Optional. Eine Konstante oder ein numerischer Ausdruck, deren Summe die Dateiattribute angibt. Wenn das Argument nicht angegeben wird, werden alle Dateien, die dem Argument `Pfadnamen` entsprechen, zurückgegeben.

Tabelle 30: Die Argumente der Funktion Dir

Das Argument `Attribute` hat die folgenden Einstellungen:

Konstante	Wert	Beschreibung
`VbNormal`	`0`	(Voreinstellung) Dateien ohne Attribute.

Tabelle 31: Die möglichen Einstellungen beim Argument Attribute

Konstante	Wert	Beschreibung
VbReadOnly	1	Schreibgeschützte Dateien, zusätzlich zu Dateien ohne Attribute.
VbHidden	2	Versteckte Dateien, zusätzlich zu Dateien ohne Attribute.
VbSystem	4	Systemdatei, zusätzlich zu Dateien ohne Attribute. Beim Macintosh nicht verfügbar.
VbVolume	8	Datenträgerbezeichnung. Falls andere Attribute angegeben wurden, wird vbVolume ignoriert. Beim Macintosh nicht verfügbar.
VbDirectory	16	Verzeichnis oder Ordner, zusätzlich zu Dateien ohne Attribute.

Tabelle 31: Die möglichen Einstellungen beim Argument Attribute (Forts.)

Im folgenden Beispiel aus Listing 47 werden aus einem Verzeichnis alle Dateien ausgelesen und das Datum der Erstellung einer jeden Datei wird protokolliert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=================================================

Sub DateienDokumentieren()
   Dim strAttr As String
   Dim strPfad As String
   Dim strName As String
   Dim intz As Integer

   strAttr = vbNormal + vbReadOnly + vbHidden
   strPfad = "c:\Eigene Dateien\"
   strName = Dir(strPfad, strAttr)

   On Error Resume Next
   Do While strName <> ""
      If strName <> "." And strName <> ".." Then
         Debug.Print strName; vbTab & FileDateTime(strPfad & strName)
         intz = intz + 1
      End If
      strName = Dir
   Loop
End Sub
```

Listing 47: Dateien auflisten und dokumentieren

Stellen Sie zuerst die gewünschten Dateiattribute über die Konstanten der Dir-Funktion zusammen und speichern Sie diese in der Variablen StrAttr. Danach geben Sie bekannt, in welchem Verzeichnis die Suche stattfinden soll. Über die Funktion Dir starten Sie die Suche,

indem Sie die beiden Argumente `StrPfad` und `StrAttr` übergeben. Diese Suche wird so lange in einer Schleife ausgeführt, bis alle Dateien des angegebenen Verzeichnisses ausgelesen wurden. Innerhalb der Schleife wenden Sie die Funktion `FileDateTime` an, um das Erstellungsdatum der Datei zu ermitteln.

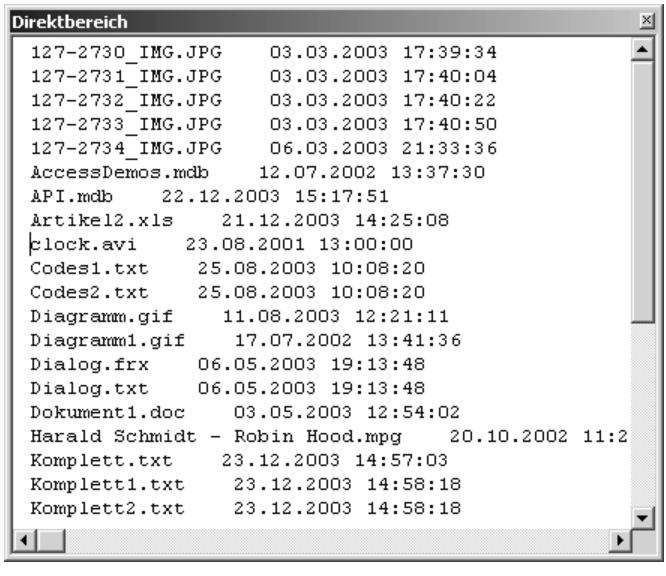


Abbildung 22: Dateidokumentation im Direktfenster der Entwicklungsumgebung

39 Verzeichnis anlegen

Mithilfe der Anweisung `MkDir` können Sie ein neues Verzeichnis erstellen. Die Syntax dieser Funktion lautet:

```
MkDir Pfad
```

Das erforderliche Argument `Pfad` ist ein Zeichenfolgenausdruck, der das zu erstellende Verzeichnis angibt. `Pfad` kann das Laufwerk enthalten. Wenn kein Laufwerk angegeben ist, erstellt die `MkDir`-Anweisung ein neues Verzeichnis oder einen neuen Ordner auf Basis des aktuellen Laufwerks.

Im folgenden Beispiel aus Listing 48 wird ein neues Verzeichnis angelegt. Im Falle, dass dieses Verzeichnis bereits existiert, fangen Sie den Fehler über die `On Error`-Klausel ab.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================
```

Listing 48: Neues Verzeichnis über MkDir anlegen

```
Sub VerzeichnisAnlegen()
   On Error Resume Next
   MkDir "C:\Eigene Dateien\Versuch"
End Sub
```

Listing 48: Neues Verzeichnis über MkDir anlegen (Forts.)

Übergeben Sie der Anweisung `MkDir` den kompletten Pfad zum Verzeichnis, welches Sie erstellen möchten. Wenn kein Laufwerk angegeben ist, erstellt die Anweisung ein neues Verzeichnis oder einen neuen Ordner auf dem aktuellen Laufwerk.

40 Verzeichnis entfernen

Über die Anweisung `RmDir` können Sie ein Verzeichnis löschen. Die Syntax dieser Funktion lautet:

```
RmDir Pfad
```

Das erforderliche Argument `Pfad` ist ein Zeichenfolgenausdruck, der das zu löschende Verzeichnis oder den zu löschenden Ordner angibt. `Pfad` kann das Laufwerk beinhalten. Wenn kein Laufwerk angegeben ist, löscht `RmDir` das Verzeichnis oder den Ordner auf dem aktuellen Laufwerk.

Hinweis

`RmDir` führt zu einem Fehler, wenn Sie die Anweisung für ein Verzeichnis ausführen, in dem Dateien enthalten sind. Löschen Sie zuerst alle Dateien mit der `Kill`-Anweisung, bevor Sie ein Verzeichnis oder einen Ordner entfernen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub VerzeichnisEntfernen()
   On Error Resume Next
   RmDir "C:\Test"
End Sub
```

Listing 49: Ein Verzeichnis über die Anweisung RmDir entfernen

Übergeben Sie der Anweisung `RmDir` den kompletten Pfad zum Verzeichnis, welches Sie löschen möchten.

41 Datei löschen

Die Anweisung `Kill` löscht eine Datei ohne Rückfrage von der Festplatte. Die Syntax dieser Funktion lautet:

```
Kill Pfadname
```

Das erforderliche Argument `Pfadname` ist ein Zeichenfolgenausdruck, der eine oder mehrere zu löschende Dateien angibt. `Pfadname` kann ein Verzeichnis sowie ein Laufwerk enthalten.

Im folgenden Beispiel aus Listing 50 wird eine Datei aus einem bestimmten Verzeichnis entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub DateiEntfernen()
   On Error Resume Next
   Kill "C:\Eigene Dateien\Artikel.txt"
End Sub
```

Listing 50: Datei löschen über die Anweisung Kill

Übergeben Sie der Anweisung `Kill` den Namen der Datei inklusive des Pfads, in dem die Datei gespeichert ist. Sollte die Datei in diesem Verzeichnis nicht enthalten sein, dann sorgt die `On Error`-Klausel dafür, dass es zu keiner Fehlermeldung kommt. Die Anweisung wird in diesem Fall einfach ignoriert.

42 Datei kopieren

Mithilfe der Anweisung `FileCopy` können Sie Dateien kopieren. Die Syntax dieser Funktion lautet:

```
FileCopy source, destination
```

Die Syntax der `FileCopy`-Anweisung verwendet die folgenden benannten Argumente:

Teil	Beschreibung
`Source`	Erforderlich. Zeichenfolgenausdruck, der den Namen der zu kopierenden Datei angibt. `Source` kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten.
`Destination`	Erforderlich. Zeichenfolgenausdruck, der den Namen der Zieldatei angibt. `Destination` kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten.

Tabelle 32: Die Argumente der Anweisung FileCopy

Im folgenden Beispiel aus Listing 51 wird eine Textdatei kopiert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=================================================

Sub KopierenDatei()
   Dim strDat As String
   Dim strDat_Kopie As String

   strDat = "C:\Artikel.txt"
   strDat_Kopie = "Kopie.txt"

   On Error GoTo fehler
   FileCopy strDat, strDat_Kopie
   Exit Sub

   fehler:
      MsgBox "Fehler beim Kopieren aufgetreten!"
End Sub
```

Listing 51: Dateien kopieren über die Anweisung FileCopy

Übergeben Sie der Anweisung `FileCopy` den Namen der Originaldatei sowie den Namen der neuen Datei, die durch diese Anweisung erstellt wird. Leider können mit dieser Anweisung keine geöffneten Dateien kopiert werden.

Bilder neu benennen und nummerieren

Auf vielen PCs gibt es Verzeichnisse, die eine große Anzahl von Bildern bzw. Fotos enthalten. Oft werden diese Bilder aus dem Internet herauskopiert oder selbst mit der digitalen Kamera fotografiert. Die Namen der einzelnen Bilder liegen häufig in unterschiedlicher Form und nicht geordnet vor. Die folgende Lösung beschreibt, wie Sie Ihre Bilder im Nachhinein gleichartig benennen und durchnummerieren können.

Als Ausgangssituation liegt ein Verzeichnis mit Fotos bzw. Bildern im jpg-Format vor. Diese Dateien sollen einheitlich benannt und durchnummeriert werden.

Um die Bilder aus Abbildung 23 umzubenennen und zu nummerieren, starten Sie das Makro aus Listing 52.

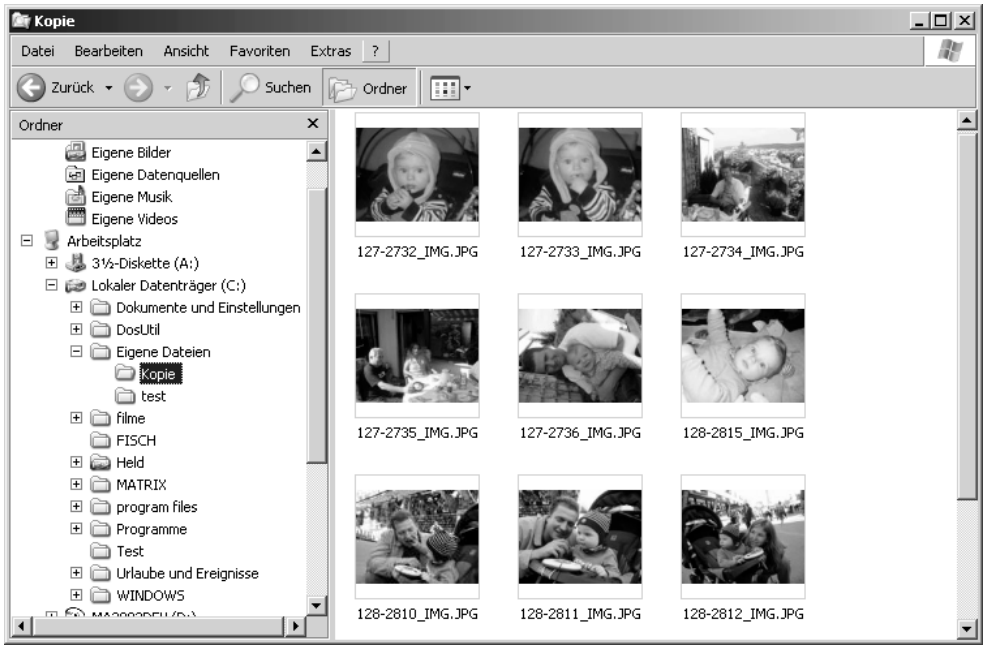


Abbildung 23: Diese Dateien sollen umbenannt und durchnummeriert werden.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'====================================================

Sub BilderBenennenUndNummerieren()
   Dim strDateiname As String
   Dim intz As Integer
   On Error Goto fehler
   intz = 1
   ChDir "C:\Eigene Dateien\Kopie\"
   strDateiname = Dir$("c:\Eigene Dateien\Kopie\*.jpg")

   Do Until strDateiname = ""
      FileCopy strDateiname, "Pic" & Format(intz, "000") & ".jpg"
      Kill strDateiname
      strDateiname = Dir$()
      If InStr(strDateiname, "Pic") <> 0 Then Exit Sub
      intz = intz + 1
   Loop
   Exit Sub
   fehler:
      Msgbox Err.Number & " " & Err.Description
End Sub
```

Listing 52: Dateien umbenennen und durchnummerieren

Zu Beginn des Makros werden zwei Variablen deklariert. In der Variablen `strDateiname` werden im weiteren Verlauf des Makros die Namen der einzelnen Bilder zwischengespeichert. In der Zählvariablen `intz` wird die Nummerierung der Bilder festgelegt. Der Variablen `intz` wird ein Startwert zugewiesen, bei dem die Nummerierung der Bilder beginnen soll. Über die Anweisung `ChDir` wechselt man in das gewünschte Verzeichnis, in dem die Bilder gespeichert sind. Danach wird über die Funktion `Dir` das erste Bild gesucht und in der Variablen `Dateiname` abgelegt. In einer anschließenden Schleife werden alle Bilder, die im Verzeichnis stehen, abgearbeitet.

Mithilfe der Anweisung `FileCopy` wird das jeweils gefundene Bild kopiert und unter einem neuen Dateinamen gespeichert. Dazu gibt man im ersten Argument der Anweisung `FileCopy` den Namen des Bilds an, welches man kopieren möchte. Diese Information steht direkt in der Variablen `Dateinamen` zur Verfügung. Im zweiten Argument wird der neue Name für das neue Bild angegeben. Dieser neue Name wird durch den Text »Pic« sowie den fortlaufenden Dateizähler gebildet. Mithilfe der Funktion `Format` wird die Art der Nummerierung festgelegt. Dabei ist die Nummerierung des Bilds immer dreistellig (*Pic001.jpg*, *Pic002.jpg*).

Innerhalb der Schleife wird die Zählvariable `intz` bei jedem Schleifendurchlauf um den Wert 1 erhöht. Damit es zu keiner Endlosschleife kommt, muss man über die Funktion `Instr` prüfen, wann die Funktion `Dir` auf die erste, neue Kopie eines Bilds stößt. Dabei kann man abfragen, ob in der Variablen `Dateiname` der Text `Pic` vorkommt. In diesem Fall wird ein Wert <> 0 zurückgegeben. Ist dies der Fall, dann wird über die Anweisung `Exit Sub` das Makro sofort beendet. Bevor am Ende der Schleife das jeweils nächste Bild über die Funktion `Dir` gesucht wird, wird mithilfe der Anweisung `Kill` das alte Bild von der Festplatte, ohne weitere Rückfrage, entfernt.

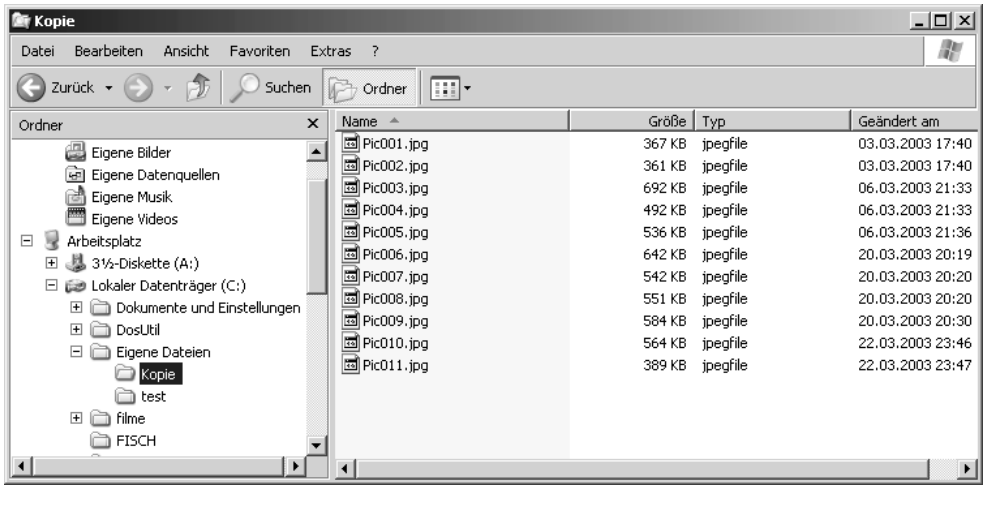


Abbildung 24: Die Bilder wurden neu einsortiert.

43 Datei umbenennen

Über die Anweisung `Name` können Sie eine Datei umbenennen. Die Syntax dieser Funktion lautet:

```
Name AlterPfadname As NeuerPfadname
```

Die Syntax der `Name`-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
`AlterPfadname`	Erforderlich. Zeichenfolgenausdruck, der einen existierenden Dateinamen und seinen Pfad angibt. In dem Wert kann ein Verzeichnis oder Ordner sowie ein Laufwerk enthalten sein.
`NeuerPfadname`	Erforderlich. Zeichenfolgenausdruck, der einen neuen Dateinamen und seinen Pfad angibt. In dem Wert kann ein Verzeichnis oder Ordner sowie ein Laufwerk enthalten sein. Die in `NeuerPfadname` angegebene Datei darf noch nicht existieren.

Tabelle 33: Die Argumente der Anweisung Name

Im folgenden Beispiel aus Listing 53 wird eine Textdatei umbenannt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================
Sub DateiUmbenennen()
   Dim strAlterName As String
   Dim strNeuername As String

   On Error GotTo fehler
   strAlterName = "C:\Artikel1.txt"
   strNeuername = "c:\ArtikelX.txt"

   Name strAlterName As strNeuername
   Exit Sub

   fehler:
      Msgbox Err.Number & " " & Err.Description
End Sub
```

Listing 53: Datei neu benennen über die Anweisung Name

Im Makro aus Listing 53 wird der Name der alten sowie der neue Name der Datei in String-Variablen zwischengespeichert. Danach wenden Sie die Anweisung `Name` an, der Sie die beiden Strings als Argumente übergeben.

44 Dateiexistenz prüfen

Möchten Sie prüfen, ob eine Datei überhaupt existiert, dann können Sie die Funktion `FileLen` einsetzen.

Die Funktion `FileLen` gibt einen Wert vom Typ `Long` zurück, der die Länge einer Datei in Byte angibt. Die Syntax dieser Funktion lautet:

```
FileLen(Pfadname)
```

Das erforderliche Argument `Pfadname` ist ein Zeichenfolgenausdruck, der eine Datei angibt. `Pfadname` kann ein Verzeichnis oder ein Laufwerk enthalten.

Im folgenden Beispiel aus Listing 54 wird die Funktion `FileLen` eingesetzt, um zu prüfen, ob eine bestimmte Datei überhaupt existiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub DateiExistenz()
   Dim intL As Integer

   On Error GoTo fehler
   intL = FileLen("C:\Artikel1.txt")
   'Weitere Aktionen

   Exit Sub


   fehler:
      If Err.Number = 53 Then
         MsgBox "Datei existiert nicht!"
      Else
         MsgBox Err.Number & " " & Err.Description
      End If
End Sub
```

Listing 54: Dateiexistenz prüfen

Übergeben Sie der Funktion `FileLen` den Namen inklusive der Pfadangaben. Als Rückgabewert erhalten Sie einen Fehler, wenn die Datei nicht existiert. Über die Fehlernummer, die Sie über das Fehlerobjekt `Err` auslesen können, lässt sich mithilfe der Eigenschaft `Number` der Fehler näher auswerten. Die Fehlernummer 53 steht dafür, dass eine Datei nicht gefunden werden kann.

45 Betriebssystem-Umgebungsvariablen auslesen

Die Anweisung `Environ` gibt die mit einer Betriebssystem-Umgebungsvariablen verbundene Zeichenfolge (`String`) zurück.

```
Environ({envstring | number})
```

Die Syntax der `Environ`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
`Envstring`	Optional. Ein Zeichenfolgenausdruck, der den Namen einer Umgebungsvariablen enthält.
`Number`	Optional. Ein numerischer Ausdruck entsprechend der numerischen Reihenfolge der Umgebungszeichenfolge in der Tabelle für Umgebungszeichenfolgen. Das Argument `number` kann ein beliebiger numerischer Ausdruck sein, der aber vor der Auswertung auf eine ganze Zahl gerundet wird.

Tabelle 34: Die Argumente der Anweisung Environ

Im folgenden Beispiel aus Listing 55 werden alle Umgebungsvariablen des Systems am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub BetriebssystemAuslesen()
   Dim intZ As Integer
   Dim strText As String

   intZ = 1
   Do Until Environ(intZ) = ""
      strText = strText & vbLf & Environ(intZ)
```

Listing 55: Die Betriebssystemumgebung auslesen

```
      intZ = intZ + 1
   Loop
   MsgBox strText
End Sub
```

Listing 55: Die Betriebssystemumgebung auslesen (Forts.)

In einer Schleife übergeben Sie der Funktion `Environ` numerische Werte. Die zurückgegebenen Umgebungsvariablen sammeln Sie in der String-Variablen `StrText`, die Sie nach dem Schleifenauftritt am Bildschirm ausgeben.

Abbildung 25: Die Betriebssystemumgebung checken

Sie können die einzelnen Betriebssystemumgebungsvariablen auch direkt ansprechen. Im Makro aus Listing 56 wird der Benutzernamen sowie der Computernamen am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
```

Listing 56: Anwendernamen sowie Computernamen auslesen

```
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub UsernameAuslesen()
   MsgBox Environ("Username") & vbLf & Environ("Computername")
End Sub
```

Listing 56: Anwendernamen sowie Computernamen auslesen (Forts.)

Hinweis

Im zweiten Kapitel diese Buchs können Sie nachschlagen, wie Sie den Anwendernamen sowie den Computernamen über eine API-Funktion ermitteln können.

46 Textdatei lesen

Zum Einlesen von Textdateien stehen Ihnen in VBA diverse Funktionen zur Verfügung. Die folgenden Beispiele werden anhand der Textdatei *Artikel.txt* beschrieben, die Sie in Abbildung 26 sehen können.

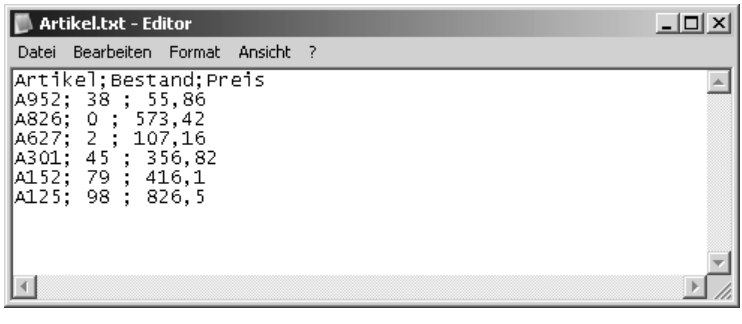


Abbildung 26: Die Ausgangsdatei

Über die Anweisung `Open` können Sie eine Datei für die Ein- bzw. Ausgabe öffnen. Die Syntax lautet:

```
Open Pfadname For Modus [Access Zugriff] [Sperre] As [#]Dateinummer [Len=Satzlänge]
```

Die Syntax der `Open`-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
`Pfadname`	Erforderlich. Zeichenfolgenausdruck, der einen Dateinamen festlegt und auch Verzeichnis-, Ordner- sowie Laufwerksangaben enthalten kann.

Tabelle 35: Die Argumente der Anweisung Open

Teil	Beschreibung
Modus	Erforderlich. Schlüsselwort, das den Zugriffsmodus für die Datei festlegt: Append, Binary, Input, Output oder Random. Wenn kein Modus festgelegt ist, wird die Datei im Zugriffsmodus Random geöffnet.
Zugriff	Optional. Schlüsselwort, das die Operationen festlegt, die auf der geöffneten Datei ausgeführt werden können: Read, Write oder Read Write.
Sperre	Optional. Schlüsselwort, das die Operationen festlegt, die von anderen Prozessen auf der geöffneten Datei ausgeführt werden können: Shared, Lock Read, Lock Write und Lock Read Write.
Dateinummer	Erforderlich. Eine gültige Dateinummer im Bereich von 1 bis 511 (einschließlich). Mit der FreeFile-Funktion erhalten Sie die nächste verfügbare Dateinummer.
Satzlänge	Optional. Zahl kleiner oder gleich 32767 (Byte). Bei Dateien mit wahlfreiem Zugriff ist dies die Datensatzlänge, bei sequentiellen Dateien die Anzahl der gepufferten Zeichen.

Tabelle 35: Die Argumente der Anweisung Open (Forts.)

Mithilfe der Anweisung `Line Input` können Sie eine einzelne Zeile aus einer geöffneten sequentiellen Datei auslesen. Die Syntax der Funktion lautet:

```
Line Input #Dateinummer, Variablenname
```

Die Syntax der `Line Input #`-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Dateinummer	Erforderlich. Eine beliebige gültige Dateinummer.
Variablenname	Erforderlich. Ein gültiger Variablenname vom Typ Variant oder String.

Tabelle 36: Die Argumente der Anweisung Line Input

Im folgenden Beispiel aus Listing 57 wird die erste Zeile einer Textdatei gelesen und im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub TextdateiLesenErsterSatz()
   Dim strTextzeile As String
```

Listing 57: Erste Zeile einer Textdatei lesen

```
    Open "c:\Artikel.txt" For Input As #1
    Line Input #1, strTextzeile
    Debug.Print strTextzeile
    Close #1
End Sub
```

Listing 57: Erste Zeile einer Textdatei lesen (Forts.)

Über die Anweisung `Line Input` wird die erste Zeile der Textdatei in die Variable `StrTextzeile` gelesen. Nach der Ausgabe im Direktfenster wird die Textdatei über die Anweisung `Close` geschlossen.

47 Erste Zeile aus Textdatei lesen

Sollen lediglich einige Zeichen aus der ersten Zeile einer Textdatei gelesen werden, dann kommt die Funktion `Input` zum Einsatz.

Die Funktion `Input` gibt einen Wert vom Typ `String` zurück, der Zeichen aus einer im Modus `Input` oder `Binary` geöffneten Datei enthält. Die Syntax dieser Funktion sieht wie folgt aus:

```
Input(Zahl, [#]Dateinummer)
```

Die Syntax der `Input`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
`Zahl`	Erforderlich. Ein beliebiger gültiger numerischer Ausdruck, der die Zahl der zurückzugebenden Zeichen angibt.
`Dateinummer`	Erforderlich. Eine beliebige gültige Dateinummer.

Tabelle 37: Die Argumente der Funktion Input

48 Nur wenige Zeichen aus Textdatei lesen

Im folgenden Beispiel aus Listing 58 werden die ersten sieben Zeichen der ersten Zeile einer Textdatei in das Direktfenster der Entwicklungsumgebung gelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub EinzelneZeichenLesen()
   Dim strZeichen As String

   Open "C:\Artikel.txt" For Input As #1
```

Listing 58: Einzelne Zeichen lesen über die Funktion Input

```
    strZeichen = Input(7, #1)
    Debug.Print strZeichen
    Close #1
End Sub
```

Listing 58: Einzelne Zeichen lesen über die Funktion Input (Forts.)

Übergeben Sie der Funktion `Input` die Anzahl der gewünschten Zeichen sowie die Dateinummer der Datei, die Sie bereits beim Öffnen der Datei vergeben haben.

Abbildung 27: Nur wenige Zeichen einer Textdatei werden gelesen.

Muss die komplette Textdatei eingelesen werden, dann wird die Funktion `EOF` eingesetzt, um den letzten Satz einer Textdatei zu ermitteln und dahingehend die Schleife aufzubauen.

Die Funktion `EOF` gibt den Wert `True` zurück, wenn das Ende einer Datei erreicht ist. Die Syntax dieser Funktion lautet:

```
EOF(Dateinummer)
```

Das erforderliche Argument `Dateinummer` ist ein Wert vom Typ `Integer`, der eine beliebige gültige Dateinummer enthält.

49 Textdatei Zeile für Zeile auslesen

Im folgenden Beispiel aus Listing 59 wird eine Textdatei Zeile für Zeile gelesen und im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=======================================================

Sub TextdateiEinlesenKomplett()
   Dim strZeile As String
   Open "C:\Artikel.txt" For Input As #1
```

Listing 59: Textdatei komplett einlesen

```
    Do While Not EOF(1)
       Line Input #1, strZeile
       Debug.Print strZeile
    Loop
    Close #1
End Sub
```

Listing 59: Textdatei komplett einlesen (Forts.)

Über die Anweisung `Open` öffnen Sie die Textdatei im Eingabemodus. In einer `Do While`-Schleife arbeiten Sie Zeile für Zeile der Textdatei ab, indem Sie innerhalb der Schleife die Anweisung `Line Input` einsetzen und in der String-Variablen `StrZeile` zwischenspeichern. Diese Zeilen geben Sie bei jedem Schleifendurchlauf über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

Die `Line Input` -Anweisung liest einzelne Zeichen aus einer Datei, bis ein Wagenrücklauf (`Chr(13)`) oder eine Folge aus Wagenrücklaufzeichen und Zeilenvorschubzeichen (`Chr(13)` + `Chr(10)`) gelesen wird. Die Folge aus Wagenrücklaufzeichen und Zeilenvorschubzeichen wird übersprungen und nicht an die gelesene Zeichenfolge angehängt.

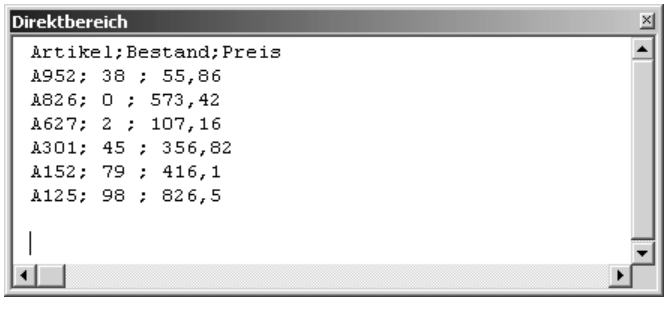


Abbildung 28: Die komplette Textdatei wurde eingelesen.

50 Textdatei schreiben

Die Anweisung `Print` schreibt Daten, die für die Ausgabe formatiert sind, in eine sequentielle Datei. Die Syntax dieser Funktion lautet:

```
Print #Dateinummer, [Ausgabeliste]
```

Die Syntax der `Print` -Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
`Dateinummer`	Erforderlich. Eine beliebige gültige Dateinummer.
`Ausgabeliste`	Optional. Ausdruck oder Liste mit Ausdrücken, die ausgegeben werden sollen.

Tabelle 38: Die Argumente der Anweisung Print

Im folgenden Beispiel aus Listing 60 wird eine neue Textdatei angelegt, das aktuelle Datum sowie der Anwendername erfasst. Anschließend wird die Textdatei wieder geschlossen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub TextdateiSchreiben()
   Open "C:\ini.txt" For Output As #1
   Print #1, Date; Tab; Application.CurrentUser
   Close #1
End Sub
```

Listing 60: Eine Textdatei schreiben

Öffnen Sie über die Anweisung `Open` die Textdatei im Ausgabemodus. Danach geben Sie über die Anweisung `Print` das aktuelle Datum über die Funktion `Date` sowie den Anwendernamen über die Eigenschaft `CurrentUser` in der Textdatei aus. Über die Funktion `Tab` können Sie die einzelnen Informationen durch eine Tabulatorschrittweite voneinander trennen.

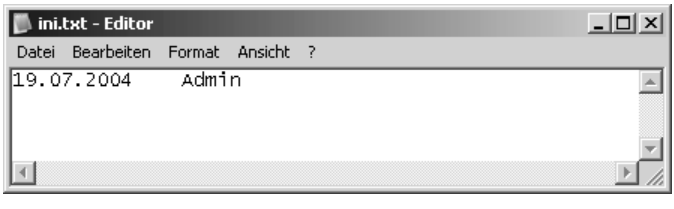


Abbildung 29: Die Textdatei wurde geschrieben.

Sollen Daten aus einer Variablen in eine Textdatei geschrieben werden, dann können Sie hierfür die Anweisung `Put` einsetzen.

Die Anweisung `Put` schreibt Daten aus einer Variablen in eine Datenträgerdatei. Die Syntax dieser Funktion lautet:

```
Put [#]Dateinummer, [Satznummer], Variablennummer
```

Die Syntax der `Put`-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
`Dateinummer`	Erforderlich. Eine beliebige gültige Dateinummer.
`Satznummer`	Optional. Wert vom Typ `Variant` (`Long`). Datensatznummer (für Dateien im Modus `Random`) oder `Byte`-Nummer (für Dateien im Modus `Binary`), bei der der Schreibvorgang beginnt.

Tabelle 39: Die Argumente der Anweisung Put

Teil	Beschreibung
Variablenname	Erforderlich. Name der Variablen, die die auf den Datenträger zu schreibenden Daten enthält.

Tabelle 39: Die Argumente der Anweisung Put (Forts.)

Im folgenden Beispiel aus Listing 61 wird eine Datei angelegt und ein Mustersatz hineingeschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Type MeinDatensatz
   NName As String * 30
   VName As String * 30
End Type

Sub TextdateiFüllen()
   Dim Datensatz As MeinDatensatz
   Dim intNr As Integer

   If Dir("C:\ArtikelNeu.txt") <> "" Then
      Kill "C:\ArtikelNeu.txt"
   End If

   intNr = FreeFile
   Open "C:\ArtikelNeu.txt" For _
   Random As #intNr Len = Len(Datensatz)

   With Datensatz
      .NName = "Vorname"
      .VName = "Nachname"
   End With
   Put #intNr, , Datensatz
   Close #intNr
End Sub
```

Listing 61: Daten aus einer Variablen in eine Textdatei schreiben

Über die Funktion `Dir` prüfen Sie, ob die Textdatei bereits im angegebenen Verzeichnis existiert. Wenn ja, dann löschen Sie diese über die Anweisung `Kill`.

Die Funktion `FreeFile` gibt einen Wert vom Typ `Integer` zurück, der die nächste verfügbare Dateinummer darstellt, die die `Open`-Anweisung zum Öffnen einer Datei verwenden kann. Die Syntax dieser Funktion lautet:

```
FreeFile[(Bereichsnummer)]
```

Das optionale Argument `Bereichsnummer` ist ein Wert vom Typ `Variant`, der den Bereich festlegt, aus dem die nächste Dateinummer zurückgegeben werden wird. Bei 0 (Voreinstellung) wird eine Dateinummer im Bereich 1 bis 255 (einschließlich) zurückgegeben. Bei 1 wird eine Dateinummer im Bereich von 256 bis 511 zurückgegeben.

Danach wird die Textdatei neu angelegt und über die Anweisung `Put` gefüllt. Nach der Füllung der Textdatei schließt die Funktion `Close` die Datei und speichert diese zugleich.

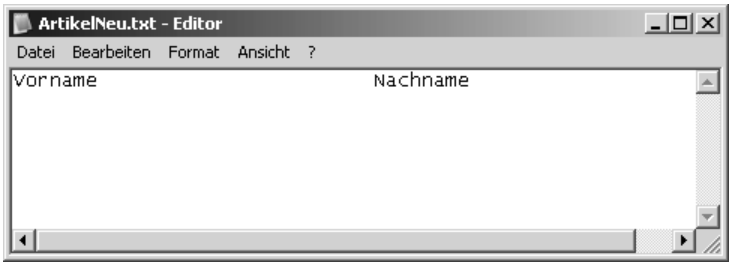


Abbildung 30: Variableninhalte in Textdatei schreiben

51 Textdateien im Batch erstellen

Das Schreiben einer Textdatei wird standardmäßig über die Anweisung `Write` realisiert.

Mit der Anweisung `Write` können Sie Daten in eine sequentielle Datei schreiben. Die Syntax dieser Funktion lautet:

```
Write #Dateinummer, [Ausgabeliste]
```

Die Syntax der `Write`-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
`Dateinummer`	Erforderlich. Eine beliebige gültige Dateinummer.
`Ausgabeliste`	Optional. Ein oder mehrere (durch Kommas getrennte) numerische Ausdrücke oder Zeichenfolgenausdrücke, die in eine Datei geschrieben werden sollen.

Tabelle 40: Die Argumente der Anweisung Write

Sollen hintereinander mehrere Textdateien erstellt werden, dann können Sie dies über eine Schleife erledigen, die Sie in Listing 62 sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================
```

Listing 62: Mehrere Dateien erstellen und schreiben

```
Sub TextdateienImBatch()
   Dim intDateiNr As Integer

   ChDir "C:\Eigene Dateien\"

   For intDateiNr = 1 To 3
      Open "Datei" & intDateiNr & ".txt" For Output As #iDateiNr
      Write #intDateiNr, Now
   Next intDateiNr
   Reset
End Sub
```

Listing 62: Mehrere Dateien erstellen und schreiben (Forts.)

Wechseln Sie zunächst über die Anweisung `ChDir` in das gewünschte Verzeichnis, in welches Sie die Textdateien schreiben möchten. Setzen Sie danach eine Schleife auf, die genau dreimal durchlaufen wird. Innerhalb der Schleife öffnen Sie die Dateien im Ausgabemodus und schreiben über die Anweisung `Write` jeweils das aktuelle Datum sowie die Uhrzeit, die Sie über die Funktion `Now` ermitteln können. Die Anweisung `Reset` schließt alle Datenträgerdateien, die mit der `Open`-Anweisung geöffnet wurden.

52 Zugriffsmodus einer Datei feststellen und setzen

Über die Funktion `FileAttr` können Sie den Zugriffsmodus einer geöffneten Datei feststellen. Die Syntax dieser Funktion lautet:

```
FileAttr(filenumber, returntype)
```

Die Syntax der `FileAttr`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
`Filenumber`	Erforderlich; ein Wert vom Typ `Integer`. Eine beliebige gültige Dateinummer.
`Returntype`	Erforderlich; ein Wert vom Typ `Integer`. Zahl, die die Art der zurückzugebenden Informationen festlegt: Input = 1 Output = 2 Random = 4 Append = 8 `Binary = 32`

Tabelle 41: Die Argumente der Funktion FileAttr

Die Anweisung `GetAttr` gibt einen Wert vom Typ `Integer` zurück, der die Attribute einer Datei, eines Verzeichnisses darstellt. Die Syntax dieser Funktion lautet:

```
GetAttr(Pfadname)
```

Das erforderliche Argument `Pfadname` ist ein Zeichenfolgenausdruck, der einen Dateinamen angibt. `Pfadname` kann ein Verzeichnis oder ein Laufwerk enthalten.

Der von `GetAttr` zurückgegebene Wert ist die Summe der folgenden Attributwerte:

Konstante	Wert	Beschreibung
`vbNormal`	0	Normal.
`vbReadOnly`	1	Schreibgeschützt.
`vbHidden`	2	Versteckt.
`vbSystem`	4	Systemdatei. Beim Macintosh nicht verfügbar.
`vbDirectory`	16	Verzeichnis oder Ordner.
`vbArchive`	32	Datei wurde seit dem letzten Sichern geändert.
`vbAlias`	64	Angegebener Dateiname ist ein Alias. Nur beim Macintosh verfügbar.

Tabelle 42: Die Argumente der Anweisung GetAttr

Im folgenden Beispiel aus Listing 63 wird eine bestimmte Datei aus dem Windows-Verzeichnis überprüft.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub DateiAttributAuslesen()
   Dim attribut As VbFileAttribute

   attribut = vbHidden
   If attribut And GetAttr _
      ("C:\Windows\WindowsShell.Manifest") Then
      MsgBox "Datei ist verborgen"
   Else
      MsgBox "Datei ist nicht verborgen"
   End If
End Sub
```

Listing 63: Dateiattribut auslesen

Über die `SetAttr`-Anweisung können Sie Dateiattribute setzen. Die Syntax dieser Funktion lautet:

```
SetAttr pathname, attributes
```

Die Syntax der `SetAttr`-Anweisung verwendet die folgenden benannten Argumente:

Teil	Beschreibung
Pathname	Erforderlich. Zeichenfolgenausdruck, der einen Dateinamen angibt. Der Dateiname kann ein Verzeichnis oder einen Ordner sowie ein Laufwerk enthalten.
Attributes	Erforderlich. Konstante oder numerischer Ausdruck, die/der die Summe der Dateiattribute angibt.

Tabelle 43: Die Argumente der Anweisung SetAttr

Das Argument `attributes` hat die folgenden Einstellungen:

Konstante	Wert	Beschreibung
vbNormal	0	Normal (Voreinstellung).
vbReadOnly	1	Schreibgeschützt.
vbHidden	2	Versteckt.
vbSystem	4	Systemdatei. Beim Macintosh nicht verfügbar.
vbArchive	32	Datei wurde seit dem letzten Speichern geändert.

Tabelle 44: Die Einstellungen für das Argument attributes

Im folgenden Beispiel aus Listing 64 wird die Datei *Artikel.txt* mit dem Attribut `Versteckt` ausgestattet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub DateiAttributSetzen()
   SetAttr "c:\Artikel.txt", vbHidden
End Sub
```

Listing 64: Dateiattribut setzen

Weisen Sie der Anweisung `SetAttr` die Konstante `vbHidden` zu, um die angegebene Datei mit dem Attribut `Versteckt` zu belegen.

53 Externes Programm starten

Mithilfe der Funktion `Shell` können Sie ein externes Programm aus Access heraus aufrufen. Die Syntax dieser Funktion lautet:

```
Shell(pathname[,windowstyle])
```

Die Syntax der `Shell`-Funktion verwendet die folgenden benannten Argumente:

Teil	Beschreibung
pathname	Erforderlich; Wert vom Typ Variant (String). Name des auszuführenden Programms sowie alle erforderlichen Argumente oder Befehlszeilenoptionen. Auch Verzeichnis- oder Laufwerksangaben können enthalten sein.
windowstyle	Optional. Wert vom Typ Variant (Integer), der dem Stil des Fensters entspricht, in dem das Programm ausgeführt werden soll. Wenn windowstyle nicht angegeben wird, erhält das Programm den Fokus und wird im minimierten Zustand gestartet.

Tabelle 45: Die Argumente der Funktion Shell

Die Werte des benannten Arguments windowstyle lauten:

Konstante	Wert	Beschreibung
vbHide	0	Das Fenster ist ausgeblendet und das ausgeblendete Fenster erhält den Fokus.
vbNormalFocus	1	Das Fenster hat den Fokus und die ursprüngliche Größe und Position werden wiederhergestellt.
VbMinimizedFocus	2	Das Fenster wird als Symbol mit Fokus angezeigt.
vbMaximizedFocus	3	Das Fenster wird maximiert mit Fokus angezeigt.
vbNormalNoFocus	4	Die zuletzt verwendete Größe und Position des Fensters wird wiederhergestellt. Das momentan aktive Fenster bleibt aktiv.
vbMinimizedNoFocus	6	Das Fenster wird als Symbol angezeigt. Das momentan aktive Fenster bleibt aktiv.

Tabelle 46: Die Einstellungen für das Argument windowstyle

Notizblock aufrufen

Im folgenden Beispiel aus Listing 65 wird das Programm Notepad gestartet. Danach wird versucht, eine bestimmte Datei zu laden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub ExternesProgrammAufrufen()
   Dim varA As Variant
```

Listing 65: Ein externes Programm aufrufen

```
    varA = Shell("C:\Windows\Notepad.exe C:\Artikel.txt", 3)
End Sub
```

Listing 65: Ein externes Programm aufrufen (Forts.)

Übergeben Sie der Funktion `Shell` den Pfad sowie den Namen der Anwendung, die Sie starten möchten. Soll zusätzlich eine Datei geöffnet werden, dann geben Sie diese inklusive des Pfads an. Über das letzte Argument legen Sie fest, in welchem Fensterstil die Anwendung angezeigt werden soll. Dabei steht die Nummer 3 für die maximierte Fensteransicht.

Windows Explorer mit eingestelltem Verzeichnis aufrufen

Im folgenden Beispiel aus Listing 66 wird der Windows-Explorer über die Funktion `Shell` aufgerufen und ein bestimmtes Verzeichnis schon ausgewählt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub WindowsExlorerAufrufen()
   On Error GoTo fehler
   Shell "C:\WINDOWS\Explorer.EXE c:\Eigene Dateien", 1
   Exit Sub

   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 66: Den Windows Explorer aufrufen und Verzeichnis vorab einstellen – Variante 1

Noch schneller können Sie den Windows-Explorer aufrufen, indem Sie den Pfad zu dieser Anwendung weglassen, wie es im Makro aus Listing 67 gezeigt wird. Die Windows-Standardprogramme werden in der Regel automatisch gefunden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================
Sub WindowsExlorerAufrufenKurzform()
   On Error GoTo fehler
   Shell "Explorer.EXE c:\Eigene Dateien", 1
   Exit Sub

   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 67: Den Windows-Explorer aufrufen und Verzeichnis einstellen – Variante 2

In beiden Makros stellen Sie das gewünschte Verzeichnis, das im Windows-Explorer angezeigt werden soll, gleich im Anschluss an die ausführbare Datei (*exe*) ein.

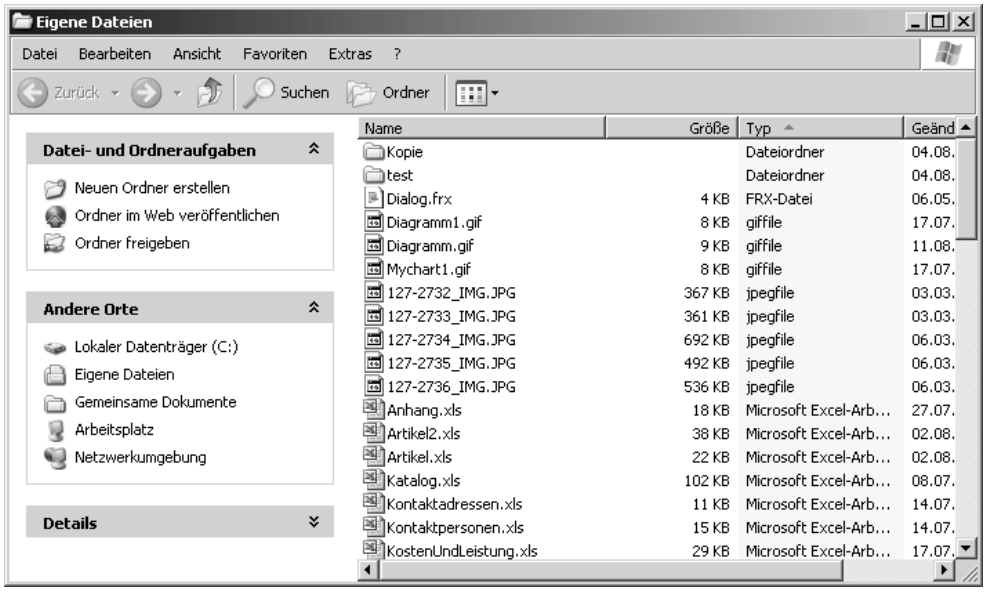


Abbildung 31: Der Explorer wurde im angegebenen Verzeichnis geöffnet.

Internet Explorer aufrufen und Bild laden

Im folgenden Beispiel aus Listing 68 wird der Internet Explorer aufgerufen und ein bestimmtes Bild aus einem Verzeichnis direkt geöffnet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================


Sub InternetExplorerAufrufen()
   On Error GoTo fehler
   Shell "c:\Programme\Internet explorer\iexplore.exe " & _
      "c:\Eigene Dateien\128-2803_IMG.JPG", 2
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 68: Internet Explorer aufrufen und Bild laden

Übergeben Sie der Funktion `Shell` den kompletten Pfad des Internet Explorers und im Anschluss den Pfad zur Bilddatei, die angezeigt werden soll.

Abbildung 32: Früh übt sich

54 Absolutwert ausgeben

Spielt das Vorzeichen einer Zahl keine Rolle, dann spricht man von einem Absolutwert einer Zahl. Diesen Absolutwert bekommen Sie, indem Sie die Funktion `ABS` einsetzen. Die Syntax dieser Funktion lautet:

```
Abs(Zahl)
```

Das erforderliche Argument `Zahl` kann ein beliebiger zulässiger numerischer Ausdruck sein. Wenn `Zahl` den Wert Null enthält, wird Null zurückgegeben. Wenn die Variable nicht initialisiert ist, wird der Wert Null zurückgegeben.

55 Vorzeichen auslesen

Die Funktion `Sgn` gibt das Vorzeichen eines Werts als Faktor (–10 oder 1) zurück. Die Syntax der Funktion lautet:

```
Sgn(Zahl)
```

Das erforderliche Argument `Zahl` kann ein beliebiger numerischer Ausdruck sein.

Wert von Zahl	Rückgabewert von Sgn
Größer als Null	1
Gleich Null	0
Kleiner als Null	–1

Tabelle 47: Die Rückgabewerte der Funktion Sgn

Im folgenden Beispiel aus Listing 69 werden einige Zahlen dem Vorzeichen nach ausgewertet. Dabei wird eine Funktion erstellt, der die Zahl übergeben wird. Als Rückgabewert meldet die Funktion einen Text, den Sie aus der ersten Spalte der Tabelle 47 entnehmen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Function VorzeichenErmitteln(sngZahl As Single) As String
   Select Case Sgn(sngZahl)
      Case 1
         VorzeichenErmitteln = "Größer Null"
      Case 0
         VorzeichenErmitteln = "Null"
      Case -1
         VorzeichenErmitteln = "Kleiner Null"
   End Select
 End Function

Sub Vorzeichen()
   MsgBox VorzeichenErmitteln(-65)
End Sub
```

Listing 69: Vorzeichen prüfen über die Funktion Sgn

56 Abschreibung errechnen (degressiv)

Soll die Abschreibung eines Guts nach der degressiven Methode errechnet werden, dann wird für diese Aufgabe die Funktion `DDB` eingesetzt. Diese Funktion errechnet den Abschreibungswert bei geometrisch degressiver Abschreibung. Die Syntax dieser Funktion lautet:

```
DDB(cost, salvage, life, period[, factor])
```

Die `DDB`-Funktion hat die folgenden benannten Argumente:

Teil	Beschreibung
`Cost`	Erforderlich. Ein Wert vom Typ `Double`, der die Anschaffungskosten des Vermögenswerts angibt.
`Salvage`	Erforderlich. Ein Wert vom Typ `Double`, der den Wert des Vermögenswerts am Ende seiner Nutzungsdauer angibt.
`Life`	Erforderlich. Ein Wert vom Typ `Double`, der die Länge der Nutzungsdauer des Vermögenswerts angibt.
`Period`	Erforderlich. Ein Wert vom Typ `Double`, der den Zeitraum angibt, für den die Abschreibung des Vermögenswerts berechnet wird.
`Factor`	Optional. Ein Wert vom Typ `Variant`, der den Faktor angibt, um den der Wert vermindert wird. Wird der Wert nicht angegeben, so wird 2 (geometrisch degressive Methode) angenommen.

Tabelle 48: Die Argumente der Funktion DDB

Im folgenden Beispiel aus Listing 70 wird eine Maschine nach der geometrisch degressiven Abschreibungsmethode abgeschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Sub AbschreibungDegressiv()
  Dim curWert As Currency
  Dim curEndwert As Currency
   Dim intDauer As Integer
   Dim intPeriode As Integer

   intDauer = 5
   curWert = 2500
   Endwert = 1
   For intPeriode = 1 To intDauer
      Debug.Print "Jahr " & intPeriode & ": " & _
      DDB(curWert, curEndwert, intDauer, intPeriode)
```

Listing 70: Die degressive Abschreibung berechnen

```
    Next intPeriode
End Sub
```

Listing 70: Die degressive Abschreibung berechnen (Forts.)

Deklarieren Sie zunächst einige Variablen und füllen Sie diese mit den Parametern, die für die Abschreibung gefordert werden. In einer `For Next`-Schleife ermitteln Sie mithilfe der Funktion `DDB` die jeweiligen Abschreibungsbeträge im Direktfenster der Entwicklungsumgebung.

Abbildung 33: Die degressive Abschreibung wurde errechnet.

57 Abschreibung errechnen (linear)

Soll die Abschreibung eines Guts nach der linearen Methode errechnet werden, dann wird für diese Aufgabe die Funktion `SLN` eingesetzt. Diese Funktion errechnet den periodischen Abschreibungswert bei linearer Abschreibung über einen bestimmten Zeitraum. Die Syntax der Funktion lautet:

```
SLN(cost, salvage, life)
```

Die `SLN`-Funktion hat folgende benannte Argumente:

Teil	Beschreibung
`Cost`	Erforderlich. Ein Wert vom Typ Double, der die Anschaffungskosten des Vermögenswerts angibt.
`Salvage`	Erforderlich. Ein Wert vom Typ Double, der den Vermögenswert am Ende seiner Nutzungsdauer angibt.
`Life`	Erforderlich. Ein Wert vom Typ Double, der die Länge der Nutzungsdauer des Vermögenswerts angibt.

Tabelle 49: Die Argumente der Funktion SLN

Im folgenden Beispiel aus Listing 71 wird eine Maschine nach der linearen Abschreibungsmethode abgeschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Sub AbschreibungLinear()
   Dim curWert As Currency
   Dim curEndwert As Currency
   Dim intDauer As Integer
   Dim intz As Integer

   intDauer = 5
   curWert = 2500
   curEndwert = 0
   For intz = 1 To intDauer
      Debug.Print "Jahr " & intz & ": " & _
      SLN(curWert, curEndwert, intDauer)
   Next intz
End Sub
```

Listing 71: Die lineare Abschreibung errechnen

Deklarieren Sie zunächst einige Variablen und füllen Sie diese mit den Parametern, die für die Abschreibung gefordert werden. In einer `For Next`-Schleife ermitteln Sie mithilfe der Funktion `SLN` die jeweiligen Abschreibungsbeträge im Direktfenster der Entwicklungsumgebung.

Abbildung 34: Die lineare Abschreibung wurde errechnet.

58 Ganzzahligen Wert ermitteln

Um aus einer Zahl mit Dezimalstellen den ganzzahligen Wert zu ermitteln, können Sie die Funktionen `Fix` oder `Int` einsetzen, je nachdem, welches Ergebnis Sie erhalten möchten.

Die Funktion `Fix` gibt den Vorkommawert einer Zahl zurück.

```
Fix(Zahl)
```

Die Funktion `Int` gibt den Vorkommawert einer Zahl zurück.

```
Int(Zahl)
```

Das erforderliche Argument `Zahl` ist ein Wert vom Typ `Double` oder ein beliebiger zulässiger numerischer Ausdruck. Wenn `Zahl` den Wert Null enthält, wird Null zurückgegeben.

Im folgenden Beispiel aus Listing 72 werden die ganzzahligen Werte einiger Zahlen ermittelt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'====================================================

Sub Ganzzahl()
   Debug.Print "Fix: aus 10,3 wird " & Fix(10.3)
   Debug.Print "Fix: aus 10,1 wird " & Fix(10.1)
   Debug.Print "Fix: aus -10,7 wird " & Fix(-10.7)
   Debug.Print "Fix: aus -10,3 wird " & Fix(-10.3)
   Debug.Print "Int: aus 10,3 wird " & Int(10.3)
   Debug.Print "Int: aus 10,1 wird " & Int(10.1)
   Debug.Print "Int: aus -10,7 wird " & Int(-10.7)
   Debug.Print "Int: aus -10,3 wird " & Int(-10.3)
End Sub
```

Listing 72: Ganzzahlige Werte über die Funktionen Fix und Int herstellen

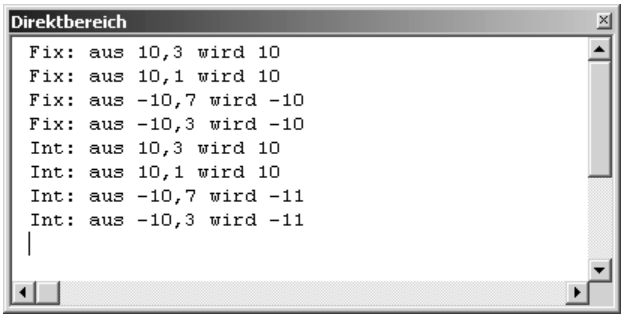


Abbildung 35: Die beiden Funktionen liefern zum Teil unterschiedliche Ergebnisse.

59 Zahlen runden

Mithilfe der Funktion `Round` runden Sie einen Zahlenwert auf Basis einer angegebenen Dezimalstelle. Die Syntax dieser Funktion lautet:

```
Round(Ausdruck [,AnzahlAnDezimalpunktn])
```

Die Syntax der `Round`-Funktion besteht aus folgenden Teilen:

Teil	Beschreibung
Ausdruck	Erforderlich. Numerischer Ausdruck, der gerundet wird.
AnzahlAnDezimalpunkten	Optional. Zahl, die angibt, wie viele Stellen rechts vom Dezimalpunkt beim Runden berücksichtigt werden. Wird dieser Wert ausgelassen, gibt die Round-Funktion Ganzzahlen zurück.

Tabelle 50: Die Argumente der Funktion Round

Im folgenden Beispiel aus Listing 73 werden einige Zahlen auf verschiedene Art und Weise gerundet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Sub Runden()
   Debug.Print Round(12.456, 0)
   Debug.Print Round(12.456, 1)
   Debug.Print Round(12.456, 2)
   Debug.Print Round(12.456, 3)
   Debug.Print Round(-12.456, 1)
   Debug.Print Round(-12.456, 2)
   Debug.Print Round(-12.456, 3)
End Sub
```

Listing 73: Zahlen runden

Je nachdem, wie Sie das zweite Argument der Funktion Round angeben, wird die Rundung der Zahlen durchgeführt.

Abbildung 36: Die Ergebnisse des Rundens

60 Zinssatz errechnen

Wird ein Kredit genommen, wobei Kreditsumme sowie Laufzeit und die monatliche Rückzahlung bekannt sind, dann können Sie den Zinssatz über die Funktion `Rate` ausrechnen.

Die Funktion `Rate` errechnet den Zinssatz zu einer Annuität bei festen Zeiträumen, konstanten Zahlungen und festem Zinssatz.

```
Rate(nper, pmt, pv[, fv[, type[, guess]]])
```

Die `Rate`-Funktion hat folgende benannte Argumente:

Teil	Beschreibung
`nper`	Erforderlich. Ein Wert vom Typ `Double`, der die Gesamtanzahl der Zahlungszeiträume für die Annuität angibt.
`pmt`	Erforderlich. Ein Wert vom Typ `Double`, der die Zahlung pro Zeitraum angibt. Die Zahlungen enthalten gewöhnlich Kapital und Zinsen und ändern sich während der Laufzeit einer Annuität nicht.
`pv`	Erforderlich. Ein Wert vom Typ `Double`, der den Barwert oder heutigen Wert einer Folge zukünftiger Aus- oder Einzahlungen angibt.
`fv`	Optional. Ein Wert vom Typ `Variant`, der den Endwert oder Kontostand angibt, der nach der letzten Zahlung erreicht sein soll.
`type`	Optional. Ein Wert vom Typ `Variant`, der angibt, wann Zahlungen fällig sind. Bei 0 sind die Zahlungen am Ende eines Zahlungszeitraums fällig, bei 1 zu Beginn des Zahlungszeitraums. Wird der Wert nicht angegeben, so wird 0 angenommen.
`guess`	Optional. Ein Wert vom Typ `Variant`, der einen von Ihnen geschätzten Wert enthält, der von `Rate` zurückgegeben wird. Wird der Wert nicht angegeben, so ist `guess` gleich 0,1 (10 Prozent).

Tabelle 51: Die Argument der Funktion Rate

Im folgenden Beispiel aus Listing 74 wird ein Kredit von 150.000 Euro aufgenommen. Der Kredit soll nach 12 Jahren zurückbezahlt sein. Monatlich werden jeweils 650 Euro abbezahlt. Wie hoch ist nun der Zinssatz?

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=======================================================
```

Listing 74: Zinssatz errechnen über die Funktion Rate

```
Sub ZinssatzErrechnen()
   Dim curEndWert As Currency
   Dim curAnfangwert As Currency
   Dim curBetrag As Currency
   Dim intMonate As Integer
   Dim intZahlTyp As Integer
   Dim sngZINS As Single

   curEndWert = 0
   curAnfangwert = 150000
   curBetrag = 650
   intMonate = 12 * 12
   intZahlTyp = 0

   sngZINS = (Rate(intMonate, -curBetrag, curAnfangwert, _
      curEndWert, intZahlTyp) * 12) * 100
   MsgBox "Ihr Zinssatz beträgt " & _
      Format(sngZINS, "#,#0.00")
End Sub
```

Listing 74: Zinssatz errechnen über die Funktion Rate (Forts.)

Deklarieren Sie zunächst einige Variablen und füllen Sie diese mit den Parametern, die für die Zinssatzberechnung gefordert werden. Füllen Sie die benötigten Variablen und übergeben Sie diese der Funktion `Rate`. Geben Sie danach das Ergebnis am Bildschirm aus.

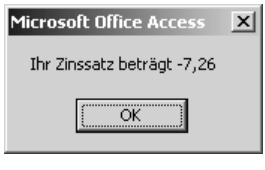


Abbildung 37: Das Ergebnis der Zinsrechnung

61 Ansparungsbetrag errechnen

Wenn Sie einen monatlichen Betrag für die Laufzeit über mehrere Jahre zu einem bestimmten Zinssatz anlegen, dann erhalten Sie zum Ende der Laufzeit ein Gesamtkapital, das Sie über die Funktion `FV` ausrechnen können.

Die Funktion `FV` errechnet den zukünftigen Wert einer Annuität bei konstanter Zahlung, festem Zins und fester Laufzeit.

```
FV(rate, nper, pmt[, pv[, type]])
```

Die `FV`-Funktion hat die folgenden benannten Argumente:

Teil	Beschreibung
rate	Erforderlich. Ein Wert vom Typ Double, der den Zinssatz pro Zeitraum angibt. Wenn Sie beispielsweise einen Kredit mit einem Jahreszins von 10 Prozent aufnehmen und monatliche Zahlungen vereinbart haben, beträgt der Zinssatz pro Zeitraum 0,1 dividiert durch 12 oder 0,0083.
nper	Erforderlich. Ein Wert vom Typ Integer, der die Gesamtanzahl der Zahlungszeiträume für die Annuität angibt. Wenn Sie beispielsweise monatliche Zahlungen für einen Autokredit mit 4 Jahren Laufzeit vereinbart haben, beträgt die Summe der Zahlungszeiträume für Ihren Kredit 4 * 12 (oder 48).
pmt	Erforderlich. Ein Wert vom Typ Double, der die Zahlung pro Zeitraum angibt. Die Zahlungen enthalten gewöhnlich Kapital und Zinsen und ändern sich während der Laufzeit einer Annuität nicht.
pv	Optional. Ein Wert vom Typ Variant, der den Barwert (oder Gesamtbetrag) einer Folge zukünftiger Zahlungen zum jetzigen Zeitpunkt angibt. Wenn Sie beispielsweise Geld aufnehmen, stellt die Kredithöhe für den Kreditgeber den Barwert der von Ihnen zu leistenden monatlichen Zahlungen dar. Wird der Barwert nicht angegeben, so wird 0 angenommen.
type	Optional. Ein Wert vom Typ Variant, der angibt, wann Zahlungen fällig sind. Bei 0 sind die Zahlungen am Ende des Zahlungszeitraums fällig, bei 1 zu Beginn des Zahlungszeitraums. Wird der Wert nicht angegeben, so wird 0 angenommen.

Tabelle 52: Die Argumente der Funktion FV

Im folgenden Beispiel aus Listing 75 wird errechnet, wie sich ein angelegtes Kapital von 10000 Euro über einen Zeitraum von 12 Jahren bei einem festen Zinssatz von 4,5% im Jahr bei einer monatlichen Einzahlung von 100 Euro entwickelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Sub EndKapitalErrechnen()
   Dim curWert As Currency
   Dim intDauer As Integer
   Dim curRate As Currency
   Dim sngZins As Single

   curWert = 10000
   intDauer = 12 * 12
   curRate = 100
   sngZins = 0.045 / 12
```

Listing 75: Das Endkapital wurde berechnet.

```
    MsgBox "Gesamtkapital nach 12 Jahren : " & _
       Format(-FV(sngZins, curRate, intDauer, curWert), "#,##0 _")
End Sub
```

Listing 75: Das Endkapital wurde berechnet. (Forts.)

Übergeben Sie der Funktion `FV` die benötigten Argumente. Da die Dauer in Monaten angegeben werden muss, multiplizieren Sie die Laufzeit in Jahren mit dem Multiplikator 12. Den Zinssatz müssen Sie ebenfalls auf Monatsbasis herunterrechnen. Über die Funktion `Format` bringen Sie das Ergebnis in das gewünschte Format.

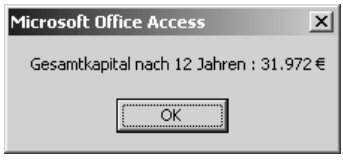


Abbildung 38: Das formatierte Endergebnis

62 Zufallszahlen erzeugen

Zum Erzeugen von Zufallszahlen werden zwei Befehle benötigt:

Die Anweisung `Randomize` initialisiert den Zufallsgenerator.

```
Randomize [Zahl]
```

Das optionale Argument `Zahl` ist ein Wert vom Typ `Variant` oder ein beliebiger zulässiger numerischer Ausdruck.

Die Funktion `Rnd` liefert eine Zufallszahl mit Dezimalstellen zwischen 0 und 1.

```
Rnd[(Zahl)]
```

Das optionale Argument `Zahl` ist ein Wert vom Typ `Single` oder ein beliebiger zulässiger numerischer Ausdruck.

Zufallszahlen im Bereich von 1 bis 100

Im folgenden Beispiel aus Listing 76 werden mithilfe der Funktion `Rnd` Zufallszahlen im Bereich 1 bis 100 gebildet.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=================================================

Sub ZufallsZahlenErzeugen()
```

Listing 76: Zufallszahlen zwischen 1 und 100 bilden

```
    Dim intz As Integer

    Randomize
    For intz = 1 To 10
       Debug.Print (Int(100 * Rnd + 1))
    Next intz
End Sub
```

Listing 76: Zufallszahlen zwischen 1 und 100 bilden (Forts.)

Über die Anweisung `Randomize` initialisieren Sie den Zufallsgenerator. Danach setzen Sie eine Schleife auf, die genau 10 Mal durchlaufen wird. Innerhalb der Schleife erstellen Sie Zufallszahlen im Bereich zwischen 1 und 100. Über den Einsatz der Funktion `Int` werden aus diesen Zufallszahlen, die noch Nachkommastellen aufweisen, Ganzzahlwerte gemacht.

Abbildung 39: Zufallszahlen wurden erzeugt.

Zufallsbuchstaben zwischen A und Z erzeugen

Wenn Sie das Makro aus Listing 76 erweitern, dann können Sie mithilfe der Funktion `Rnd` sowie der Funktion `Chr` Zufallsbuchstaben wie in Listing 77 gezeigt bilden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlMath
'=====================================================

Sub ZufallsBuchstabenErzeugen()
   Dim intz As Integer

   Randomize
   For intz = 1 To 10
      Debug.Print (Chr(Int((122 - 97 + 1) * Rnd + 97)))
   Next intz
End Sub
```

Listing 77: Zufallsbuchstaben erzeugen

Über die Anweisung `Randomize` initialisieren Sie den Zufallsgenerator. Danach setzen Sie eine Schleife auf, die genau 10 Mal durchlaufen wird. Innerhalb der Schleife erstellen Sie Zufallsbuchstaben. Dabei wenden Sie die Funktion `Chr` an, um den Zahlenwert in einen Buchstaben zu wandeln.

Abbildung 40: Die Zufallsbuchstaben wurden erzeugt.

63 Eingabe prüfen

Die Prüffunktionen kommen immer dann zum Einsatz, wenn Sie kontrollieren möchten, welche Eingabe ein Anwender in einem Formular oder einem anderen Dialog vornimmt.

Im folgenden Beispiel aus Listing 78 muss der Anwender einen numerischen Wert in eine Inputbox eingeben. Diese Eingabe können Sie über die Funktion `IsNumeric` überprüfen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlPrüf
'=====================================================

Sub EingabeChecken()
   Dim strEingabe As String

   strEingabe = InputBox("Geben Sie Ihr Alter ein!")

   If strEingabe <> "" Then
      If IsNumeric(strEingabe) Then
         MsgBox "Eingabe ok!"
      Else
         MsgBox "Eingabe falsch!"
      End If
   End If
End Sub
```

Listing 78: Eingabe auf Richtigkeit prüfen

Um zu testen, ob der Anwender überhaupt eine Eingabe vorgenommen hat, überprüfen Sie, ob nach dem Aufruf der Inputbox und dem Bestätigen mit der OK-Schaltfläche die Variable `StrEingabe` einen Inhalt hat. Nur wenn dies so ist, dann wenden Sie die Funktion `IsNumeric` an, um zu testen, ob eine korrekte Zahl eingegeben wurde.

64 Datenfeldcheck

Mithilfe der Funktion `IsArray` prüfen Sie, ob es sich bei der angesprochenen Variablen um ein Datenfeld (Array) handelt.

Um diese Funktion zu üben, schreiben Sie ein Makro, in dem Sie festlegen, wie groß ein Datenfeld angelegt werden soll. Diese Information übergeben Sie einer Funktion, die das Datenfeld in der gewünschten Größe anlegt und an die aufrufende Prozedur zurückliefert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlPrüf
'=====================================================

Function ArrayBilden(intA As Integer) As Variant
   ReDim ArrK(lngG)
   Dim intz As Integer

   For intz = LBound(ArrK, 1) To UBound(ArrK, 1)
      ArrK(intz) = intz
   Next intz
   ArrayBilden = ArrK
End Function


Sub DynamischenArrayBilden()
   Dim ArrK As Variant
   Dim intz As Integer

   ArrK = ArrayBilden(10)
   If IsArray(ArrK) Then
      For intz = LBound(ArrK, 1) To UBound(ArrK, 1)
         Debug.Print ArrK(intz)
      Next intz
   End If
End Sub
```

Listing 79: Datenfelder bilden und prüfen

In der Zeile `ArrK=ArrayBilden(10)` rufen Sie die Funktion `ArrayBilden` auf und übergeben ihr den Wert 10. Damit legt die Funktion `ArrayBilden` ein Datenfeld mit genau 10 Datenfeldern an, eigentlich ja 11, da die 0 mitgezählt wird. Nachdem die Funktion an die aufrufende Prozedur das angelegte Datenfeld zurückmeldet, prüfen Sie über die Funktion `IsArray`, ob die Rückgabe der Funktion auch den richtigen Datentyp, nämlich ein Datenfeld, liefert. Wenn ja, dann setzen Sie eine Schleife auf, in der das Datenfeld ausgelesen wird. Setzen Sie dazu den Index l zu Beginn der Schleife mithilfe der Funktion `LBound` auf den ersten Eintrag des Datenfelds und arbeiten Sie sich dann bis zum letzten Feld des Datenfelds vor, welches Sie über die Funktion `UBound` ermitteln. Geben Sie nach jedem Schleifendurchlauf den Inhalt des jeweiligen Datenfelds im Direktbereich über die Anweisung `Debug.Print` aus.

Abbildung 41: Das Datenfeld wurde angelegt.

65 Tabellencheck vornehmen

Mithilfe der Funktion `IsObject` können Sie beispielsweise überprüfen, ob sich ein bestimmtes Objekt in Ihrer Datenbank befindet. So wird diese Funktion im nächsten Beispiel eingesetzt, um zu prüfen, ob sich eine bestimmte Tabelle in der aktuellen Datenbank befindet. Nur dann soll diese Tabelle auch geöffnet werden. Diese Prüfroutine sieht wie folgt aus:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlPrüf
'=====================================================

Function Tabelleda(strTab As String) As Boolean
   On Error Resume Next
   Tabelleda = IsObject(CurrentDb.TableDefs(strTab))
End Function
```

Listing 80: Tabellenexistenz prüfen

```
Sub PrüfenAufTabelle()
   Dim b As Boolean

   If Tabelleda("Personal") = True Then
      DoCmd.OpenTable "Personal"
   Else
      MsgBox "Tabelle ist nicht vorhanden!"
   End If
End Sub
```

Listing 80: Tabellenexistenz prüfen (Forts.)

Über die Methode `CurrentDb` haben Sie Zugriff auf die aktuelle geöffnete Datenbank. Mithilfe der Auflistung `TableDefs` können Sie kontrollieren, ob sich die übergebene Tabelle in der Variablen `s` in der Datenbank befindet. Wenn ja, dann wenden Sie die Methode `OpenTable` an, um die gewünschte Tabelle zu öffnen.

Hinweis

Wie Sie weitere Prüffunktionen für Access programmieren können, erfahren Sie im folgenden Kapitel.

66 Punkte entfernen

Mithilfe der Typumwandlungsfunktion `CDbl` können Sie beispielsweise aus einem String, der einen Wert enthält, alle Punkte entfernen. Dieser String wird dann in einen Ausdruck des Datentyps `Double` umgewandelt.

Im nächsten Beispiel aus Listing 81 werden aus einem String, der Tausenderpunkte enthält, diese Punkte entfernt und in einer Variablen vom Typ `Double` gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=====================================================

Sub PunkteRaus()
   Dim strText As String
   Dim dblBetrag As Double

   strText = "255.435.190,45"
   Debug.Print "Zeichenkette vorher: " & strText
   dblBetrag = CDbl(strText)
   Debug.Print "Zeichenkette nachher: " & dblBetrag
End Sub
```

Listing 81: Punkte aus Zeichenfolge eliminieren

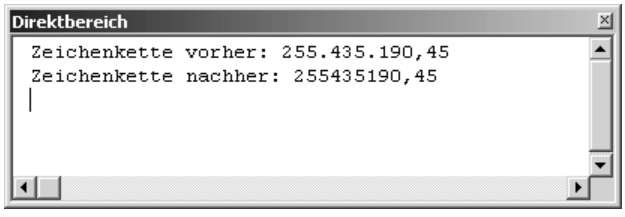


Abbildung 42: Die Punkte wurden entfernt.

67 Datumswerte umwandeln

Mithilfe der Typumwandlungsfunktion `CDate` können Sie Datumsangaben in Zeichenfolgen in echte Datumsangaben umwandeln. So wird beispielsweise aus der Zeichenfolge 21. November 2004 das Datum 21.11.2004.

Sehen Sie weitere Beispiele im Listing 82.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=====================================================



Sub DatumswerteWandeln()
   Const Datum1 = "11/25/1998 8:30 AM"
   Const Datum2 = "12. Februar 2002  "
   Const Datum3 = "Januar, 2002      "

   Debug.Print Datum1 & " ---> " & CDate(Datum1)
   Debug.Print Datum2 & " ---> " & CDate(Datum2)
   Debug.Print Datum3 & " ---> " & CDate(Datum3)
End Sub
```

Listing 82: Zeichenfolgen in Datumswerte umwandeln

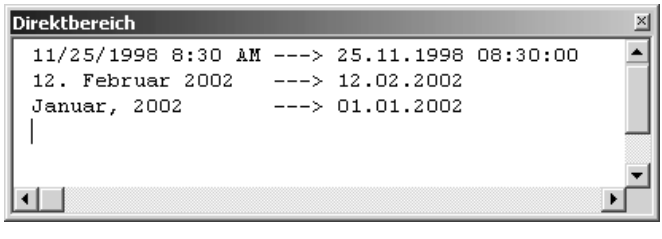


Abbildung 43: Die Textfolgen wurden erfolgreich umgewandelt.

68 Nachkommastellen entfernen

Mithilfe der Funktion `CLng` können Sie einen Wert in einen Datentyp `Long` umwandeln. Dabei werden eventuell existierende Nachkommastellen gerundet.

Im folgenden Beispiel aus Listing 83 werden genau zehn Zufallszahlen im Zahlenbereich zwischen 1000 und 99999 gebildet. Diese werden mithilfe der Funktion `CLng` in den richtigen Datentyp umgewandelt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=================================================

Sub NachkommastellenWeg()
   Dim intz As Integer
   Dim lngZahl As Long

   For intz = 1 To 10
      lngZahl = CLng(99900 * Rnd + 1)
      Debug.Print "Zufallszahl " & Format(intz, "00") & _
         " lautet " & Format(lngZahl, "0,##")
   Next intz
End Sub
```

Listing 83: Zufallszahlen ohne Nachkommastellen bilden

Über die Anweisung `CLng(99999 * Rnd + 1)` bilden Sie eine Zufallszahl zwischen 10000 und 99999 und weisen dieser Zahl den Datentyp `Long` zu. Über die Funktion `Format` bringen Sie die Werte in die gewünschte Form.

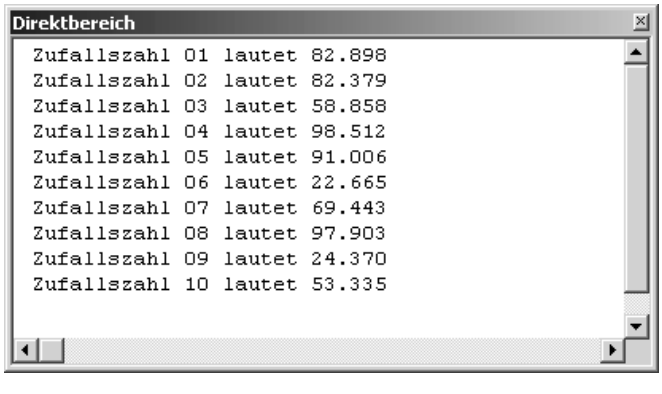


Abbildung 44: Zufallszahlen mit Tausenderpunkt

69 Zahl aus Zeichenfolge extrahieren

Mithilfe der Funktion `Val` können Sie in einer Zeichenfolge enthaltene numerische Werte herauszuholen.

Im folgenden Beispiel aus Listing 84 wird aus einer Zeichenfolge nur der numerische Wert extrahiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=====================================================

Sub ZahlAusFeldExtrahieren()
   Debug.Print "70469 Stuttgart -----> " & Val("70469 Stuttgart")
   Debug.Print "70467 Stuttgart -----> " & Val("70467 Stuttgart")
End Sub
```

Listing 84: Zahlenteil einer Zeichenfolge extrahieren

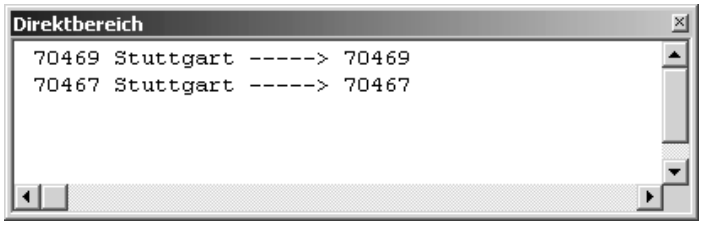


Abbildung 45: Die Postleitzahl wird extrahiert.

70 Zahlen in Texte umwandeln

Für die Umwandlung von Zahlen in Texte wird die Funktion `Str` eingesetzt. Dabei wird bei der Umwandlung immer für die erste Stelle ein Vorzeichen reserviert. Im Makro aus Listing 85 sehen Sie, wie die Umwandlung dabei vor sich geht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap01
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=====================================================

Sub ZahlInText()
   Dim strText As String
```

Listing 85: Zahlen in Textwerte umwandeln

```
    strText = Str(679)        ' Liefert " 679".
    strText = Str(-679.34)    ' Liefert "-679.34".
    strText = Str(123.986)    ' Liefert " 123.986".
End Sub
```

Listing 85: Zahlen in Textwerte umwandeln (Forts.)

Weitere Funktionen

In diesem Kapitel können Sie ausgesuchte API-Funktionen sowie Beispiele für eigene Funktionen nachschlagen. Viele Aufgaben in Access müssen nicht unbedingt durch den Einsatz von VBA gelöst werden. Zahlreiche Lösungen stecken auch schon direkt in Windows. Ihr Betriebssystem stellt Ihnen eine ganze Reihe Dynamic Link Libraries (DLLs) zur Verfügung, die Sie bei der Programmierung von Access einsetzen können. Diese DLLs enthalten Funktionen, die u.a. für die Speicher- und Ressourcenverwaltung, die Bereitstellung von Dialog- und Steuerelementen, den Einsatz von Zeichenfunktionen, die Behandlung von Menü- und Symbolleisten, die Verwendung von Multimediafunktionen oder auch die Verwaltung von E-Mails verantwortlich sind.

Wie Sie sehen, decken API-Funktionen nahezu alle Aufgaben in Windows und somit auch in Access ab. Der Vorteil von API-Aufrufen ist beispielsweise auch deren universelle Einsetzbarkeit. Ob Sie in Access, Excel, Word oder einer anderen beliebigen Windows-Anwendung mit Programmierschnittstelle arbeiten, ist dabei gleichgültig. Sie können API-Funktionen aus nahezu jeder Anwendung heraus aufrufen und für Ihre Aufgaben nutzen.

Die bekannteste API ist die Windows-API, zu der auch die DLLs gehören, aus denen das Windows-Betriebssystem besteht. Jede Windows-Anwendung kommuniziert direkt oder indirekt mit der Windows-API. Durch die Windows-API wird sichergestellt, dass das Verhalten aller unter Windows laufenden Anwendungen konsistent bleibt.

Ein Nachteil von API-Funktionen ist, dass Sie diese in der Online-Hilfe leider nicht nachschlagen können bzw. dass die Syntax von API-Funktionen recht kompliziert ist. Ein weiterer Punkt ist der, dass API-Funktionen sehr sensibel reagieren, wenn Sie irgendein Argument bzw. die Syntax der API-Funktion nicht genau einhalten. Access reagiert dann nicht selten mit einem Programmabsturz.

Wenn Sie im Besitz des Office-Developer-Pakets sind, können Sie mithilfe des API-Viewers gezielt nach bestimmten API-Funktionen suchen. Im anderen Fall empfiehlt sich für dieses Thema weiterführende Literatur.

Da es sich unser Buch vorrangig mit der VBA-Programmierung beschäftigt, werden auf den nächsten Seiten gezielt einige Aufgaben vorgestellt, deren Einsatz in der Programmierung erstens viel Zeit spart und die zweitens in der Praxis gut verwendbar sind. Aber auch interessante Möglichkeiten durch die Verwendung von API-Funktionen sollen nicht zu kurz kommen.

71 Ermittlung des CD-ROM-Laufwerks

In der ersten Aufgabe aus Listing 86 werden Sie über den Einsatz einer API-Funktion herausfinden, welcher Laufwerksbuchstabe Ihrem CD-ROM-Laufwerk zugeordnet ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetDrive
'=====================================================

Declare Function GetDriveType Lib "kernel32" Alias _
   "GetDriveTypeA" (ByVal nDrive As String) As Long

Declare Function GetLogicalDriveStrings Lib _
   "kernel32" Alias "GetLogicalDriveStringsA" _
   (ByVal nBufferLength As Long, _
   ByVal lpBuffer As String) As Long

Public Const DRIVE_CDROM As Long = 5

Function CdRomLWBuchstabe() As String
   Dim lLWTyp As Long
   Dim sLW As String
   Dim l As Long
   Dim l1 As Long
   Dim sBuffer As String

   sBuffer = Space(200)
   l = GetLogicalDriveStrings(200, sBuffer)

   If l = 0 Then
      CdRomLWBuchstabe = vbNullString
      Exit Function
   End If
   l1 = 1
   sLW = Mid(sBuffer, l1, 3)

   Do While (Mid(sBuffer, l1, 1) <> vbNullChar)
      lLWTyp = GetDriveType(sLW)
      If lLWTyp = 5 Then
         CdRomLWBuchstabe = sLW
         Exit Function
      End If
      l1 = l1 + 4
      sLW = Mid(sBuffer, l1, 3)
   Loop
End Function
```

Listing 86: API-Funktion zur Ermittlung des CD-ROM-Laufwerks

Bevor von VBA aus eine Funktion in einer DLL aufgerufen werden kann, müssen Sie VBA mitteilen, wo sich die Funktion befindet und wie sie aufgerufen wird. Dazu verwenden Sie eine `Declare`-Anweisung in einem Modul. Sobald der Verweis festgelegt ist, kann die DLL-Funktion so aufgerufen werden, als sei sie Teil des Projekts. Durch das Schlüsselwort `Lib` wird bestimmt, welche DLL die Funktion enthält. Folgende DLLs sind die am häufigsten verwendeten Bibliotheken in Windows:

- KERNEL32.DLL: Hier finden Sie Betriebssystemfunktionen z. B. für die Speicherverwaltung und die Ressourcenbehandlung.
- USER32.DLL: Diese DLL übernimmt die Fensterverwaltungsfunktionen wie z. B. Meldungen, Menüs, Symbolleisten und Kommunikation.
- GDI32.DLL: Damit ist die GDI-Bibliothek (GDI = Graphics Device Interface) gemeint. Sie enthält die Funktionen für die Geräteausgabe, z. B. Zeichnen, Anzeigen von Kontext und Schriftartenverwaltung.

Im zweiten Schritt schreiben Sie eine Funktion und definieren einen Puffer vom Typ `String`, in dem Sie alle logischen Laufwerksbezeichnungen Ihres Systems einlesen. Danach rufen Sie die API-Funktion `GetLogicalDriveStrings` auf, die alle verwendeten Laufwerksbuchstaben in den Puffer schreibt. Prüfen Sie nach dem Füllen des Puffers gleich einmal die Länge des Puffers. Sind im Puffer `0` Zeichen enthalten, konnte die API-Funktion keine Laufwerksbuchstaben ermitteln. In diesem Fall übergeben Sie der Funktion als Rückgabewert die Konstante `vbNullString`, welche eine Zeichenfolge mit dem Wert `0` darstellt. Für den Fall, dass der Puffer gefüllt ist, setzen Sie die Verarbeitung fort.

Im nächsten Schritt müssen Sie die einzelnen Laufwerke auslesen. Dazu legen Sie die Startposition fest, bei der im Puffer begonnen werden soll, und speichern den Startwert `1` in der Variablen `l`. Zerlegen Sie danach den Puffer, indem Sie die Funktion `Mid` einsetzen. Die Funktion `Mid` überträgt eine bestimmte Anzahl von Zeichen (genau drei, z. B. `a:\`), beginnend bei der Startposition `1`, in die String-Variable `s`. Setzen Sie nun eine Schleife auf, die den Puffer Stück für Stück zerlegt. Innerhalb der Schleife wenden Sie die API-Funktion `GetDriveType` an. Diese API-Funktion liefert den Index für das Laufwerk in der Variablen `sLW` zurück. Jeder Laufwerkstyp weist einen eindeutigen Laufwerksindex auf. Diese Indizes können Sie der folgenden Tabelle entnehmen.

Index	Beschreibung
2	Disketten oder auch ZIP-Laufwerk
3	Lokale Festplatte
4	Netzlaufwerk
5	CD-ROM-Laufwerk
6	RAM-Laufwerk

Tabelle 53: Die Laufwerksindizes für die API-Funktion GetDriveType

Rufen Sie nun die API-Funktion über das Makro aus Listing 87 auf.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetDrive
```

Listing 87: CD-ROM-Laufwerk ermitteln

```
'====================================================

Sub CDRomLW()
   Dim strLW As String
   strLW = CdRomLWBuchstabe()
   MsgBox "Das CD-ROM-Laufwerk hat den Buchstaben " & strLW
End Sub
```

Listing 87: CD-ROM-Laufwerk ermitteln (Forts.)

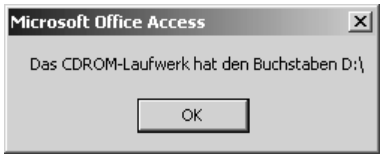


Abbildung 46: Das CD-ROM-Laufwerk wurde ermittelt.

72 Namen des Anwenders ermitteln

Im folgenden Beispiel soll der Name des angemeldeten Benutzers unter Windows ermittelt und ausgegeben werden. Für diesen Zweck greifen Sie auf die Bibliothek ADVAPI32.DLL zurück und nützen die API-Funktion `GetUserName`.

Erfassen Sie jetzt die API-Funktion aus Listing 88.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetUser
'====================================================

Private Declare Function GetUserName Lib _
   "advapi32.dll" Alias "GetUserNameA" _
   (ByVal lpBuffer As String, nSize As Long) As Long

Function AnwenderName() As String
   Dim l As Long
   Dim s As String
   Const D_Puffer = 255
   s = Space(D_Puffer)
   l = D_Puffer

   If CBool(GetUserName(s, l)) Then
      AnwenderName = Left$(s, l - 1)
   Else
      AnwenderName = ""
   End If
End Function
```

Listing 88: Anwendernamen ermitteln

```
Sub WerBinIch()
 MsgBox "Der Anwender " & AnwenderName & " ist angemeldet!"
End Sub
```

Listing 88: Anwendernamen ermitteln (Forts.)

Rufen Sie über eine Prozedur die Funktion `Anwendername` auf. In dieser Funktion wird ein leerer Datenpuffer erzeugt. Danach rufen Sie die API-Funktion `GetUserName` auf und überprüfen, ob ein Anwendername ermittelt werden konnte. Um eventuell mitgeschleppte Leerzeichen abzuschneiden, setzen Sie die Funktion `Left` ein und übertragen den ermittelten Anwendernamen. Da die API-Funktion ein Zeichen zu viel überträgt, subtrahieren Sie dieses von der Gesamtlänge des ermittelten Anwendernamens.

Abbildung 47: Anwendernamen auslesen

73 Computernamen ermitteln

Den Computernamen können Sie in der Systemsteuerung von Windows unter NETZWERK herausfinden. Schneller geht es aber mit der API-Funktion aus Listing 89.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetComp
'=====================================================

Private Declare Function GetComputerName Lib _
   "kernel32" Alias "GetComputerNameA" _
   (ByVal lpBuffer As String, nSize As Long) As Long

Sub ComputerNameAnzeigen()
   Dim strString As String

   strString = String(255, Chr$(0))
   GetComputerName strString, 255
   MsgBox "Der Computer heißt: " & strString
End Sub
```

Listing 89: Computernamen ausgeben

Über die API-Funktion `GetComputerName` können Sie den Computernamen ermitteln.

Abbildung 48: Computernamen auslesen

74 Bedienung des CD-ROM-Laufwerks

Haben Sie sich schon einmal überlegt, dass Sie Ihr CD-ROM-Laufwerk auch gut als Tassenhalter verwenden können? Dazu müssen Sie Ihr CD-ROM-Laufwerk nur öffnen und schon haben Sie eine clevere Ablagemöglichkeit mehr. Diese nicht ganz ernst gemeinte Anwendung können Sie automatisieren, indem Sie Ihr CD-ROM-Laufwerk auf Kommando öffnen bzw. schließen. Die API-Funktion für diese Aufgabe heißt `mciSendString`.

Öffnen bzw. schließen Sie mithilfe der folgenden API-Funktion aus Listing 90 das CD-ROM-Laufwerk.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlCDROM
'=====================================================

Public Declare Function mciSendString _
   Lib "winmm.dll" Alias "mciSendStringA" _
   (ByVal lpstrCommand As String, _
   ByVal lpstrReturnString As String, _
   ByVal uReturnLength As Long, _
   ByVal hwndCallback As Long) As Long

Sub ÖffnenCDROMLaufwerk()
   mciSendString "Set CDAudio Door Open", 0&, 0, 0
End Sub

Sub SchließenCDROMLaufwerk()
   mciSendString "Set CDAudio Door Closed", 0&, 0, 0
End Sub
```

Listing 90: CD-ROM-Laufwerk ansprechen

75 Die Bildschirmauflösung ermitteln

Möchten Sie prüfen, mit welcher Bildschirmauflösung ein Anwender arbeitet, dann setzen Sie die API-Funktion aus Listing 91 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetDev
'=====================================================

Declare Function GetDeviceCaps Lib "gdi32" _
   (ByVal hdc As Long, ByVal nIndex As Long) As Long

Declare Function GetDC Lib "user32" _
   (ByVal hwnd As Long) As Long

Declare Function ReleaseDC Lib "user32" _
  (ByVal hwnd As Long, ByVal hdc As Long) As Long

Const HORZRES = 8
Const VERTRES = 10

Sub BildschirmAuflösungErmitteln()
   MsgBox "Ihre Bildschirmauflösung lautet:" & _
   Chr(13) & BilschirmAuflösung()
End Sub

Function BilschirmAuflösung()
   Dim lRval As Long
   Dim lDc As Long
   Dim lHSize As Long
   Dim lVSize As Long

   lDc = GetDC(0&)
   lHSize = GetDeviceCaps(lDc, HORZRES)
   lVSize = GetDeviceCaps(lDc, VERTRES)
   lRval = ReleaseDC(0, lDc)
   BilschirmAuflösung = lHSize & "x" & lVSize
End Function
```

Listing 91: Bildschirmauflösung ermitteln

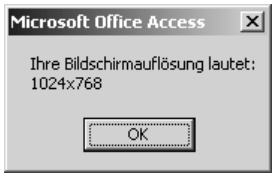


Abbildung 49: Bildschirmauflösung ermitteln

76 Ist ein externes Programm gestartet?

Im nächsten Beispiel prüfen Sie, ob Ihr E-Mail-Programm OUTLOOK bereits gestartet ist. Die API-Funktion für diese Aufgabe finden Sie in Listing 92.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlFindW
'=====================================================

Private Declare Function FindWindow Lib "user32" _
   Alias "FindWindowA" (ByVal szClass$, ByVal szTitle$) _
   As Long

Sub MailProgrammAktiv()
   Dim hfenster As String

   hfenster = FindWindow(vbNullString, "Microsoft Outlook")
   If hfenster = 0 Then
      MsgBox "Outlook nicht aktiv!"
   Else
      MsgBox ("Outlook gestartet!")
   End If
End Sub
```

Listing 92: Ist ein externes Programm momentan geöffnet?

Über die API-Funktion `FindWindow` können Sie prüfen, ob eine andere Anwendung bereits gestartet ist. Ist dies der Fall, dann wird Ihnen ein Wert <> `0` zurückgeliefert.

77 Externes Programm aufrufen

Um externe Programme aufzurufen, setzen Sie die API-Funktion `ShellExecute` ein. Im folgenden Makro aus Listing 93 wird beispielsweise der Windows-Rechner aufgerufen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlShell
'=====================================================

Private Declare Function ShellExecute Lib _
   "shell32.dll" Alias "ShellExecuteA" _
   (ByVal hwnd As Long, ByVal lpOperation As String, _
   ByVal lpFile As String, ByVal lpParameters As String, _
   ByVal lpDirectory As String, ByVal bShowCMD As Long) _
   As Long

Sub RechnerAufrufen()
   ShellExecute hwnd, "open", "calc.exe", 0, 0, SW_SHOW
End Sub
```

Listing 93: Ein externes Programm starten

Übergeben Sie der API-Funktion unter anderem den Namen der Anwendung, die Sie starten möchten.

Abbildung 50: Externes Programm aufrufen

78 Wie lange läuft ein externes Programm?

Im nächsten Beispiel soll ermittelt werden, wie lange ein externes Programm läuft, und der Start- und Endezeitpunkt im Direktfenster dokumentiert werden. Nehmen wir beispielsweise an, Sie möchten protokollieren, wann und wie lange Sie das Spiel »Solitär« gespielt haben. Dazu setzen Sie die API-Funktionen `OpenProcess` sowie `GetExitCodeProcess` ein. Die eine API-Funktion aus Listing 94 eröffnet und überwacht den Start einer externen Anwendung, die andere meldet das Beenden einer Anwendung zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      Mdl0penProc
'=====================================================

Declare Function OpenProcess Lib "kernel32" _
  (ByVal dwDesiredAccess As Long, _
   ByVal bInheritHandle As Long, _
   ByVal dwProcessId As Long) As Long

 Declare Function GetExitCodeProcess Lib "kernel32" _
   (ByVal L_Prozess As Long, _
    l_Ende As Long) As Long

Public Const PROCESS_QUERY_INFORMATION = &H400
Public Const STILL_ACTIVE = &H103

Sub StartenExternerAnwendung()
   Dim Beginn As Double
```

Listing 94: Wie lange läuft ein gestartetes Programm?

```
    Dim l As Long
    Dim L_Prozess As Long
    Dim l_Ende As Long

    Beginn = Now
    Debug.Print "Start des Programms um: " & Now
    l = Shell("sol.exe", 1)
    L_Prozess = _
    OpenProcess(PROCESS_QUERY_INFORMATION, False, l)
    MsgBox "Zeitnahme läuft..."
    Do While l_Ende = STILL_ACTIVE
       GetExitCodeProcess L_Prozess, l_Ende
       DoEvents
    Loop
    Ende = Now
    Debug.Print "Die Anwendung wurde um " & Now & " beendet!"
    Debug.Print "Das Programm lief genau: " & Format(Ende-Beginn, "s") & " Sekunden!"
End Sub
```

Listing 94: Wie lange läuft ein gestartetes Programm? (Forts.)

Speichern Sie gleich zu Beginn des Makros die aktuelle Uhrzeit mithilfe der Funktion `Now` in der Variablen `Beginn`. Danach schreiben Sie diesen Startzeitpunkt in das Direktfenster. Als Nächstes setzen Sie die Funktion `Shell` ein, um das externe Programm zu starten. Dabei haben Sie die Möglichkeit, als Argument die Anordnung der externen Anwendung auf dem Bildschirm zu bestimmen. Entnehmen Sie diese verschiedenen Indizes und deren Bedeutung der Tabelle 54.

Index	Beschreibung
0	Das Fenster der aufgerufenen Anwendung ist ausgeblendet und das ausgeblendete Fenster erhält den Fokus.
1	Das Fenster hat den Fokus und die ursprüngliche Größe und Position werden wiederhergestellt.
2	Das Fenster wird als Symbol mit Fokus angezeigt.
3	Das Fenster wird maximiert (Vollbild) mit Fokus angezeigt.
4	Die zuletzt verwendete Größe und Position des Fensters wird wiederhergestellt. Das momentan aktive Fenster bleibt aktiv.
6	Das Fenster wird als Symbol angezeigt. Das momentan aktive Fenster bleibt aktiv.

Tabelle 54: Die Möglichkeiten bei der Fensteranordnung von Programmen

Rufen Sie jetzt die erste API-Funktion `OpenProcess` auf, die den aktiven Prozess (im Beispiel das Spiel »Solitär«) verwaltet. Den Rückgabewert dieser Funktion speichern Sie in einer Variablen vom Datentyp `Long`. Setzen Sie nun eine Schleife auf, die so lange abgearbeitet wird, bis die API-Funktion `GetExitCodeProcess` die Beendigung des externen Programms meldet. Mithilfe der Funktion `DoEvents` übergeben Sie die Steuerung an das Betriebssystem, damit andere Ereignisse verarbeitet werden können. Somit läuft diese Schleife im Hinter-

grund weiter und wird erst beendet, wenn der Prozess beendet ist. Nach Beendigung der externen Anwendung schreiben Sie die Laufzeit des externen Programms in das Direktfenster. Die Laufzeit ermitteln Sie, indem Sie wiederum die Funktion `Now` heranziehen und diese Zeit mit der vorher gespeicherten Startzeit in der Variablen `Beginn` abgleichen.

79 Access schlafen schicken

Möchten Sie eine Anwendung ein wenig zum Schlafen schicken, dann können Sie die API-Funktion aus Listing 95 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlSleep
'=====================================================

Declare Sub Sleep Lib "Kernel32.dll" _
   (ByVal SleepTime As Long)

Sub AccessZumSchlafenSchicken()
   Sleep (10000)
   MsgBox "Ruhepause beendet"
End Sub
```

Listing 95: API-Funktion zum »Schlafenlegen« einer Anwendung

Die Zeitangabe ist in Millisekunden angegeben.

80 Verzeichnisse erstellen

Neben der Standard-VBA-Funktion `MkDir`, um einen neuen Ordner anzulegen, gibt es auch hierfür eine eigene API-Funktion. Über API-Funktionen haben Sie ganz allgemein Zugriff auf alle Ordner und Laufwerke Ihres Betriebssystems. So können Sie auch Verzeichnisse erstellen und löschen. In Listing 96 wird nach einer Benutzereingabe das gewünschte Verzeichnis erstellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlCreateD
'=====================================================

Private Declare Function CreateDirectory Lib _
   "kernel32" Alias "CreateDirectoryA" _
   (ByVal lpPathName As String, _
   lpSecurityAttributes As SECURITY_ATTRIBUTES) As Long
```

Listing 96: Verzeichnis erstellen mit einer API-Funktion

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

Sub VerzeichnisErstellen()
   Dim Security As SECURITY_ATTRIBUTES
   Dim l As Long
   Dim strV As String

   strV = InputBox("Geben Sie das Verzeichnis ein", _
      "Verzeichnis erstellen", "c:\")
   If strV = "" Then Exit Sub

   l = CreateDirectory(strV, Security)
   If l = 0 Then
      MsgBox "Das Verzeichnis konnte nicht erstellt werden!", _
      vbCritical + vbOKOnly
   End If
End Sub
```

Listing 96: Verzeichnis erstellen mit einer API-Funktion (Forts.)

Mit der API-Funktion `CreateDirectory` können Sie eigene Verzeichnisse auf Ihrer Festplatte anlegen.

81 Verzeichnis löschen

Um ein Verzeichnis zu entfernen, können Sie die API-Funktion `RemoveDirectory` aus Listing 97 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlRemoveD
'=====================================================

Private Declare Function RemoveDirectory Lib _
   "kernel32" Alias "RemoveDirectoryA" _
   (ByVal lpPathName As String) As Long

Sub VerzeichnisLöschen()
   Dim lngV As Long
   lngV = RemoveDirectory("C:\Test")
End Sub
```

Listing 97: Verzeichnis löschen mit einer API-Funktion

82 Verzeichnisbaum anzeigen und auswerten

Im folgenden Beispiel aus Listing 3.98 wird Ihre Verzeichnisstruktur in einem Dialogfeld als Verzeichnisbaum angezeigt und ausgewertet.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetPath
'====================================================

Type BROWSEINFO
   hOwner As Long
   pidlRoot As Long
   pszDisplayName As String
   lpszTitle As String
   ulFlags As Long
   lpfn As Long
   lParam As Long
   iImage As Long
End Type

Declare Function SHGetPathFromIDList Lib "shell32.dll" _
   Alias "SHGetPathFromIDListA" _
   (ByVal pidl As Long, ByVal pszPath As _
   String) As Long
Declare Function SHBrowseForFolder Lib "shell32.dll" _
   Alias "SHBrowseForFolderA" _
   (lpBrowseInfo As BROWSEINFO) As Long
Declare Function FindWindow Lib "user32" _
   Alias "FindWindowA" (ByVal _
   lpClassName As String, _
   ByVal lpWindowName As String) As Long

Function VerzeichnisErmitteln(Msg) As String
   Dim bInfo As BROWSEINFO
   Dim path As String
   Dim l As Long

   bInfo.pidlRoot = 0&
   l = SHBrowseForFolder(bInfo)
   path = Space$(512)
   If SHGetPathFromIDList(ByVal l, ByVal path) Then
      VerzeichnisErmitteln = Left(path, InStr(path, Chr$(0)) - 1)
   Else
      VerzeichnisErmitteln = ""
   End If
End Function

Sub VerzeichnisBaum()
```

Listing 98: Verzeichnis auswählen und abfragen

```
    Dim sVerz As String

    sVerz = VerzeichnisErmitteln(sVerz)
    If sVerz = "" Then Exit Sub
    MsgBox "Das ausgewählte Verzeichnis lautet: " & sVerz
End Sub
```

Listing 98: Verzeichnis auswählen und abfragen (Forts.)

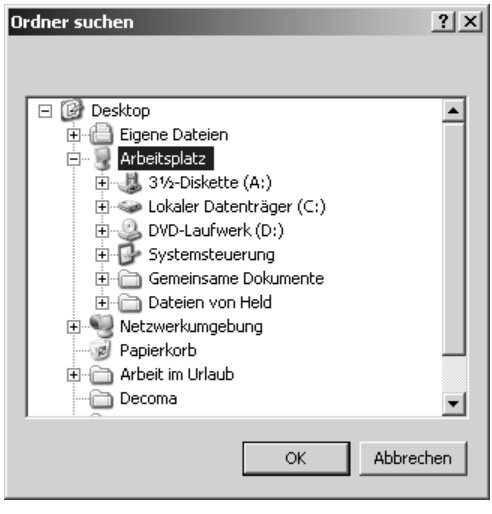


Abbildung 51: Den Verzeichnisbaum anzeigen

Bei der Anweisung `bInfo.pidlRoot = 0&` kann man die Zahl jeweils erhöhen, was folgende Effekte hat.

`1` --> Internet Explorer

`2` --> Programme

`3` --> Systemsteuerung

`4` --> Drucker

`5` --> Eigene Dateien

`6` --> Favoriten

`7` --> Autostart

`8` --> Recent

`9` --> SendTo

`10`--> Papierkorb

83 Windows-Version ermitteln

Um die installierte Windows-Version zu ermitteln, können Sie mit der API-Funktion `GetVersionEx` aus Listing 99 arbeiten.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetVersion
'====================================================

Private Type OSVERSIONINFO
   dwOSVersionInfoSize As Long
   dwMajorVersion As Long
   dwMinorVersion As Long
   dwBuildNumber As Long
   dwPlatformId As Long
   szCSDVersion As String * 128
End Type

Private Const VER_PLATFORM_WIN32s = 0
Private Const VER_PLATFORM_WIN32_WINDOWS = 1
Private Const VER_PLATFORM_WIN32_NT = 2

Private Declare Function GetVersionEx Lib "kernel32" _
   Alias "GetVersionExA" _
   (lpVersionInformation As OSVERSIONINFO) As Long

Sub WindowsVersion()
   Dim osInfo As OSVERSIONINFO
   Dim lRet As Long

   osInfo.dwOSVersionInfoSize = 148
   osInfo.szCSDVersion = Space(128)

   lRet = GetVersionEx(osInfo)

   Select Case osInfo.dwPlatformId
      Case VER_PLATFORM_WIN32s
         Debug.Print "Win32s on Windows 3.1"
      Case VER_PLATFORM_WIN32_WINDOWS
         If osInfo.dwMinorVersion = 0 Then
            Debug.Print "Win32 mit Windows 95"
         Else
            Debug.Print "Win32 mit Windows 98"
         End If
      Case VER_PLATFORM_WIN32_NT
         If osInfo.dwMajorVersion = 5 And osInfo.dwMinorVersion = 0 Then
            Debug.Print "Win32 mit Windows 2000"
         Else
```

Listing 99: Die installierte Windows-Version erkennen

```
            If osInfo.dwMajorVersion = 5 And osInfo.dwMinorVersion >= 1 Then
               Debug.Print "Win32 mit Windows XP"
            Else
               Debug.Print "Win32 mit Windows NT"
            End If
         End If
      Case Else
         Debug.Print "Betriebssystem unbekannt!"
   End Select

   Debug.Print "Version: " & osInfo.dwMajorVersion & "." & osInfo.dwMinorVersion
   Debug.Print "Build: " & osInfo.dwBuildNumber
   Debug.Print "Info: " & osInfo.szCSDVersion
End Sub
```

Listing 99: Die installierte Windows-Version erkennen (Forts.)

Die neueren Windows-Versionen werden über die `dwplatformId` mit dem Wert 2 interpretiert. Darunter werten Sie die Argumente `dwMinorVersion` und `dwMajorVersion` aus, um das Betriebssystem zu bestimmen.

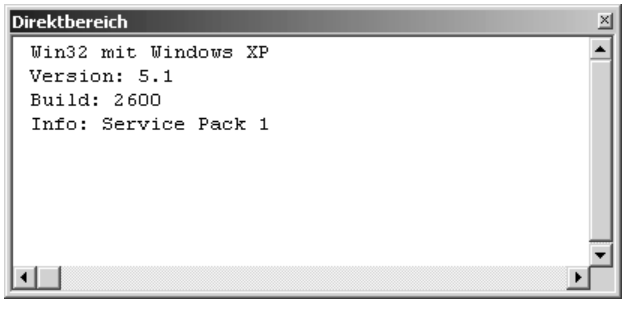


Abbildung 52: Die Windows-Version ermitteln

84 Windows-Verzeichnis ermitteln

Da es mehrere Windows-Versionen gibt und diese sich namentlich auch unterscheiden können, ist es wichtig zu wissen, wie der exakte Namen des Windows-Verzeichnisses lautet. Für diese Aufgabe können Sie folgende API-Funktion aus Listing 100 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetWinD
'=====================================================

Declare Function GetWindowsDirectory Lib _
```

Listing 100: Windows-Verzeichnis ermitteln

```
    "kernel32" Alias "GetWindowsDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

 Function GetWinDir() As String
    Dim lpBuffer As String * 255
    Dim length As Long

    length = _
    GetWindowsDirectory(lpBuffer, Len(lpBuffer))
    GetWinDir = Left(lpBuffer, length)
 End Function

 Sub WindowsVerzeichnis()
    Dim t As String
    t = GetWinDir
    MsgBox "Ihr Windows-Verzeichnis heißt: " & t
 End Sub
```

Listing 100: Windows-Verzeichnis ermitteln (Forts.)

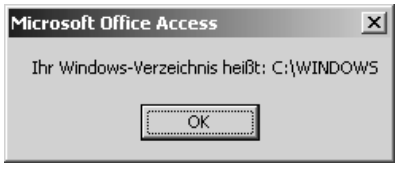


Abbildung 53: Windows-Installationsverzeichnis auslesen

85 Windows-Systemverzeichnis ermitteln

Soll das Windows-System-Verzeichnis ermittelt werden, dann setzen Sie die API-Funktion `GetSystemDirectory` aus Listing 101 ein.

```
 '=====================================================
 ' Auf CD     Buchdaten\Beispiele\Kap02
 ' Dateiname  WeitereFunktionen.mdb
 ' Modul      MdlGetSystemD
 '=====================================================

 Declare Function GetSystemDirectory Lib _
    "kernel32" Alias "GetSystemDirectoryA" _
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

 Function GetSystemDir() As String
    Dim lpBuffer As String * 255
    Dim length As Long
    length = _
    GetSystemDirectory(lpBuffer, Len(lpBuffer))
```

Listing 101: Windows-Systemverzeichnis ermitteln

```
    GetSystemDir = Left(lpBuffer, length)
End Function

Sub WindowsSystemVerzeichnis()
    Dim strV As String
    strV = GetSystemDir
    MsgBox "Ihr Windows-System-Verzeichnis heißt: " & strV
End Sub
```

Listing 101: Windows-Systemverzeichnis ermitteln (Forts.)

Abbildung 54: Das Windows-System-Verzeichnis ermitteln

86 Das temporäre Verzeichnis ermitteln

Windows benötigt ein Verzeichnis, in dem es temporäre Dateien ablegt, die bei Bedarf geladen bzw. wieder geschlossen werden. Um dieses temporäre Verzeichnis aufzuspüren, setzen Sie die API-Funktion `GetTempPath` aus Listing 102 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetTempP
'=====================================================

Declare Function GetTempPath Lib _
    "kernel32" Alias "GetTempPathA" _
    (ByVal nSize As Long, ByVal lpBuffer As String) As Long

Function GetTempDir() As String
    Dim lpBuffer As String * 255
    Dim length As Long

    length = _
    GetTempPath(Len(lpBuffer), lpBuffer)
    GetTempDir = Left(lpBuffer, length)
End Function

Sub TemporäresVerzeichnis()
    Dim strV As String
```

Listing 102: Das temporäre Verzeichnis ermitteln

```
    strV = GetTempDir
    MsgBox "Ihr temporäres Verzeichnis heißt: " & strV
End Sub
```

Listing 102: Das temporäre Verzeichnis ermitteln (Forts.)

Abbildung 55: Das temporäre Verzeichnis von Windows ermitteln

87 Das aktuelle Verzeichnis ermitteln

Neben der API-Funktion `GetWindowsDirekctory` gibt es weitere API-Funktionen, um bestimmte Verzeichnisse zu ermitteln. Mithilfe der Funktion `GetCurrentDirectory` aus Listing 103 können Sie ermitteln, welches Verzeichnis gerade aktiv ist.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetCurrD
'====================================================

Declare Function GetCurrentDirectory Lib _
   "kernel32" Alias "GetCurrentDirectoryA" _
   (ByVal nSize As Long, ByVal lpBuffer As String) As Long

Function GetAktDir() As String
   Dim lpBuffer As String * 255
   Dim length As Long

length = _
   GetCurrentDirectory(Len(lpBuffer), lpBuffer)
   GetAktDir = Left(lpBuffer, length)
End Function

Sub AktuellesVerzeichnis()
Dim t As String
t = GetAktDir
MsgBox "Ihr  aktuelles Verzeichnis heißt: " & t
End Sub
```

Listing 103: Das aktuelle Verzeichnis ermitteln

Abbildung 56: Das aktuelle Verzeichnis ermitteln

88 Windows-Infobildschirm anzeigen

Möchten Sie einen Info-Bildschirm in Excel anzeigen, dann können Sie dazu eine API-Funktion aus Listing 104 einsetzen. Das spart Ihnen das Entwerfen und Programmieren eines eigenen Formulars. Die API-Funktion für diesen Zweck heißt `ShellAbout`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlShellA
'=====================================================

Private Declare Function ShellAbout Lib "shell32.dll" _
   Alias "ShellAboutA" (ByVal hwnd As Long, _
   ByVal szApp As String, ByVal szOtherStuff As String, _
   ByVal hIcon As Long) As Long

Sub BildschirmAnzeigen()
   Dim hwnd As Long
   Dim lngSymbol As Long

   ShellAbout hwnd, "Held-Office", _
      "Bernd Held, MVP für Microsoft Excel" & vbCrLf & _
      "http://held-office.de", lngSymbol
End Sub
```

Listing 104: API-Funktion zum Anzeigen eines Infoschirms

89 Access-Verzeichnis ermitteln

Sie können Access auch starten, indem Sie auf Ihrem Windows-Desktop die Schaltfläche START anklicken und dann den Befehl AUSFÜHREN wählen. Im Dialogfeld AUSFÜHREN klicken Sie auf die Schaltfläche DURCHSUCHEN, stellen Ihr Office-Verzeichnis ein und markieren die Datei *MSACCESS.exe*.

Um genau diesen Pfad zur ermitteln, können Sie die API-Funktion aus Listing 105 verwenden.

Abbildung 57: Windows-Infoschirm manipulieren

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetCommandL
'=====================================================

Declare Function GetCommandLine Lib "Kernel32" _
   Alias "GetCommandLineA" () As String

Sub VerzeichnisAccessAnsteuern()
   MsgBox "Access befindet sich unter: " & GetCommandLine()
End Sub
```

Listing 105: Ermitteln des Access-Startverzeichnisses

Die API-Funktion besteht lediglich aus einem Kommando, welches Sie durch ein Makro aufrufen und über die Funktion `MsgBox` auf dem Bildschirm ausgeben.

90 Standardverzeichnis festlegen

Standardmäßig ist das Verzeichnis *Eigene Dateien* Ihr Standardverzeichnis in Access. Hier werden Access-Datenbanken standardmäßig gespeichert bzw. beim Dialog ÖFFNEN wird immer dieses Verzeichnis eingestellt. Diese Einstellung können Sie ändern, indem Sie die API-Funktion aus Listing 106 einsetzen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlSetCurrentD
'=================================================

Private Declare Function SetCurrentDirectoryA _
   Lib "kernel32" (ByVal lpPathName As String) As Long


Sub VerzeichnisFestlegenÜberAPI()
   SetCurrentDirectoryA ("c:\temp")
End Sub
```

Listing 106: Standardverzeichnis festlegen

91 Dateityp und Anwendung ermitteln

Im nächsten Beispiel soll ausgehend von einer bestimmten Datei die dazugehörige Anwendung ermittelt werden. Den Code für diese Aufgabe finden Sie in Listing 107.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlFindEx
'=================================================

Private Declare Function FindExecutable Lib "shell32.dll" _
   Alias "FindExecutableA" (ByVal lpFile As String, _
   ByVal lpDirectory As String, _
   ByVal lpResult As String) As Long

Public Function AppFile(Datei As String) As String
   Dim strT As String
   strT = Space(256)
   FindExecutable Datei, vbNullString, strT
   AppFile = Left$(strT, InStr(strT, vbNullChar) - 1)
End Function

Sub Datei()
   Dim strT As String

   strT = AppFile("c:\Eigene Dateien\Umsatz.xls")
   If strT <> "" Then
      MsgBox "Die dazugehörige Anwendung lautet: " & vbLf & strT
   Else
      MsgBox "Anwendung konnte nicht ermittelt werden!", vbCritical
   End If
End Sub
```

Listing 107: Die zur Datei gehörende Anwendung ermitteln

Wenn Sie eine Datei mitsamt dem Pfad angeben, liefert `FindExecutable` den kompletten Pfad zur verknüpften Anwendung zurück, der in einem String über den Parameter `lpResult` übergeben wird. Existiert die angegebene Datei nicht, so wird ein Leerstring zurückgegeben.

Abbildung 58: Die dazugehörige Anwendung wurde aufgespürt.

92 Kurze Pfadnamen ermitteln

Bei sehr ausgeprägten Verzeichnisstrukturen wird der Pfad in Windows oft umgebrochen. Die API-Funktion, die dafür verantwortlich ist, heißt `GetShortPathName`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetShort
'=====================================================

Declare Function GetShortPathName Lib "kernel32" _
   Alias "GetShortPathNameA" _
   (ByVal lpszLongPath As String, ByVal lpszShortPath As String, _
   ByVal cchBuffer As Long) As Long

Sub KurzerDateiname()
   Dim strLangerName As String
   Dim strKurzerName As String
   Dim lngl As Long

   strLangerName = Application.CurrentDb.Name
   strKurzerName = Space$(250)

   lngl = GetShortPathName(strLangerName, strKurzerName, Len(strKurzerName))

   strKurzerName = Left$(strKurzerName, lngl)
   MsgBox " Kurz: " & strKurzerName & vbLf & _
      "Lang: " & Application.CurrentDb.Name, vbInformation
End Sub
```

Listing 108: Die gekürzte Pfadangabe ausgeben

Über die Anweisung `Application.CurrentDb.Name` ermitteln Sie den Dateinamen inklusive des Pfads der aktuell geöffneten Datenbank. Die Variable `KurzerName` wird mit 250 Leerzeichen zunächst initialisiert. Danach wird die API-Funktion `GetShortPathName` aufgerufen. Dieser Funktion wird sowohl die Variable `LangerName` als auch die noch leere Variable `KurzerName`

sowie deren Länge übergeben. Die API-Funktion kürzt den Pfadnamen, den Sie danach über die Funktion `MsgBox` auf dem Bildschirm ausgeben.

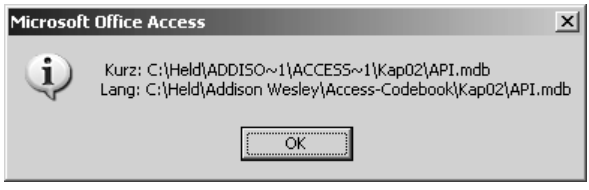


Abbildung 59: Kurze und lange Dateinamen auslesen

93 Texte mit API-Funktionen konvertieren

Es ist möglich, mithilfe von API-Funktionen auch Texte zu konvertieren. So konvertiert die API-Funktion `CharUpper` einen Text in Großschreibweise und entspricht dabei der VBA-Funktion `UCase`. Die API-Funktion `CharLower` konvertiert einen vorgegebenen Text in Kleinschreibweise und findet ihre Entsprechung in der VBA-Funktion `LCase`. Sehen Sie sich dazu die API-Funktion aus Listing 109 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlChartLower
'=====================================================

Private Declare Function CharLower Lib "user32" _
   Alias "CharLowerA" (ByVal lpsz As String) As Long

Private Declare Function CharUpper Lib "user32" _
   Alias "CharUpperA" (ByVal lpsz As String) As Long


Sub TexteUmsetzen()
   Dim strText As String

   strText = "API-Funktionen sind o.k.!"
   Debug.Print "Originalzustand: " + strText
   CharUpper strText
   Debug.Print "GROSS: " + strText
   CharLower strText
   Debug.Print "klein: " + strText
End Sub
```

Listing 109: Texte konvertieren

Das Ergebnis der Konvertierungsfunktionen wird im Direktbereich ausgegeben.

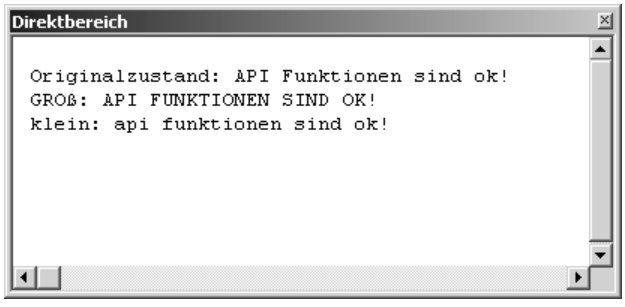


Abbildung 60: Texte konvertieren über eine API-Funktion

94 Zwischenablage löschen

Möchten Sie die Zwischenablage durch eine API-Funktion löschen, verwenden Sie die API-Funktion aus Listing 110

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlOpenClip
'=====================================================

Private Declare Function OpenClipboard Lib "user32" _
   (ByVal hwnd As Long) As Long

Private Declare Function EmptyClipboard Lib "user32" () As Long

Private Declare Function CloseClipboard Lib "user32" () As Long

Sub ZwischenablageLeeren()
   If OpenClipboard(0&) <> 0 Then
      Call EmptyClipboard
      Call CloseClipboard
   End If
End Sub
```

Listing 110: Zwischenablage löschen

Befindet sich etwas in der Zwischenablage, dann meldet die API-Funktion `OpenClipboard` einen Wert `<> 0`. In diesem Fall rufen Sie die beiden API-Funktionen nacheinander auf. Die `OpenClipboard`-Funktion wird dazu verwendet, die Zwischenlage zu öffnen, zu prüfen und zu verhindern, dass ihr Inhalt durch andere Anwendungen geändert wird. Mit der `GetClipboardData`-Funktion werden die in der Zwischenablage gespeicherten Daten zurückgegeben, während die `CloseClipboard`-Funktion dazu dient, die Zwischenablage zu schließen und sie für andere Anwendungen wieder verfügbar zu machen.

95 Soundkarte checken

Bevor Sie Klangdateien auf Ihrem Computer abspielen können, führen Sie zunächst einen Test durch, ob überhaupt eine Soundkarte auf Ihrem System verfügbar ist. Für diese Aufgabe können Sie die API-Funktion `WaveOutGetNumDevs` aus Listing 111 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWaveOut
'=====================================================

Private Declare Function waveOutGetNumDevs Lib _ "winmm.dll" () As Long

Function PrüfenSound() As Boolean
   PrüfenSound = waveOutGetNumDevs()
End Function

Sub SoundMöglich()
   Dim b As Boolean

   If PrüfenSound = True Then
      MsgBox "Soundkarte verfügbar!", vbInformation
   Else
      MsgBox "Soundkarte verfügbar!", vbInformation
   End If
End Sub
```

Listing 111: Überprüfen, ob Soundkarte einsatzbereit ist

96 Sounds per API-Funktion ausgeben

Mithilfe von API-Funktionen können Sie auch Ihre Soundkarte ansprechen. Mit der API-Funktion aus Listing 112 spielen Sie eine WAV-Datei ab.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlPlaySound
'=====================================================

Declare Function sndPlaySound32 Lib "winmm.dll" Alias _
   "sndPlaySoundA" (ByVal lpszSoundName As String, _
   ByVal uFlags As Long) As Long

Sub SoundAusgeben()
   Call sndPlaySound32("c:\APPLAUSE.WAV", 0)
End Sub
```

Listing 112: Sounddatei abspielen

97 PC piepsen lassen

Soll Ihr PC nur einen Piepston abgeben, dann setzen Sie die API-Funktion `Beep` aus Listing 113 ein.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlBeep
'====================================================

Declare Function Beep Lib "kernel32.dll"  _
   (ByVal dwFreq As Long, ByVal dwDuration As Long) As Long

Sub ComputerPiepst()
   Dim L As Long

   L = Beep(250, 1000)

   If L = 0 Then
      MsgBox "PC kann nicht piepsen!"
   End If
End Sub
```

Listing 113: PC piepsen lassen

Die API-Funktion benötigt zwei Parameter. Im ersten Argument legen Sie fest, mit wie viel Herz der Piepston erzeugt werden soll. Mögliche noch hörbare Werte liegen im Bereich zwischen 37 und 5500. Im zweiten Argument legen Sie die Dauer des Piepstons in Millisekunden fest.

98 Tasten abfangen

Durch den Einsatz von API-Funktionen können Sie bestimmte Tasten abfangen. Im folgenden Beispiel aus Listing 114 wird die Taste Esc überwacht.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlKeyboard
'====================================================

Type KeyboardBytes
   kbb(0 To 255) As Byte
End Type

Declare Function GetKeyboardState Lib "User32.dll" _
   (kbArray As KeyboardBytes) As Long
```

Listing 114: API-Funktion zum Abfangen von Tastenklicks

```
Sub TasteESCabfangen()
   Dim kbArray As KeyboardBytes

   Do
      DoEvents
      GetKeyboardState kbArray
      If kbArray.kbb(27) And 128 Then
         ESCgedrückt
      End If
      ' Makro beenden mit STRG
   Loop Until kbArray.kbb(17) And 128
End Sub

Sub ESCgedrückt()
   MsgBox "Sie haben die Taste ESC gedrückt"
End Sub
```

Listing 114: API-Funktion zum Abfangen von Tastenklicks (Forts.)

Starten Sie das Makro `TasteESCAbfangen`. Dieses Programm läuft im Hintergrund ab und gibt jedes Mal eine Bildschirmmeldung aus, wenn Sie die Taste Esc drücken. Drücken Sie die Taste Strg, um dieses Makro wieder zu beenden und die Überwachung der Taste Esc abzuschließen.

99 Dateien suchen

Mithilfe der API-Funktion `SearchTreeForFile` aus Listing 115 können Sie auch Dateien suchen und deren Speicherpfad ermitteln.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlSearch
'=====================================================

Private Declare Function SearchTreeForFile _
   Lib "imagehlp" (ByVal RootPath As String, _
   ByVal InputPathName As String, _
   ByVal OutputPathBuffer As String) As Long

Const Verzeichnisse = 100

Sub DateiSuchen()
   Dim strS As String
   Dim lngL As Long
   strS = String(Verzeichnisse, 0)
```

Listing 115: Datei suchen über eine API-Funktion

```
    lngL = SearchTreeForFile("C:\", "Mappe1.xls", strS)
    If lngL <> 0 Then
       MsgBox "Datei wurde gefunden: " & vbLf & strS, vbInformation
    Else
       MsgBox "Datei konnte nicht gefunden werden!", vbCritical
    End If
End Sub
```

Listing 115: Datei suchen über eine API-Funktion (Forts.)

Abbildung 61: Dateien suchen

100 Dateiinformationen auslesen

Mithilfe von API-Funktionen lassen sich Datumsinformationen zu Dateien ermitteln. So wird im folgenden Beispiel aus Listing 116 das Erstellungsdatum, das letzte Zugriffsdatum sowie das letzte Speicherdatum einer Datenbank ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlFindFirst
'=====================================================

Public Declare Function FindFirstFile Lib _
   "kernel32" Alias "FindFirstFileA" _
   (ByVal lpFileName As String, _
   lpFindFileData As WIN32_FIND_DATA) As Long

Public Declare Function FileTimeToSystemTime _
   Lib "kernel32" (lpFileTime As FILETIME, _
   lpSystemTime As SYSTEMTIME) As Long

Declare Function FileTimeToLocalFileTime _
   Lib "kernel32" (lpFileTime As FILETIME, _
   lpLocalFileTime As FILETIME) As Long

Public Type FILETIME
   dwLowDateTime As Long
   dwHighDateTime As Long
End Type
Public Type SYSTEMTIME
```

Listing 116: Dateiinformationen auslesen

```
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Long
End Type

Public Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    ftName As String * 150
End Type

Private Function DatPrüf(FT As FILETIME) As String
    Dim ST As SYSTEMTIME
    Dim LT As FILETIME
    Dim t As Long
    Dim lDatum As Double
    Dim lZeit As Double

    t = FileTimeToLocalFileTime(FT, LT)
    t = FileTimeToSystemTime(LT, ST)
    If t Then
       lDatum = DateSerial(ST.wYear, ST.wMonth, ST.wDay)
       lZeit = TimeSerial(ST.wHour, ST.wMinute, ST.wSecond)
       lDatum = lDatum + lZeit

       If lDatum > 0 Then
          DatPrüf = Format$(lDatum, "dd.mm.yy hh:mm:ss")
       Else
          DatPrüf = "kein gültiges Datum"
       End If
    End If
End Function

Sub DateiInfoDialog()
    Dim hFile As Long
    Dim WFD As WIN32_FIND_DATA
    Dim strDatei As String
    Dim Erstellt As String
    Dim Gespeichert As String
    Dim Zugriff As String

    strDatei = Application.CurrentDb.Name
    hFile = FindFirstFile(strDatei, WFD)
    If hFile > 0 Then
```

Listing 116: Dateiinformationen auslesen (Forts.)

```
      Erstellt = DatPrüf(WFD.ftCreationTime)
      Gespeichert = DatPrüf(WFD.ftLastWriteTime)
      Zugriff = DatPrüf(WFD.ftLastAccessTime)
      MsgBox "Datei angelegt: " & Erstellt & vbLf _
         & "Letzte Änderung: " & Gespeichert & vbLf _
         & "Letzter Zugriff: " & Zugriff, vbInformation, sDatei
   Else
      MsgBox "Datei wurde nicht gefunden!", vbCritical, sDatei
   End If
End Sub
```

Listing 116: Dateiinformationen auslesen (Forts.)

Mithilfe der Anweisung `Application.CurrentDb.Name` ermitteln Sie den Namen der aktuell geöffneten Datenbank. Diesen Namen übergeben Sie an die API-Funktion `FindFirstFile`. Danach übergeben Sie die Dateiinformationen an die Funktion `DatPrüf`. Dort werden die gewünschten Informationen ermittelt und in das richtige Format gebracht und an das aufrufende Makro zurückgegeben. Dort werden alle Informationen zusammengefasst und in einer Bildschirmmeldung ausgegeben.

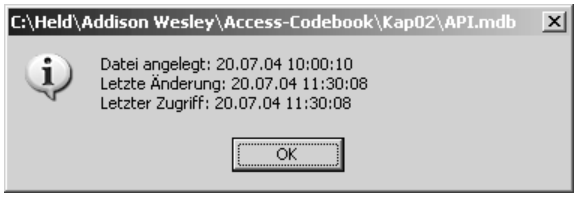


Abbildung 62: Auf Dateieigenschaften zugreifen

101 Internetverbindung aktiv?

Wenn Sie mit einem VBA-Makro ins Internet gehen möchten, dann sollten Sie vorher prüfen, ob überhaupt eine Verbindung mit dem Internet besteht. Dazu setzen Sie die API-Funktion `InternetGetConnectedState` aus Listing 117 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlInternet
'=====================================================

Public Declare Function InternetGetConnectedState Lib "wininet.dll" _
   (ByRef lpdwFlags As Long, ByVal dwReserved As Long) As Long

Sub InternetAktiv()
   Dim b As Boolean
   b = InternetGetConnectedState(0&, 0&)
```

Listing 117: Internetverbindung überprüfen

```
    If b = True Then
       MsgBox "Internetverbindung steht!"
    Else
       MsgBox "Internetverbindung fehlt!!"
    End If
End Sub
```

Listing 117: Internetverbindung überprüfen (Forts.)

Über die API-Funktion `InternetGetConnectedState` lässt sich ermitteln, ob eine Verbindung mit dem Internet besteht. Wenn ja, dann meldet diese Funktion den Wert `True` zurück, den Sie auswerten können.

102 Cursorposition in Pixel angeben

In Excel oder Access haben Sie in einer Tabelle über die Koordinaten der Zeilen und Spalten einen relativ schnellen Überblick, wo genau der Mauszeiger steht. Diese Angabe in der A1-Schreibweise sagt aber nicht aus, wo der Mauszeiger genau steht. Die genaue Position des Mauszeigers können Sie nur über die API-Funktion `GetCursorPos` aus Listing 118 bestimmen. Je nach Breite der Spalten bzw. Höhe der Zeilen ändern sich dann die Koordinaten.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlGetCursor
'=================================================

Declare Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long

Type POINTAPI
   x As Long
   y As Long
End Type

Sub KoordinatenErmitteln()
   Dim Point As POINTAPI
   Dim i As Integer

   i = GetCursorPos(Point)
   If i <> 0 Then
      MsgBox "X-Position: " & Point.x & vbLf & _
         "Y-Position: " & Point.y, vbInformation
   Else
      MsgBox "Es konnte keine Position ermittelt werden"
   End If
End Sub
```

Listing 118: Punktgenaue Koordinaten ermitteln

Definieren Sie zuerst die Struktur `POINTAPI`, in der Sie die Position des Mauszeigers in Form der Pixelkoordinaten speichern. Wenden Sie im Anschluss daran die API-Funktion `GetCursorPos` an, die Ihnen die aktuelle Position des Mauszeigers wiedergibt. Geben Sie die Koordinaten danach in einer Meldung auf dem Bildschirm aus.

Abbildung 63: Die genaue Cursorposition ermitteln

103 Stückzahl pro Stunde errechnen

Stellen Sie sich vor, Sie haben eine Gesamtzeit für die Produktion sowie eine in dieser Zeit erstellte Stückzahl von Produkten. Die Aufgabe besteht nun darin, die Stundenleistung zu errechnen. Um diese Aufgabe zu lösen, schreiben Sie eine Funktion, der Sie als Argumente die Gesamtzeit sowie die Stückzahl übergeben. Als Ergebnis liefert die Funktion aus Listing 119 die Anzahl in Stück/Stunde.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function StckPerStd(Zeit As Date, Stck) As Integer
   StckPerStd = Stck / (Zeit * 24)
End Function

Sub LeistungErrechnen()
   Debug.Print StckPerStd("4:35", 500)
   Debug.Print StckPerStd("3:25", 450)
   Debug.Print StckPerStd("5:45", 650)
   Debug.Print StckPerStd("7:25", 750)
End Sub
```

Listing 119: Stückzahl pro Stunde errechnen

Abbildung 64: Die einzelnen Stundenleistungen

104 Stunden in Minuten umrechnen

Bei der folgenden Aufgabe aus Listing 120 soll eine Zeitangabe, die in Stunden und Minuten vorliegt, in Minuten umgerechnet werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function ZeitInMinuten(Zeit As Date) As Integer
   ZeitInMinuten = Zeit * 24 * 60
End Function


Sub Bearbeitungszeit()
   MsgBox ZeitInMinuten("1:25")
End Sub
```

Listing 120: Stunden in Minuten umrechnen

Um eine Zeitangabe in Minuten umzurechnen, multiplizieren Sie die Zeitangabe zuerst mit 24 (24 Stunden am Tag) und anschließend mit 60 (60 Minuten pro Stunde). Selbstverständlich könnten Sie auch sofort mit dem Wert 1440 multiplizieren.

105 Minuten in Stunden umrechnen

Auch der umgekehrte Weg ist möglich. Es liegt eine Zeitangabe in Minuten vor, die in Stunden umgewandelt werden soll. Die Lösung dieser Aufgabe können Sie in Listing 121 sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================
```

Listing 121: Minuten in Stunden umrechnen

```
Function MinutenInStd(intMin As Integer) As Date
   MinutenInStd = intMin / 24 / 60
End Function

Sub Bearbeitungszeit2()
   MsgBox MinutenInStd(85)
End Sub
```

Listing 121: Minuten in Stunden umrechnen (Forts.)

Analog zur letzten Aufgabe aus Listing 120 dividieren Sie bei der Umrechnung von Minuten in Stunden mit den Werten 24 und 60 bzw. gleich mit 1440.

106 Normalzeit in Industriezeit umrechnen

Bei einer Industrieminute wird beispielsweise die Uhrzeit 3:30 als 3,5 dargestellt, also auf 100-er Basis umgerechnet. Dazu kann eine Funktion erstellt werden, die Sie in Listing 122 sehen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function IndZeit(Zeit As Date) As Single
   IndZeit = Zeit * 24
End Function

Sub NormalZeitInInustriezeitWandeln()
   MsgBox IndZeit("3:30")
End Sub
```

Listing 122: Normalzeit in Industriezeit umrechnen

107 Industriezeit in Normalzeit umwandeln

Der umgekehrte Vorgang ist ebenso von Interesse und wird über die Funktion aus Listing 123 realisiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================
```

Listing 123: Industriezeit in Normalzeit umrechnen

```
Function NormZeit(intZeit As Single) As Date
   NormZeit = intZeit / 24
End Function

Sub InustriezeiInNormalzeitWandeln()
   MsgBox NormZeit(3.5)
End Sub
```

Listing 123: Industriezeit in Normalzeit umrechnen (Forts.)

108 Schaltjahr oder nicht?

Im folgenden Beispiel aus Listing 124 wird über eine Funktion geprüft, ob es sich bei einem bestimmten Jahr um ein Schaltjahr handelt oder nicht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function Schaltjahr(lngJahr As Long) As Boolean
   Schaltjahr = Month(DateSerial(lngJahr, 2, 29)) = 2
End Function

Sub JahrTesten()
   If Schaltjahr(2004) Then
      MsgBox "Es handelt sich um ein Schaltjahr!"
   Else
      MsgBox "Normales Jahr"
   End If
End Sub
```

Listing 124: Der Schaltjahrtest über eine Funktion

Die Funktion `Schaltjahr` erwartet eine Jahresangabe. Die Funktion extrahiert daraus mithilfe der Funktion `Month` den Monat Februar und prüft, ob dieser Monat 29 Tage hat. Wenn ja, dann handelt es sich um ein Schaltjahr.

109 Stundengeschwindigkeit errechnen

Im folgenden Beispiel wird ein Marathon von einem Läufer in 2 Stunden und 13 Minuten zurückgelegt. Um jetzt eine Geschwindigkeitsangabe in km/h zu errechnen, setzen Sie die Funktion aus Listing 125 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function StdKm(sngkm As Single, Zeit As Date) As Single
   StdKm = sngkm / (Zeit * 24)
End Function

Sub Auswertung()
   Dim sngGesamtKm As Single
   Dim ZeitGesamt As Date

   sngGesamtKm = 42
   ZeitGesamt = "2:13"

   MsgBox "Die Geschwindigkeitsangabe km/h beträgt: " & _
      Round(StdKm(sngGesamtKm, ZeitGesamt),2)
End Sub
```

Listing 125: Durchschnittsgeschwindigkeit km/h errechnen

Abbildung 65: Eine relativ flotte Geschwindigkeit für einen Marathon

110 Durchschnittlichen Benzinverbrauch ermitteln

Um den durchschnittlichen Benzinverbrauch auf 100 km zu ermitteln, benötigen Sie die Menge des getankten Benzins sowie die damit gefahrenen Kilometer. Daraus lässt sich dann die Funktion aus Listing 126 schreiben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function Benzin(ltr As Single, km As Single) As Single
   Benzin = Round((ltr / km) * 100, 2)
End Function
```

Listing 126: Den Benzinverbrauch errechnen

```
Sub Benzinverbrauch()
   MsgBox "Der durchschnittliche Verbrauch liegt bei " & _
      Benzin(35, 488) & " Liter/Km", vbInformation
End Sub
```

Listing 126: Den Benzinverbrauch errechnen (Forts.)

Abbildung 66: Benzinverbrauch auf 100 km ausrechnen

111 Das Idealgewicht bestimmen

Wenn Sie wissen möchten, welches Ihr Ideal- bzw. Normalgewicht ist, dann können Sie beide Informationen in einer Funktion errechnen.

Das Normalgewicht errechnet sich, indem Sie von Ihrer Körpergröße in cm den Wert 100 subtrahieren und das Ergebnis daraus mit dem Multiplikator 0,9 multiplizieren. Das Idealgewicht entspricht der Körpergröße in cm, von der 100 Zentimeter subtrahiert werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function Gewicht(intG As Integer, intZ As Integer) As Single

Select Case intZ
   Case 1
      ' Normalgewicht
      Gewicht = Round((intG - 100) * 0.9, 2)
   Case 2
     ' Idealgewicht
      Gewicht = Round(intG - 100, 2)
   Case Else
      'Weitere Berechnungen
   End Select
End Function

Sub VorgabeGewichtErrechnen()
   Debug.Print "Normalgewicht: " & Gewicht(183, 1)
   Debug.Print "Idealgewicht: " & Gewicht(183, 2)
End Sub
```

Listing 127: Normal- und Idealgewicht errechnen

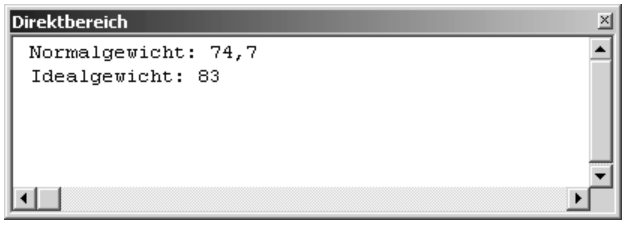


Abbildung 67: Normal- und Idealgewicht bei einer Größe von 1.83

112 Fahrleistung pro Tag errechnen

Möchten Sie Ihre durchschnittlich gefahrenen Kilometer pro Tag ermitteln, dann benötigen Sie folgende Informationen:

- Datum des Autokaufs
- Aktuelles Datum
- Bisher gefahrene Gesamtkilometer

Aus diesen Informationen schreiben Sie die Funktion aus Listing 128.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function KmProTag(sngKMGesamt As Single, _
   DateAkt As Date, DateKauf As Date) As Single
   KmProTag = Round(sngKMGesamt / (DateAkt - DateKauf), 2)
End Function

Sub FahrleistungProTag()
   MsgBox "Durchschnittlich wurden " & _
   KmProTag(34567, "20.07.2004", "25.10.2001") & " Km pro Tag gefahren!"
End Sub
```

Listing 128: Die durchschnittliche Kilometerleistung pro Tag

Abbildung 68: Durchschnittsleistung pro Tag

113 Berechnung des Bremswegs

Die Berechnung des Bremswegs bei einem Auto erfolgt mittels drei Einzelformeln:

- Reaktionsweg = Geschwindigkeit (in km/h) / 10)*3
- Bremsweg = Geschwindigkeit (in km/h) / 10) ^2
- Anhalteweg = Reaktionsweg + Bremsweg

Für den Reaktionsweg wird eine Reaktionsdauer von 1 Sekunde unterstellt. Diese Prämisse gilt aber nur für Autofahrer in guter körperlicher Verfassung. Müdigkeit, Alkohol, Ablenkung etwa durch laute Musik verlängern die Reaktionszeit.

Beim Bremsweg ist die Strecke zwischen dem Ansprechen der Bremsen bis zum absoluten Stillstand des Autos gemeint. Ganz grob kann man sagen, dass sich bei einer Verdopplung der Geschwindigkeit der Bremsweg vervierfacht.

Der Anhalteweg ergibt sich aus der Addition von Reaktionsweg und Bremsweg.

Diese Informationen packen Sie in die Funktion aus Listing 129.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=================================================

Function AnhaltWeg(intGeschw As Integer) As Single
   AnhaltWeg = (intGeschw / 10) * 3 + (intGeschw / 10) ^ 2
End Function

Sub WannStehtAuto()
   Dim intkm As Integer

   For intkm = 100 To 250 Step 10
      Debug.Print "Geschwindigkeit: " & intkm & vbTab & _
         "steht nach " & AnhaltWeg(intkm) & " Metern"
   Next intkm
End Sub
```

Listing 129: Anhalteweg errechnen

114 Verzeichnis überprüfen

Viele Fehler in der Programmierung können abgefangen werden, wenn Sie vor dem Zugriff auf ein Objekt sicherheitshalber dessen Existenz überprüfen. Dabei können Sie Prüffunktionen verwenden, die aus Makros aufgerufen werden und als Rückgabewert den Wert `True` liefern, sofern ein Objekt verfügbar ist, bzw. den Wert `False`, wenn ein Objekt nicht zur Verfügung steht. Im zweiten Fall können Sie dann entscheiden, wie Sie im Einzelnen weiter vorgehen möchten.

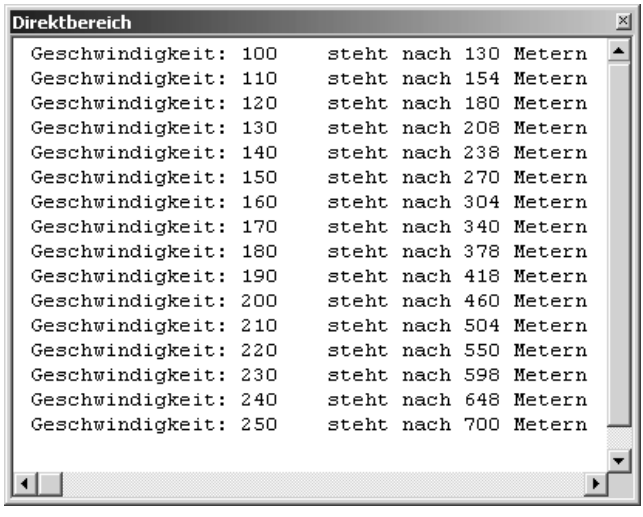


Abbildung 69: Die Dokumentation des Anhaltewegs

Im Beispiel aus Listing 130 möchten Sie in ein bestimmten Verzeichnis wechseln. Zu diesem Zweck müssen Sie sicherstellen, dass das Zielverzeichnis auch existiert. Wenn nicht, dann soll das Makro das nicht vorhandene Verzeichnis selbstständig anlegen. Danach soll der Verzeichniswechsel wie gewohnt stattfinden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function VerzDa(strName As String) As Boolean
   On Error GoTo fehler
   ChDir (strName)
   VerzDa = True
   Exit Function

   fehler:
      VerzDa = False
End Function

Sub PfadEinstellen()
   Dim b As Boolean
   Const Ordner = "C:\Eigene Dateien\"
   b = VerzDa(Ordner)

   If b = False Then
      ' Neuanlage des Verzeichnisses
      MkDir Ordner
```

Listing 130: Prüfung, ob ein Verzeichnis existiert

```
    Else
       ChDir Ordner$
    End If
 End Sub
```

Listing 130: Prüfung, ob ein Verzeichnis existiert (Forts.)

Die Funktion `VerzDa` erwartet als Empfangsargument einen String. In diesem String `StrName` übergeben Sie das Verzeichnis, dessen Existenz Sie überprüfen möchten. Über die Anweisung `ChDir` versuchen Sie, innerhalb der Funktion einfach mal dieses Verzeichnis einzustellen. Schlägt dieser Vorgang fehl, dann wird ein Fehler erzeugt, den Sie über die `On Error`-Klausel abfangen können. In diesem Fall verzweigen Sie ans Ende der Funktion und geben als Rückgabeargument den Wert `False` zurück. Ergab die Prüfung des Verzeichnisses ein positives Ergebnis geben Sie als Rückgabeargument den Wert `True` zurück und springen danach über die Anweisung `Exit Function` direkt aus der Funktion.

Deklarieren Sie beim aufrufenden Makro im ersten Schritt eine Variable vom Typ `Boolean`. Diese Variable meldet Ihnen das Ergebnis der Funktion `VerzDa` in Form eines Wahrheitswerts mit `True` (Verzeichnis existiert) bzw. `False` (Verzeichnis existiert nicht). Den Ordner, dessen Existenz Sie überprüfen möchten, können Sie entweder direkt beim Funktionsaufruf angeben oder in einer Konstanten, gleich zu Beginn des Makros. Rufen Sie die Funktion `VerzDa` nun auf und warten Sie auf den Rückgabewert der Funktion, welchen Sie über eine `If`-Abfrage auswerten. Für den Fall, dass das Verzeichnis noch nicht verfügbar ist, legen Sie das gewünschte Verzeichnis über die Anweisung `MkDir` an. Wechseln Sie danach über die Anweisung `ChDir` in das Zielverzeichnis.

115 Datenbank überprüfen

Die Existenz einer Datenbank sollte auf jeden Fall gewährleistet sein, wenn Sie versuchen, eine Datenbank zu öffnen bzw. eine Datenbank zu löschen. Beim Versuch, eine Datenbank zu öffnen, die nicht vorhanden ist, wird der Laufzeitfehler 1004 ausgelöst. Beim Versuch, eine nicht existente Datenbank von der Festplatte zu entfernen, wird der Laufzeitfehler 53 (Datei nicht gefunden) angezeigt. Beide Fehlermeldungen können Sie unterdrücken, indem Sie die folgende Lösung aus Listing 131 anwenden:

```
 '=====================================================
 ' Auf CD     Buchdaten\Beispiele\Kap02
 ' Dateiname  WeitereFunktionen.mdb
 ' Modul      MdlWeitereFunktionen
 '=====================================================

 Function DBDa(strName As String) As Boolean
    DBDa = False
    If Len(strName) > 0 Then DBDa = (Dir(strName) <> "")
 End Function
```

Listing 131: Prüfung vor der Löschung einer Datenbank

```
Sub DBLöschen()
   Dim b As Boolean
   Const Datei = "c:\Eigene Dateien\DB1.mdb"

   b = DBDa(Datei)
   If b = True Then
      Kill Datei
   Else
      MsgBox "Die angegebene Datei konnte nicht gefunden werden!", vbCritical
   End If
End Sub
```

Listing 131: Prüfung vor der Löschung einer Datenbank (Forts.)

Die Funktion `DBDa` erwartet einen String, in dem Sie den Namen sowie den kompletten Pfad der Datei übergeben, die Sie öffnen bzw. löschen möchten. Über die Funktion `Len` können Sie überprüfen, ob überhaupt ein gültiger Dateiname an die Funktion übergeben wurde. Wenn ja, dann liefert diese Funktion einen Wert größer Null. In diesem Fall wenden Sie die Funktion `Dir` an, um zu testen, ob die gewünschte Datei überhaupt vorhanden ist. Wird die so angesteuerte Datei gefunden, dann meldet die Funktion `Dir` einen Wert ungleich leer zurück. In diesem Fall gibt die Funktion `DBDa` als Rückgabeargument den Wert `True` zurück. Im anderen Fall wird als Rückgabeargument der Wert `False` übergeben, den Sie ja standardmäßig zu Beginn der Funktion gesetzt haben.

116 Datenbankstatus überprüfen

Wenn Sie in einem Netzwerk arbeiten und versuchen, eine Datenbank zu öffnen, die ein Kollege bereits geöffnet hat, dann sollten Sie vor dem Öffnen der Datenbank prüfen, ob Sie diese im Exklusivzugriff haben. Die Funktion für diesen Zweck lautet:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function DBInBearbeitung(strDatei As String) As Boolean
   On Error Resume Next
   Open strDatei For Binary Access Read Lock Read As #1
   Close #1
   If Err.Number <> 0 Then
      DBInBearbeitung = True
      Err.Clear
   End If
End Function
```

Listing 132: Datenbankstatus prüfen

```
Sub DateiFrei()
   Const DB = "DB1.mdb"

   If DBInBearbeitung("C:\Eigene Dateien\" & DB) = False Then
      MsgBox "Die Datenbank " & DB & " ist für Bearbeitung frei!"
   Else
      MsgBox "Die Datenbank " & DB & " ist in Bearbeitung!"
   End if
End Sub
```

Listing 132: Datenbankstatus prüfen (Forts.)

Mit der Methode `Open` öffnen Sie die Datenbank mit Lesezugriffsrechten. Ist diese Datenbank bereits geöffnet, liefert Ihnen die Eigenschaft `Number` des `Err`-Objekts einen Laufzeitfehler > 0. In diesem Fall wird die Datenbank momentan von einem anderen Anwender bearbeitet.

117 Objektstatus erkennen

Sehr nützlich ist auch die Funktion aus Listing 133. Diese Funktion prüft den Status eines Objekts. Dabei können Sie bei dieser Lösung erkennen, ob eine Tabelle, eine Abfrage, ein Formular, ein Bericht, ein Makro oder ein Modul geöffnet ist oder nicht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function ObjektStatus(ObjTyp As Integer, StrName As String) As Boolean
   ObjektStatus = SysCmd(acSysCmdGetObjectState, ObjTyp, StrName)
End Function

Sub ObjektPrüfen()
   Dim b As Boolean

   b = ObjektStatus(0, "Kategorien")
   If b = True Then
      MsgBox "Objekt geöffnet!"
   Else
      MsgBox "Objekt geschlossen!"
   End If
End Sub
```

Listing 133: Prüfung, ob ein Objekt geöffnet ist

Übergeben Sie der Methode über einen numerischen Wert, die den Objekttyp darstellt, das zu prüfende Objekt. Dabei gelten folgende Zuordnungen:

- Tabelle: 0
- Abfrage:1
- Formular:2
- Bericht:3
- Makro:4
- Modul:5

Im zweiten Argument übergeben Sie den Namen des Objekts. In Abhängigkeit vom Status wird dementsprechend eine Meldung am Bildschirm angezeigt.

118 Dateien zählen

Stellen Sie sich vor, Sie müssten in einem Makro feststellen, wie viele Dateien sich in einem Verzeichnis befinden. Dazu erfassen Sie zunächst folgende Funktion aus Listing 134:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function DZ(strText As String) As Integer
   Dim strDatei As String
   Dim intz As Integer

   strDatei = Dir$(strText & "\*.*")
   Do While Len(strDatei) > 0
      intz = intz + 1
      strDatei = Dir$()
   Loop
   DZ = intz
End Function

Sub ZählenDateien()
   Const Verz = "C:\Eigene Dateien\"
   Dim intz As Integer

   intz = DZ(Verz)
   MsgBox "Das Verzeichnis " & Verz & " enthält " & intz & " Dateien!"
End Sub
```

Listing 134: Dateien aus einem Verzeichnis zählen

Die Funktion `DZ` erwartet als Eingabe den Namen des Verzeichnisses, auf das Sie zugreifen möchten. Als Ergebnis liefert die Funktion Ihnen im Datentyp `Long` die Anzahl der ermittelten Dateien. Wenn Sie nur bestimmte Dateien gezählt haben möchten, können Sie die obige Funktion abändern, indem Sie die Zeichenfolge `StrDatei = Dir$(strText & "\*.*")` bei-

spielsweise in `StrDatei = Dir$(strText & "\*.mdb")` ändern. Diese kleine Änderung bewirkt, dass nur Access-Datenbanken gezählt werden.

Im aufrufenden Makro legen Sie gleich zu Beginn fest, welches Verzeichnis Sie durchsuchen möchten. Übergeben Sie anschließend der Funktion `DZ` genau dieses Verzeichnis.

Abbildung 70: Dateien zählen

119 Initialen erstellen

Sollen aus einem Vor-, Zu- und Nachnamen Initialen gebildet werden, so dass aus dem Namen Bernd Michael Held die Initiale BMH wird, dann können Sie die Funktion aus Listing 135 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function Initit(strName As String) As String
   Dim intZ As Integer
   strName = " " & RTrim(strName)

   For intZ = 2 To Len(strName)
      If Mid(strName, intZ - 1, 1) = " " Then
         Initit = Initit & Mid(strName, intZ, 1)
      End if
   Next intZ
End Function

Sub InitialenErstellen()
   Debug.Print "Bernd Michael Held --> " & Initit("Bernd Michael Held")
   Debug.Print "Anna Maria Schmitt --> " & Initit("Anna Maria Schmitt ")
   Debug.Print "Jürgen Bär --> " & Initit("Jürgen Bär")
End Sub
```

Listing 135: Initialen erstellen

Entfernen Sie im ersten Schritt mithilfe der Funktion `RTrim` nachgestellte, eventuell vorkommende Leerzeichen. Danach beginnen Sie die Verarbeitung beim zweiten Zeichen des ersten Namens. Dabei setzen Sie eine Schleife auf, die Buchstaben für Buchstaben abarbeitet. Innerhalb der Schleife prüfen Sie, wann der jeweilige Namen zu Ende geht, da Sie von jedem Ein-

zelnamen nur das erste Zeichen übertragen müssen. Diese jeweils ersten Zeichen geben Sie am Ende der Funktion an das aufrufende Makro zurück.

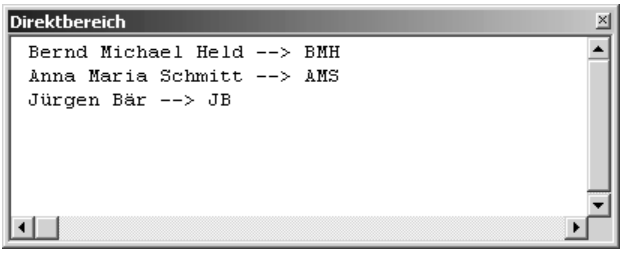


Abbildung 71: Auf schnelle Art und Weise Initialen bilden

120 Position der ersten Zahl erkennen

Haben Sie einen Text vorliegen, der sowohl numerische als auch alphanumerische Werte enthält, dann könnte Sie die folgende Funktion interessieren. Sie meldet Ihnen die Position des ersten numerischen Zeichens im Text. Diese Funktion können Sie in Listing 136 sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function PosErsteZahl(strText As String) As Integer
   Dim intZ As Integer

   For intZ = 1 To Len(strText)
      Select Case Asc(Mid(strText, intZ, 1))
         Case 48 To 57
            PosErsteZahl = intZ
            Exit Function
      End Select
   Next intZ
   PosErsteZahl = 0
End Function

Sub ZahlUndText()
   Debug.Print "Jahr2004 --> " & PosErsteZahl("Jahr2004") & ". Stelle"
   Debug.Print "0815     --> " & PosErsteZahl("0815") & ". Stelle"
   Debug.Print "Test     --> " & PosErsteZahl("Test") & ". Stelle"
   Debug.Print "H2O      --> " & PosErsteZahl("H2O") & ". Stelle"
End Sub
```

Listing 136: Position der ersten Ziffer melden

Ermitteln Sie im ersten Schritt die Länge des Textes, setzen Sie dafür die Funktion `Len` ein. Danach prüfen Sie mithilfe der Funktion `Asc` das jeweils aktuelle Zeichen des Textes, indem Sie dieses in einen Integer-Wert umwandeln. Mit der Funktion `Mid` extrahieren Sie jeweils das nächste Zeichen aus dem Text. Dabei entsprechen die Werte 48 bis 57 den Zahlen 0 bis 9. Diese Wertebereiche grenzen Sie innerhalb der `Select Case`-Anweisung aus. Wird das erste numerische Zeichen im Text gefunden, springen Sie mit der Anweisung `Exit Function` aus der Funktion. In der Variablen steht dann automatisch die richtige Position des Zeichens. Wurde kein numerisches Zeichen gefunden, dann meldet die Funktion den Wert `0` zurück.

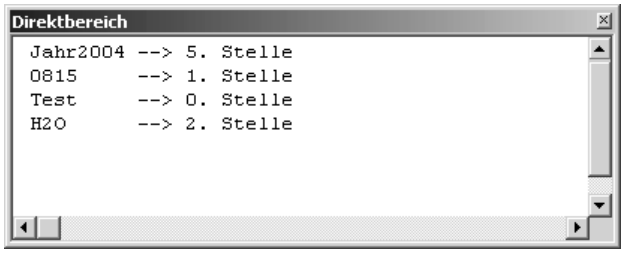


Abbildung 72: Das jeweils erste numerische Zeichen wird gemeldet.

121 Position des ersten Buchstabens erkennen

Analog dazu können Sie die Position des ersten Buchstabens, im Bereich A–Z und a–z, ermitteln, indem Sie die Funktion aus Listing 137 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function PosErsterBuch(strText As String) As Integer
   Dim intZ As Integer

   For intZ = 1 To Len(strText)
      Select Case Asc(Mid(strText, intZ, 1))
         Case 0 To 64, 123 To 197
            'keine Aktion
         Case Else
            PosErsterBuch = intZ
            Exit Function
      End Select
   Next intZ
   PosErsterBuch = 0
End Function
```

Listing 137: Position des ersten Buchstabens ermitteln

```
Sub WoStecktDerBuchstabe()
   MsgBox "Der erste Buchstabe im Text 00045T steckt an Position " & _
   PosErsterBuch("00045T")
End Sub
```

Listing 137: Position des ersten Buchstabens ermitteln (Forts.)

Ermitteln Sie im ersten Schritt die Länge des Textes und setzen Sie dafür die Funktion `Len` ein. Danach prüfen Sie mithilfe der Funktion `Asc` das jeweils aktuelle Zeichen des Textes, indem Sie dieses in einen Integer-Wert umwandeln. Mit der Funktion `Mid` extrahieren Sie jeweils das nächste Zeichen aus dem Text. Dabei entsprechen die Werte 65 bis 90 Kleinbuchstaben und die Werte 97 bis 122 den Großbuchstaben. Diese Wertebereiche grenzen Sie innerhalb der `Select Case`-Anweisung aus. Sobald das erste Zeichen in der Zelle gefunden wird, welches numerisch ist, springen Sie mit der Anweisung `Exit Function` aus der Funktion. In der Variablen steht dann automatisch die richtige Position des Zeichens. Wurde kein numerisches Zeichen gefunden, dann meldet die Funktion den Wert 0 zurück.

Abbildung 73: Das große O sieht der 0 ziemlich ähnlich.

122 Sonderzeichen entfernen

Müssen Sie Daten weiterverarbeiten, in denen Sonderzeichen wie Punkte und Kommas vorkommen, die Sie nicht weiterverarbeiten möchten, dann schreiben Sie eine Funktion, die diese Zeichen aus einem String entfernt.

Sehen Sie sich dazu die folgende Funktion aus Listing 138 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function SonderzRaus(strText As String) As Double
   Dim strPuffer As String
   Dim intz As Integer

   strPuffer = ""
   intz = 1
```

Listing 138: Punkte und Kommas aus Zeichenfolge entfernen

```
    While InStr(intz, strText, ",") > 0
       strPuffer = strPuffer & _
       Mid(strText, intz, InStr(intz, strText, ",") - intz)
       intz = InStr(intz, strText, ",") + 1
    Wend

    strPuffer = strPuffer & Mid(strText, intz)
    SonderzRaus = CDbl(strPuffer)
 End Function

 Sub PunkteUndKommasRaus()
    Dim strTextVorher As String
    Dim strTextNachher As String

    strTextVorher = "169.123.278,45"
    Debug.Print "Zeichenkette vorher: " & strTextVorher>
    strTextNachher = SonderzRaus(strTextVorher)
    Debug.Print "Zeichenkette nachher: " & strTextNachher
 End Sub
```

Listing 138: Punkte und Kommas aus Zeichenfolge entfernen (Forts.)

In der Funktion durchlaufen Sie eine Schleife, in der die Zeichen jeweils bis zum nächsten Komma in den String `strPuffer` übertragen werden. Dabei wird das Komma aber nicht übertragen, da Sie es jeweils wieder über die Variable `intz` subtrahieren. Ermitteln Sie danach die Position des nächsten Kommas über die Funktion `Instr`.

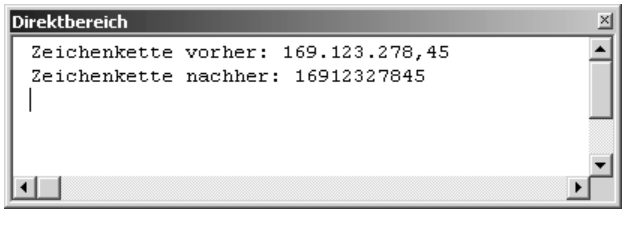


Abbildung 74: Bereinigung von Zeichenketten

123 Numerische Ziffern in Zeichenfolge zählen

Im nächsten Beispiel aus Listing 139 wird die Anzahl der Zahlen, die in einem Text enthalten sind, ermittelt.

```
 '=====================================================
 ' Auf CD     Buchdaten\Beispiele\Kap02
 ' Dateiname  WeitereFunktionen.mdb
 ' Modul      MdlWeitereFunktionen
 '=====================================================
```

Listing 139: Die Anzahl der Zahlen zählen

```
Function AnzahlZahlen(strText As String) As Integer
   Dim intZ As Integer

   For intZ = 1 To Len(strText)
      If Mid(strText, intZ, 1) Like "#" Then
         AnzahlZahlen = AnzahlZahlen + 1
      End If
   Next intZ
End Function

Sub ZahlenZählen()
   Debug.Print "WGO897   --> " & AnzahlZahlen("WGO897")
   Debug.Print "XY O9O   --> " & AnzahlZahlen("XY O9O")
   Debug.Print "1O1O1OTZ --> " & AnzahlZahlen("1O1O1OTZ")
End Sub
```

Listing 139: Die Anzahl der Zahlen zählen (Forts.)

Ermitteln Sie zuerst einmal die Gesamtlänge des Textes über die Funktion `Len`. Danach durchlaufen Sie eine Schleife und arbeiten mithilfe der Funktion `Mid` ein Zeichen nach dem anderen ab. Setzen Sie den Operator `Like` ein, um zu ermitteln, ob es sich bei dem Zeichen um einen Zahlenwert handelt. In diesem Fall meldet Ihnen dieser Operator einen Wert zwischen `0` und `9` zurück. Addieren Sie dann den Wert `1` in Ihrer Funktion.

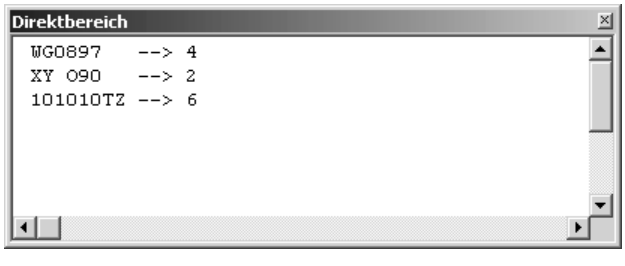


Abbildung 75: Auch hier ist wieder auf den Unterschied zwischen 0 und O zu achten!

124 Buchstaben in Zeichenfolge zählen

Analog zur vorherigen Aufgabe zählt die Funktion aus Listing 140 alle Buchstaben, die in einem Text auftreten.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================
```

Listing 140: Alle Buchstaben in einem Text zählen

```
Function AnzahlBuch(strText As String) As Integer
   Dim intZ As Integer

   For intZ = 1 To Len(strText)
      Select Case Asc(Mid(strText, intZ, 1))
         Case 0 To 64, 123 To 197
            'keine Aktion
         Case Else
            AnzahlBuch = AnzahlBuch + 1
      End Select
   Next intZ
End Function

Sub BuchstabenZählen()
   Debug.Print "WG0897   --> " & AnzahlBuch("WG0897")
   Debug.Print "XY O9O   --> " & AnzahlBuch("XY O9O")
   Debug.Print "1O1O1OTZ --> " & AnzahlBuch("1O1O1OTZ")
End Sub
```

Listing 140: Alle Buchstaben in einem Text zählen (Forts.)

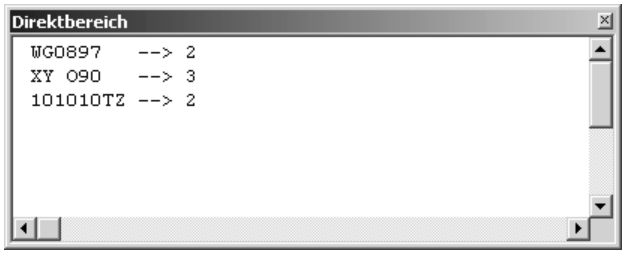


Abbildung 76: Die Anzahl der Buchstaben wurde ermittelt..

125 Römische Zahlen umwandeln

Liegen römische Ziffern vor, die in Zahlen umgewandelt werden sollen, dann können Sie die folgende Funktion aus Listing 141 verwenden.

Das römische Zahlensystem verwendete Buchstaben als Zahlen. So war der Buchstabe M gleichbedeutend mit dem Wert 1000. Von daher müssen Sie Zeichen für Zeichen im Text abarbeiten und die entsprechenden Werte sammeln, um sie am Ende als Rückgabewert auszugeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================
```

Listing 141: Umwandeln von römischen Zahlen

```
Function Arabisch(strText As String) As Integer
   Dim intz As Integer
   Dim intTeilW   As Integer
   Dim intTeilW2  As Integer
   Dim intGesamtW As Integer

   intGesamtW = 0
   intTeilW = 0
   intTeilW2 = 0

   For intz = 1 To Len(strText)
      Select Case Mid(strText, intz, 1)
         Case Is = "M"
            intTeilW = 1000
         Case Is = "D"
            intTeilW = 500
         Case Is = "C"
            intTeilW = 100
         Case Is = "L"
            intTeilW = 50
         Case Is = "X"
            intTeilW = 10
         Case Is = "V"
            intTeilW = 5
         Case Is = "I"
            intTeilW = 1
         Case Else
            intTeilW = 0
      End Select

      If intTeilW2 < intTeilW Then
         intGesamtW = intGesamtW - intTeilW2 * 2 + intTeilW
      Else
         intGesamtW = intGesamtW + intTeilW
      End If
      intTeilW2 = intTeilW
   Next intz
   Arabisch = intGesamtW
End Function


Sub ZahlRom()
   Debug.Print "MDLX  --> " & Arabisch("MDLX")
   Debug.Print "MCCL  --> " & Arabisch("MCCL")
   Debug.Print "DIX   --> " & Arabisch("DIX")
   Debug.Print "CCCLX --> " & Arabisch("CCCLX")
End Sub
```

Listing 141: Umwandeln von römischen Zahlen (Forts.)

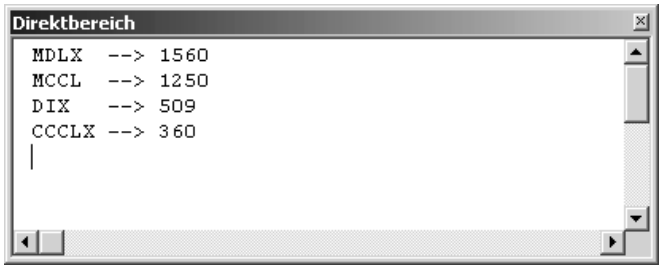


Abbildung 77: Das Ergebnis der Umwandlung

126 Arabische Zahlen umwandeln

Beim gerade umgekehrten Vorgang werden Sie an dieser Stelle eine etwas andere Vorgehensweise kennen lernen. Möchten Sie eine arabische Zahl in eine römische Zahl umwandeln, dann können Sie die Excel-Tabellenfunktion RÖMISCH einsetzen. Diese Funktion können Sie direkt aus Access aufrufen, indem Sie in der Entwicklungsumgebung die Bibliothek MICROSOFT EXCEL mithilfe des Menübefehls EXTRAS/VERWEISE aktivieren. Danach erfassen Sie folgenden Quellcode aus Listing 142:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function Römisch(strZahl As String) As String
   Dim xlApp As Excel.Application

   Set xlApp = New Excel.Application
   Römisch = xlApp.worksheetfunction.roman(strZahl)
   xlApp.Quit
   Set xlApp = Nothing
End Function


Sub ZahlArab()
   Debug.Print " 567 --> " & Römisch(567)
   Debug.Print "1579 --> " & Römisch(1579)
   Debug.Print " 936 --> " & Römisch(936)
End Sub
```

Listing 142: Arabische Zahlen in römische Zahlen umwandeln

Erstellen Sie zunächst ein neues Excel-Objekt. Danach greifen Sie über das Objekt `WorksheetFunction` auf die Tabellenfunktion `roman` zurück und übergeben dieser Funktion die arabische Zahl. Als Rückgabewert liefert die Funktion die dazugehörende römische Ziffer.

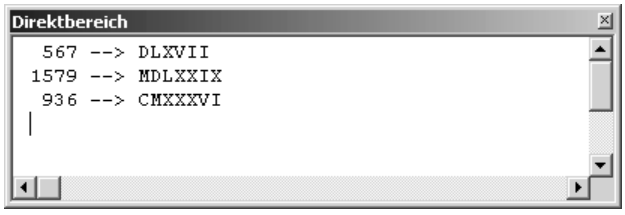


Abbildung 78: Die Zahlen wurden in die römische Syntax umgesetzt.

127 Auf Dokumenteigenschaften zugreifen

Im folgenden Beispiel aus Listing 143 wird auf die Dokumenteigenschaften einer Datenbank zugegriffen. Dabei orientiert sich die Funktion an Tabelle 55.

Eigenschaftsnummer	Beschreibung
0	Dateiname mit Pfad
1	nur Pfad
2	nur Dateiname
3	Dateityp
4	Dateigröße in Byte
5	erstellt am
6	letzte Änderung am
7	letzter Zugriff am

Tabelle 55: Die verfügbaren Dokumenteigenschaften

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function ZeigeDateiEigenschaften(Dateiname, EigenschaftsNr As Byte)
   Dim fso As Object
   Dim tmp As String

   On Error Resume Next
   Set fso = CreateObject("Scripting.FileSystemObject")

   With fso.GetFile(Dateiname)
      Select Case EigenschaftsNr
         Case Is = 0: tmp = .path
         Case Is = 1: tmp = Mid(.path, 1, Len(.path) - Len(.Name))
         Case Is = 2: tmp = .Name
```

Listing 143: Dokumenteigenschaften über eine Funktion abfragen

```
         Case Is = 3: tmp = .Type
         Case Is = 4: tmp = .Size
         Case Is = 5: tmp = CDate(.DateCreated)
         Case Is = 6: tmp = CDate(.DateLastModified)
         Case Is = 7: tmp = CDate(.DateLastAccessed)
         Case Else
            tmp = "Ungültige EigenschaftsNr!"
      End Select
   End With
   ZeigeDateiEigenschaften = tmp
End Function

Sub DokumentEigenschaften()
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 1)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 2)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 3)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 4)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 5)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 6)
   Debug.Print ZeigeDateiEigenschaften(CurrentDb.Name, 7)
End Sub
```

Listing 143: Dokumenteigenschaften über eine Funktion abfragen (Forts.)

Erstellen Sie im ersten Schritt einen Verweis auf das `FileSystemObject`, um damit die Informationen bezüglich der Datenbank zu erhalten. Danach werten Sie die übergebene Eigenschaftsnummer in einer `Select Case`-Anweisung aus. Bei den Datumswerten wenden Sie die Funktion `CDate` an, um diese Angaben in das richtige Format zu bringen.

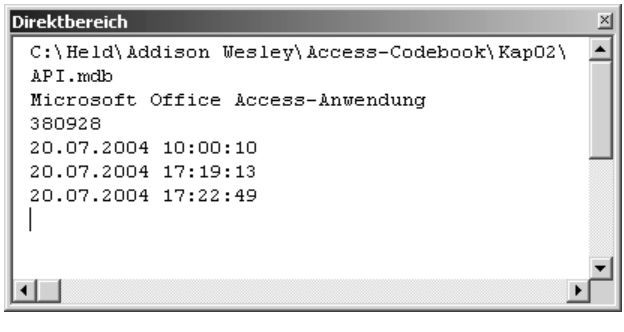


Abbildung 79: Die Dokumenteigenschaften werden über eine Nummer ermittelt.

128 Den letzten Tag im Monat ermitteln

Vielleicht haben Sie manchmal auch Probleme, den letzten Tag eines Monats schnell zu erkennen. Hat der Monat jetzt 30 oder 31 Tage? Es gibt hierfür zwar recht einfache Bauernregeln, wie etwa das Zählen der Mulden zwischen den Fingerknochen. Aber auch da herrscht doch relativ große Unsicherheit, je nachdem, ob Sie von links oder rechts anfangen zu zählen. Eine nahezu sichere Lösung bietet die Funktion aus Listing 144.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap02
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'====================================================

Function LTImMo(inputdate As Date) As Date
   LTImMo = DateSerial(Year(inputdate), Month(inputdate) + 1, 0)
End Function


Sub LetzterTagImMonatErmitteln()
   Debug.Print "01.01.2004 --> " & LTImMo("01.01.2004")
   Debug.Print "01.02.2004 --> " & LTImMo("01.02.2004")
   Debug.Print "01.03.2004 --> " & LTImMo("01.03.2004")
   Debug.Print "01.04.2004 --> " & LTImMo("01.04.2004")
   Debug.Print "01.05.2004 --> " & LTImMo("01.05.2004")
   Debug.Print "01.06.2004 --> " & LTImMo("01.06.2004")
   Debug.Print "01.07.2004 --> " & LTImMo("01.07.2004")
   Debug.Print "01.08.2004 --> " & LTImMo("01.08.2004")
   Debug.Print "01.09.2004 --> " & LTImMo("01.09.2004")
   Debug.Print "01.10.2004 --> " & LTImMo("01.10.2004")
   Debug.Print "01.11.2004 --> " & LTImMo("01.11.2004")
   Debug.Print "01.12.2004 --> " & LTImMo("01.12.2004")
End Sub
```

Listing 144: Den letzten Tag/Monat ermitteln

Mithilfe der Funktion `DateSerial` wird ein Datum in seine Bestandteile zerlegt. Über die Funktionen `Year` und `Month` extrahieren Sie dann das jeweilige Jahr sowie den Monat.

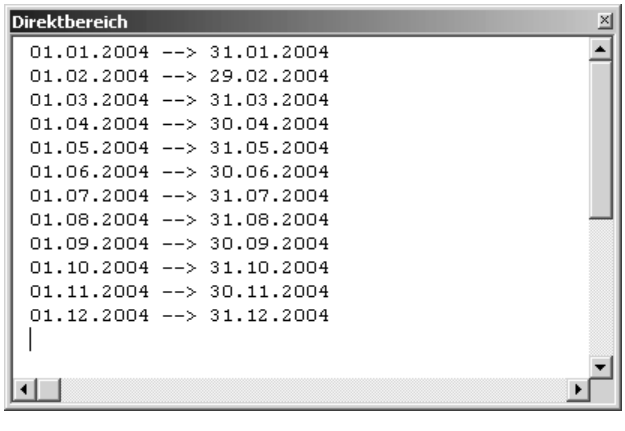


Abbildung 80: Eine komplette Liste mit den »Letzten Tagen« liegt vor.

Die Access-Objekte

In diesem Kapitel können Sie Beispiele zu den wichtigsten Access-Objekten nachschlagen. Zu jedem Objekt stehen Ihnen zahlreiche Methoden und Eigenschaften zur Verfügung, die Sie bei der Programmierung einsetzen können.

129 Das Objekt AccessObject

Mithilfe des `AccessObject`-Objekts können Sie auf Auflistungsobjekte zugreifen und diese auswerten. Dabei stehen Ihnen folgende Auflistungsobjekte zur Verfügung:

AccessObject	Auflistung	Enthält Informationen über
Datenzugriffsseite	`AllDataAccessPages`	gespeicherte Datenzugriffs-seiten
Datenbankdiagramm	`AllDatabaseDiagrams`	gespeicherte Datenbank-diagramme
Form	`AllForms`	gespeicherte Formulare
Funktion	`AllFunctions`	gespeicherte Funktionen
Makro	`AllMacros`	gespeicherte Makros
Modul	`AllModules`	gespeicherte Module
Abfrage	`AllQueries`	gespeicherte Abfragen
Bericht	`AllReports`	gespeicherte Berichte
Gespeicherte Prozedur	`AllStoredProcedures`	gespeicherte Prozeduren
Table	`AllTables`	gespeicherte Tabellen
Sicht	`AllViews`	gespeicherte Ansichten

Tabelle 56: Alle `AccessObjects` im Überblick

130 Alle Module auflisten

Um alle in der Entwicklungsumgebung angelegten Module aufzulisten, können Sie auf die Auflistung `AllModules` zurückgreifen. Im Beispiel aus Listing 145 werden die Namen sowie das Erstellungsdatum aller Module im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'=================================================

Sub ModuleAuflisten()
   Dim AccObj As AccessObject
   Dim dbs As CurrentProject

   Set dbs = Application.CurrentProject
   For Each AccObj In dbs.AllModules
      Debug.Print "Name        : " & AccObj.Name
      Debug.Print "Erstellt am : " & AccObj.DateCreated & vbLf
   Next AccObj
End Sub
```

Listing 145: Alle Module werden im Direktfenster dokumentiert.

Über die Eigenschaft `CurrentProject` haben Sie Zugriff auf die aktuelle Datenbank. Über die Eigenschaft `Name` können Sie sich die Namen der einzelnen Module ausgeben lassen, wenn Sie das Auflistungsobjekt `AllModules` einsetzen. Mithilfe der Eigenschaft `DateCreated` können Sie abfragen, wann das Modul erstellt wurde.

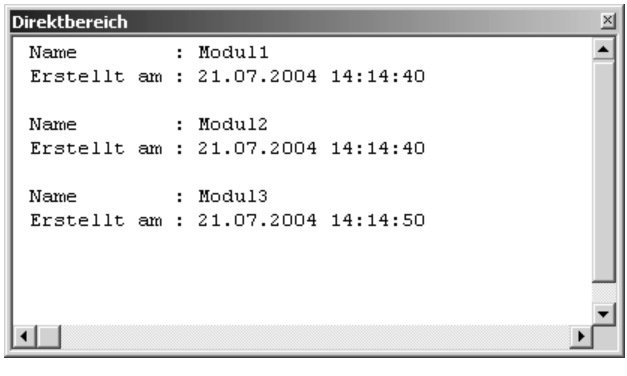


Abbildung 81: Die verfügbaren Module der Datenbank

Hinweis

In Kapitel 9 können Sie nachschlagen, wie Sie mithilfe der VBE-Programmierung noch weiter gehende Aufgaben schnell erledigen können.

131 Alle Tabellen auflisten

Über die Auflistung `AllTables` haben Sie Zugriff auf alle Tabellen, die sich in der Datenbank befinden. Im Makro aus Listing 146 werden die Namen aller Tabellen in das Direktfenster der

Entwicklungsumgebung geschrieben. Dabei wird geprüft, ob die jeweilige Tabelle momentan geöffnet ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'=====================================================

Sub TabellenAuflisten()
   Dim Accobj As AccessObject
   Dim dbs As CurrentData

   Set dbs = Application.CurrentData
   For Each Accobj In dbs.AllTables
      Debug.Print "Tabellenname: " & Accobj.Name
      Debug.Print "Geöffnet    : " & Accobj.IsLoaded & vbLf
   Next Accobj
End Sub
```

Listing 146: Tabellennamen dokumentieren

Über die Eigenschaft `CurrentData` haben Sie Zugriff auf einige Objekte, die sich in der geöffneten Datenbank befinden. Unter diesen Objekten befinden sich auch Tabellen, die Sie über die Auflistung `AllTables` ansprechen können. Mithilfe der Eigenschaft `Name` ermitteln Sie den Namen der jeweiligen Tabelle. Die Funktion `IsLoaded` gibt Auskunft darüber, ob eine Tabelle momentan geöffnet ist. Wenn ja, dann meldet diese Funktion den Rückgabewert `True`.

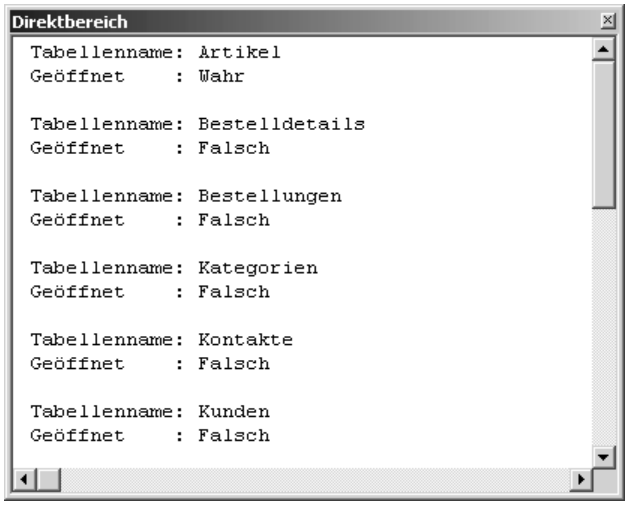


Abbildung 82: Tabellen auflisten und Status prüfen

132 Alle Abfragen auflisten

Über den Einsatz des `AccessObject` können Sie alle Abfragen der Datenbank anzeigen lassen. Im Makro aus Listing 147 werden die Namen aller in der Datenbank befindlichen Abfragen dokumentiert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'=================================================

Sub AbfragenAuflisten()
   Dim AccObj As AccessObject
   Dim dbs As CurrentData

   Set dbs = Application.CurrentData

   For Each AccObj In dbs.AllQueries
      Debug.Print AccObj.Name
   Next AccObj
End Sub
```

Listing 147: Alle Abfragen dokumentieren

In einer `For each next`-Schleife durchlaufen Sie Abfragen und ermitteln die Namen der Abfragen über die Eigenschaft `Name`.

133 Die Application-Objekte

Das `Application`-Objekt steht an oberster Ebene. Es bezieht sich auf die aktive Microsoft Access-Anwendung und beinhaltet alle darunter liegenden Objekte wie Formulare, Reports, Drucker und Bildschirm.

134 Name der aktuellen Datenbank abfragen

Soll der Name der aktuell geöffneten Datenbank abgefragt werden, dann können Sie diese Aufgabe über das Makro aus Listing 148 durchführen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=================================================
```

Listing 148: Name der Datenbank ermitteln – Variante 1

```
Sub DatenbankName()
   MsgBox "Die aktuelle Datenbank heißt: " & _
   Application.CurrentProject.Name
End Sub
```

Listing 148: Name der Datenbank ermitteln – Variante 1 (Forts.)

Über die Eigenschaft `Name`, welche Sie auf das Objekt `CurrentProject` anwenden, können Sie den Namen der aktuellen Datenbank ermitteln.

Eine zweite Variante, um den Namen der Datenbank zu ermitteln, sehen Sie im Makro aus Listing 149.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DatenbankName2()
   MsgBox "Die aktuelle Datenbank heißt: " & _
   Application.CurrentDb.Name
End Sub
```

Listing 149: Name der Datenbank ermitteln – Variante 2

Über die Eigenschaft `Name`, welche Sie auf das Objekt `CurrentDb` anwenden, können Sie den Namen der aktuellen Datenbank ermitteln.

135 Pfad der aktuellen Datenbank abfragen

Um den Pfad der aktuell geöffneten Datenbank abzufragen, setzen Sie die Eigenschaft `Path` wie in Listing 150 gezeigt, ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub PfadAusgeben()
   MsgBox "Die aktuelle Datenbank heißt: " & _
   Application.CurrentProject.Path
End Sub
```

Listing 150: Pfad der Datenbank ermitteln

Um beide Informationen, also den Pfad sowie den Namen der aktuellen Datenbank abzufragen, verwenden Sie die Eigenschaft `FullName`.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'====================================================

Sub PfadUndNameAusgeben()
   MsgBox "Die aktuelle Datenbank heißt: " & _
   Application.CurrentProject.FullName
End Sub
```

Listing 151: Pfad und Dateiname der Datenbank ermitteln

Abbildung 83: Pfad und Dateiname abfragen

136 Pfad der Anwendung abfragen

Durch den Einsatz der Methode `SysCmd` können Sie den Pfad der Anwendung ermitteln. Im Beispiel aus Listing 152 wird der Installationspfad von Microsoft Access ermittelt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'====================================================

Sub InstallationsPfadAusgeben()
   MsgBox "Das Installationsverzeichnis von Access lautet: " & _
      vbLf & SysCmd(acSysCmdAccessDir), vbInformation
End Sub
```

Listing 152: Den Pfad der Anwendung abfragen

Übergeben Sie der Methode `SysCmd` als Konstante `acSysCmdAccessDir`, um das Installationsverzeichnis von Microsoft Access abzufragen.

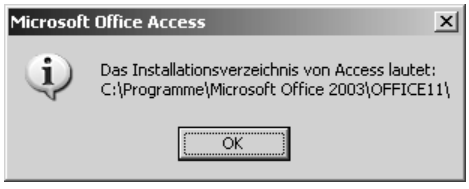


Abbildung 84: Den Installationspfad von Access ausgeben

137 Access-Version feststellen

Ebenso geeignet, um die eingesetzte Access-Version zu ermitteln, ist der Einsatz der Methode `SysCmd`. Im folgenden Beispiel aus Listing 153 wird die Access-Versionsnummer am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub AccessVersionsNrAusgeben()
   MsgBox "Die eingesetzte Access-Version lautet: " & _
      vbLf & SysCmd(acSysCmdAccessVer), vbInformation
End Sub
```

Listing 153: Die eingesetzte Access-Version abfragen

Übergeben Sie der Methode `SysCmd` als Konstante `acSysCmdAccessVer`, um die Access-Versionsnummer zu ermitteln.

Abbildung 85: Die Versionsabfrage wurde durchgeführt.

Hinweis

Die Versionsnummer 11.0 steht für Microsoft Access 2003.

138 Aktuellen Anwendernamen abfragen

Mithilfe der Methode `CurrentUser` können Sie den Namen des aktuellen Benutzers der Datenbank zurückgeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=================================================

Sub AnwenderNamenAusgeben()
   MsgBox "Der aktuelle Benutzer ist: " & _
      Application.CurrentUser, vbInformation
End Sub
```

Listing 154: Den aktuellen Anwendernamen abfragen

139 Aktuelle Datenbank schließen

Soll die aktuelle Datenbank geschlossen werden, dann können Sie für diese Aufgabe die Methode `CloseCurrentDatabase` einsetzen. Diese Methode kann unter anderem dann eingesetzt werden, wenn Sie eine Microsoft Access-Datenbank von einer anderen Anwendung aus über Automatisierung geöffnet haben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=================================================

Sub DatenbankBeenden()
   Application.CloseCurrentDatabase
End Sub
```

Listing 155: Aktuelle Datenbank schließen

140 Applikation beenden

Möchten Sie Microsoft Access beenden, dann setzen Sie die Methode `Quit` ein, welche Sie auf das Objekt `Application` anwenden. Im Makro aus Listing 156 wird die Anwendung beendet und alle Änderungen an der aktuellen Datenbank werden vorher gesichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub ApplikationBeenden()
   Application.Quit acQuitSaveAll
End Sub
```

Listing 156: Applikation beenden

Die Methode `Quit` kann mit folgenden Optionen eingesetzt werden:

- `acQuitPrompt`: Zeigt ein Dialogfeld an, in dem Sie gefragt werden, ob geänderte, aber nicht gesicherte Objekte vor dem Beenden von Microsoft Access gespeichert werden sollen.
- `acQuitSaveAll`: Speichert alle Objekte, ohne ein Dialogfeld anzuzeigen. (Standardeinstellung).
- `acQuitSaveNone`: Beendet Microsoft Access, ohne Objekte zu speichern.

141 Drucker auflisten

Um zu ermitteln, welche Drucker Sie im Einsatz haben und an welchem Anschluss diese hängen, können Sie das neue Auflistungsobjekt `Printers` nutzen, das Sie im Zusammenspiel mit dem Objekt `Application` einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DruckerListen()
   Dim Drucker As Printer

   For Each Drucker In Application.Printers
      With Drucker
         Debug.Print "Druckername: " & .DeviceName & vbCr _
            & "Anschluss: " & .Port
      End With
   Next Drucker
End Sub
```

Listing 157: Drucker und Ports auflisten

Die Eigenschaft `DeviceName` zeigt den Druckernamen an. Mithilfe der Eigenschaft `Port` können Sie den Anschluss anzeigen, dem Ihr Drucker zugeordnet ist.

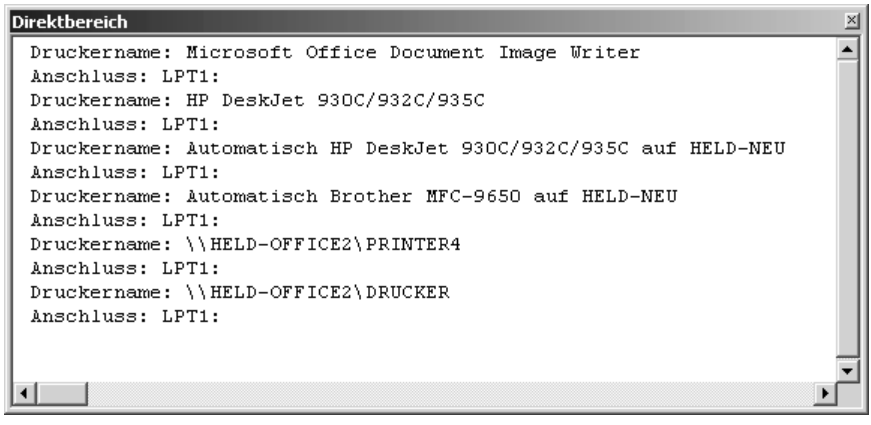


Abbildung 86: Die verfügbaren Drucker mit Anschlussbelegung

142 Access-Version abfragen

Über die Eigenschaft `Version`, die Sie auf das Objekt `Application` anwenden, können Sie die eingesetzte Access-Version ermitteln.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub VersionAusgeben()
   MsgBox "Sie arbeiten mit der Access-Version: " _
      & Application.Version
End Sub
```

Listing 158: Die eingesetzte Access-Version ermitteln

143 Das Control-Objekt

Das Objekt `Control` stellt ein Steuerelement dar, das sich in einem Formular, einem Bericht, einem Bereich oder innerhalb eines anderen Steuerelements befindet oder einem anderen Steuerelement beigefügt ist. Im folgenden Beispiel aus Listing 159 werden alle Textfelder des Formulars `Artikel` im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCtl
'=====================================================
```

Listing 159: Alle Textfelder in einem Formular auslesen

```
Sub SteuerelementeDokumentieren()
   Dim ctrl As Control
   Dim frm As Form
   Set frm = Form_Kunden

   For Each ctrl In frm.Controls
      If TypeName(ctrl) = "Textbox" Then
         Debug.Print ctrl.Name
      End If
   Next ctrl
End Sub
```

Listing 159: Alle Textfelder in einem Formular auslesen (Forts.)

Über die Anweisung `Set` geben Sie zunächst einmal an, wie das Formular heißt. Danach setzen Sie eine `For Each Next`-Schleife auf, die alle Steuerelemente des Formulars abarbeitet. Innerhalb der Schleife prüfen Sie über den Einsatz der Funktion `TypeName`, ob es sich bei dem jeweiligen Steuerelement um ein Textfeld handelt. Wenn ja, dann meldet diese Funktion den Ausdruck `Textbox` zurück. In diesem Fall geben Sie den Namen des Steuerelements über die Eigenschaft `Name` aus.

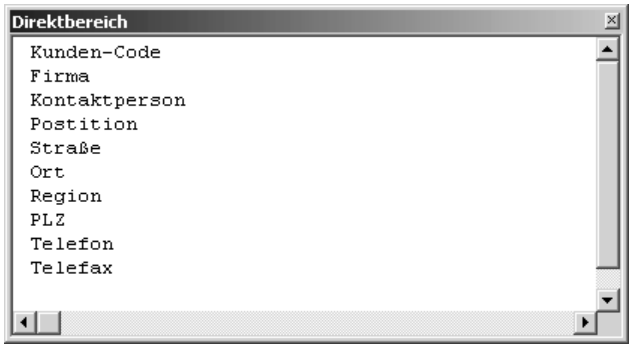


Abbildung 87: Die Textfelder eines Formulars wurden dokumentiert.

144 Das DoCmd-Objekt

Über den Einsatz des Objekts `DoCmd` können Sie Aktionen wie das Schließen von Fenstern, das Öffnen von Formularen und das Festlegen der Werte von Steuerelementen ausführen.

Auf den nächsten Seiten erfolgt eine Auswahl von typischen Aufgaben, die in Verbindung mit diesem Objekt durchgeführt werden können.

145 Filter in Tabellen setzen

Mithilfe der Methode `ApplyFilter` können Sie einen Filter hinzufügen. Dabei setzen Sie diesen Filter gewöhnlich in Tabellen oder Formularen ein. Die Syntax dieser Methode lautet:

```
Ausdruck.ApplyFilter(Filtername, Bedingung)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
Filtername	Optional. Legt den Namen eines Filters oder einer Abfrage in der aktuellen Datenbank fest.
Bedingung	Optional. Legt die Bedingung fest

Tabelle 57: Die Argumente der Methode ApplyFilter

Im folgenden Beispiel aus Listing 160 wird die Tabelle Artikel geöffnet. Danach werden alle Artikel ausgefiltert, die einen Lagerbestand kleiner 10 aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub FilterSetzen()
   DoCmd.OpenTable "Artikel"
   DoCmd.ApplyFilter , "Lagerbestand < 10"
End Sub
```

Listing 160: Einen Tabellenfilter setzen

Über die Methode OpenTable öffnen Sie die Tabelle, danach wenden Sie die Methode ApplyFilter an, um den Filter zu setzen. Das Filterkriterium wird dabei in Anführungszeichen angegeben.

Abbildung 88: Alle Artikel mit einem Lagerbestand von weniger als zehn Artikeln werden angezeigt.

Sollen beispielsweise gleich mehrere Artikel ausgefiltert werden, dann können Sie diese Aufgabe über das Makro aus Listing 161 lösen. Dabei werden alle Artikel mit dem Namen TOFU oder KONBU in der Tabelle Artikel gefiltert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub FilterSetzen2()
   DoCmd.OpenTable "Artikel"
   DoCmd.ApplyFilter , "Artikelname = 'Tofu' OR Artikelname= 'Konbu'"
End Sub
```

Listing 161: Mehrere Bedingungen beim Filtern angeben

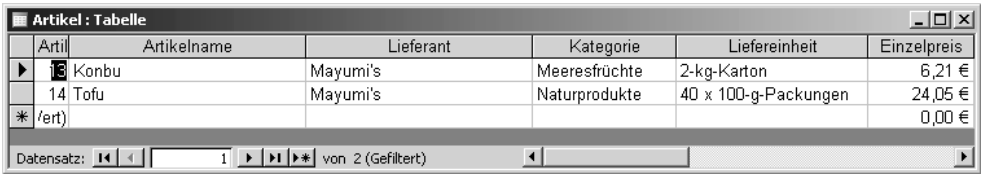


Abbildung 89: Die Artikel Konbu und Tofu wurden gefiltert.

Hinweis

Durch die Methode `ShowAllRecords` werden alle vorhandenen Filter entfernt, die für die aktuelle Tabelle, die aktuelle Abfrage oder das aktuelle Formular eingerichtet wurden. Diese Methode hat keine weiteren Argumente.

146 Objekte umbenennen

Mithilfe der Methode `Rename` können Sie Objekte wie Tabellen, Formulare und sonstige umbenennen. Die Syntax dieser Methode lautet:

```
Ausdruck.Rename(NewName, ObjectType, OldName)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`NewName`	Erforderlich. Legt den neuen Namen des umzubenennenden Objekts fest.
`ObjectType`	Optional. Gibt den Typ des umzubenennenden Objekts an.
`OldName`	Optional. Gibt den alten Namen des Objekts an.

Tabelle 58: Die Argumente der Methode Rename

Im folgenden Beispiel aus Listing 162 wird eine Tabelle umbenannt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=================================================

Sub TabelleUmbenennen()
   On Error GoTo fehler
   DoCmd.Rename "ArtikelNeu", acTable, _
      "ArtikelAktuell"
   Exit Sub

   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 162: Eine Tabelle umbenennen

Sollte die Umbenennung nicht erfolgreich sein, dann können Sie anstatt eines Makroabsturzes über die Anweisung `On Error GoTo` zu einer Fehlerbehandlung übergehen. Dabei werten Sie das Objekt `Err` aus, indem Sie über die Eigenschaften `Number` und `Description` die Fehlernummer sowie eine Beschreibung des Fehlers am Bildschirm anzeigen.

147 Objekte kopieren

Mithilfe der Methode `CopyObject` können Sie Objekte wie Tabellen und Berichte kopieren. Die Syntax dieser Methode lautet:

```
Ausdruck.CopyObject(Zieldatenbank, NeuerName, Quellobjekttyp, Quellobjektname)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Zieldatenbank`	Optional. Gibt den gültigen Pfad und Dateinamen für die Datenbank an, in die das Objekt kopiert werden soll.
`NeuerName`	Optional. Legt den neuen Namen für das zu kopierende Objekt fest. Sie können beim Kopieren in eine andere Datenbank denselben Namen verwenden, indem Sie dieses Argument nicht angeben.
`Quellobjekttyp`	Optional. Gibt an, um welchen Objekttyp es sich handelt.
`Quellobjektname`	Optional. Gibt den Namen des Quellobjekts an.

Tabelle 59: Die Argumente der Methode CopyObject

Beim folgenden Beispiel aus Listing 163 wird die Tabelle `Artikel` kopiert und der Namen `ArtikelAktuell` vergeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleKopieren()
   DoCmd.CopyObject , "ArtikelAktuell", acTable, _
      "Artikel"
End Sub
```

Listing 163: Tabelle in aktueller Datenbank kopieren

Soll die Tabelle `Artikel` in eine andere Datenbank kopiert werden, die momentan nicht geöffnet ist, dann wenden Sie das Makro aus Listing 164 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleKopierenInAndereDB()
   On Error GoTo fehler
   DoCmd.CopyObject "C:\Eigene Dateien\BackUp.mdb", _
      "ArtikelAktuell", acTable, "Artikel"
   Exit Sub

   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 164: Tabelle in andere Datenbank kopieren

Sollte der Kopiervorgang nicht erfolgreich sein, dann können Sie anstatt eines Makroabsturzes über die Anweisung `On Error GoTo` zu einer Fehlerbehandlung übergehen. Dabei werten Sie das Objekt `Err` aus, indem Sie über die Eigenschaften `Number` und `Description` die Fehlernummer sowie eine Beschreibung des Fehlers am Bildschirm anzeigen.

Abbildung 90: Den Fehlerfall abfangen über das Objekt Err.

148 Objekte löschen

Über die Methode `DeleteObject` können Sie ein Objekt wie beispielsweise eine Tabelle oder ein Formular aus einer Datenbank entfernen. Die Syntax dieser Methode lautet:

```
Ausdruck.DeleteObject(Objekttyp, Objektname)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Objekttyp`	Gibt den Objekttyp an.
`Objektname`	Optional. Gibt den Namen des Objekts bekannt, das gelöscht werden soll.

Tabelle 60: Die Argumente der Methode DeleteObject

Im folgenden Beispiel aus Listing 165 wird die Tabelle `ArtikelAktuell` aus der aktuellen Datenbank entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleEntfernen()
   On Error GoTo fehler
   DoCmd.DeleteObject acTable, "ArtikelAktuell"
   Exit Sub

   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 165: Eine Tabelle aus der aktuellen Datenbank entfernen

149 Objekte exportieren

Mit der Methode `OutputTo` können Sie die Daten in einem bestimmten Microsoft Access-Datenbankobjekt (einem Datenblatt, einem Formular, einem Bericht, einem Modul oder einer Datenzugriffsseite) in verschiedenen Formaten ausgeben. Die Syntax dieser Methode lautet:

```
OutputTo(ObjectType, ObjectName, OutputFormat, OutputFile, AutoStart, TemplateFile)
```

Argument	Beschreibung
ObjectType	Erforderlich. Legt die Art des Access-Objekts fest, dessen Daten Sie exportieren möchten. Dabei haben Sie folgende Möglichkeiten: acOutputForm: Export der Daten eines Formulars acOutputFunction: Export einer Funktion acOutputModule: Export eines kompletten Moduls inklusive aller Funktionen und Makros acOutputQuery: Export der Ergebnisse einer Abfrage acOutputReport: Export eines Berichts acOutputServerView: Export einer Serveransicht acOutputStoredProcedure: Export einer gespeicherten Prozedur acOutputTable: Export einer Tabelle
ObjectName	Optional. Gibt den Namen des Objekts an, das Sie exportieren möchten.
OutPutFormat	Optional. Legt fest, in welchem Datenformat Sie die Daten transferieren. Die bekanntesten Formate sind Folgende: acFormatHTML: Konvertiert die Daten in das HTML-Format. acFormatRTF: Konvertiert die Daten in das Rich-Text-Format. Dieses Format kann beispielsweise problemlos in Microsoft Word eingelesen werden. acFormatTXT: Mit diesem Format ist das Textformat gemeint. acFormatXLS: Konvertiert die Daten in das Microsoft Excel-Format.
OutputFile	Optional. Der Pfad sowie den Dateinamen der Datei, in welche Sie die Daten transferieren möchten. Dabei muss die Datei noch nicht vorhanden sein. Access legt diese bei Bedarf selber an.
AutoStart	Optional. Damit haben Sie die Möglichkeit, die so erstellte Exportdatei gleich zu öffnen. Verwenden Sie den Wert True, um die entsprechende auf Windows basierende Anwendung sofort zu starten. Setzen Sie das Argument auf den Wert False oder lassen Sie es weg, wenn Sie die Exportdatei nicht öffnen möchten.
TemplateFile	Optional. Dieses Argument ist dann von Interesse, wenn Sie eine Vorlage beispielsweise für die HTML-Datei verwenden möchten. In diesem Fall ist dann der komplette Pfad dieser Vorlagendatei anzugeben.

Tabelle 61: Die Argumente der Methode OutPutTo

Im folgenden Beispiel aus Listing 166 wird die Tabelle Artikel in eine Textdatei geschrieben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
```

Listing 166: Tabelle in eine Textdatei exportieren

```
'=================================================

Sub TabelleExportieren()
   On Error GoTo fehler
   DoCmd.OutputTo acOutputTable, "Artikel", _
      acFormatTXT, "C:\Eigene Dateien\Artikel.txt", True
   Exit Sub

   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 166: Tabelle in eine Textdatei exportieren (Forts.)

Geben Sie bei der Methode `OutputTo` den Objekttyp, den Namen des Objekts, das gewünschte Ausgabeformat sowie den Namen der Zieldatei an. Dabei wird die Zieldatei, sofern sie noch nicht existiert, neu angelegt. Im letzten Argument der Methode geben Sie den Wert `True` an, was bedeutet, dass die Zieldatei nach dem Export angezeigt werden soll.

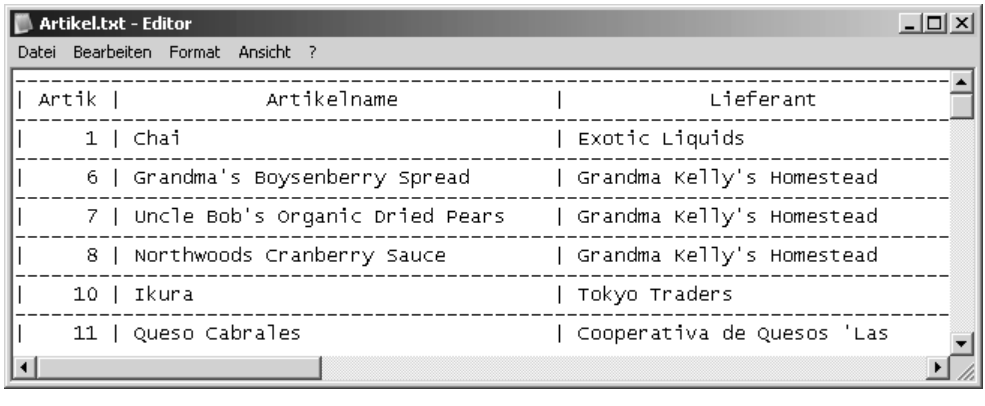


Abbildung 91: Die Tabelle liegt nun als Textdatei vor.

150 Objekte drucken

Über die Methode `PrintOut` können Sie einen Drucken-Befehl in Access ausführen. Die Syntax dieser Methode lautet:

```
Ausdruck.PrintOut(Druckbereich, Von, Bis, Druckqualität, Exemplare,
ExemplareSortieren)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Druckbereich`	Optional. `acPages`, `acPrintAll` (Standard), `acSelection`

Tabelle 62: Die Argumente der Methode PrintOut

Argument	Beschreibung
Von	Optional. Gibt die erste Seite des Bereichs an, ab der gedruckt werden soll. Dieses Argument ist erforderlich, wenn Sie für das Argument Druckbereich die Konstante acPages angeben.
Bis	Optional. Gibt die letzte Seite an, bis zu der gedruckt werden soll. Dieses Argument ist erforderlich, wenn Sie für das Argument Druckbereich die Konstante acPages angeben.
Druckqualität	Optional. Dabei gelten folgende Konstanten: acDraft (Entwurfsqualität) acHigh (bestmögliche Druckqualität) acLow (Konzeptdruckqualität) acMedium (mittlere Druckqualität)
Exemplare	Optional. Gibt die Anzahl der Kopien an.
ExemplareSortieren	Optional. Verwenden Sie True, um die Exemplare zu sortieren, und False, um die Exemplare nicht während des Druckvorgangs zu sortieren.

Tabelle 62: Die Argumente der Methode PrintOut (Forts.)

Im folgenden Beispiel aus Listing 167 wird ein Bericht in der aktuell geöffneten Datenbank geöffnet und es werden einige Seiten davon ausgedruckt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub BerichtDrucken()
   Application.Echo False
   DoCmd.OpenReport "Rechnung", acViewPreview
   DoCmd.PrintOut acPages, 1, 1, , 2
   DoCmd.Close
   Application.Echo True
End Sub
```

Listing 167: Bericht drucken

Über die Methode Echo können Sie festlegen, ob die Bildschirmaktualisierung während des Makroablaufs eingeschaltet bzw. ausgeschaltet wird. Wenn Sie diese Methode mit dem Argument False einsetzen, dann erfolgt der Druckvorgang gänzlich im Hintergrund. Nach dem Drucken der Seiten wenden Sie die Methode Close an, um den Bericht wieder zu schließen, und schalten Sie die Bildschirmaktualisierung wieder ein.

151 Objekte versenden

Für das Versenden von E-Mails können Sie in Access die Methode `SendObject` einsetzen. Dabei können Sie ganz genau festlegen, welchen Bestandteil einer Access-Datenbank Sie versenden möchten. Ferner übergeben Sie dieser Methode die Adressaten sowie den Begleittext der E-Mail. Die Syntax dieser Methode lautet:

```
SendObject(Objekttyp, Objektname, Ausgabeformat, An, Cc, Bcc, Betreff, Nachricht,
NachrichtBearbeiten, Vorlagedatei)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Objekttyp`	Optional. Dabei geben Sie in einer Konstanten an, welchen Bestandteil der Datenbank Sie per E-Mail versenden möchten. Folgende Konstanten stehen Ihnen dabei zur Verfügung: `acSendDataAccessPage`: Eine Access-Datenzugriffsseite wird einem E-Mail-Empfänger zugestellt. `acSendForm`: Ein Formular soll über eine E-Mail versendet werden. `acSendModule`: Ein Modul wird per E-Mail versendet. `acSendNoObject`: Es wird lediglich eine Text-E-Mail, ohne Anhang, versendet. Es handelt sich dabei um die Standardeinstellung. `acSendQuery`: Hierbei soll eine Abfrage per E-Mail versendet werden. `acSendReport`: Bei dieser Angabe wird ein Bericht versendet. `acSendTable`: Diese Konstante steht für das Versenden einer bestimmten Tabelle aus einer Datenbank.
`Objektname`	Optional. Hier muss der Name des Objekts angegeben werden, der per E-Mail versendet werden soll.
`Ausgabeformat`	Optional. Hier können Sie festlegen, in welcher Form das Access-Objekt versendet werden soll. Dabei haben Sie unter anderem die Auswahl zwischen folgenden Konstanten: `acFormatHTML`: Ausgabe des Access-Objekts über das HTML-Format, das Sie mit jedem Browser ansehen können. `acFormatRTF`: Beim RTF-Format handelt es sich um ein Textformat, das Sie mit nahezu jedem Textverarbeitungsprogramm öffnen können. `acFormatTXT`: Dieses Textformat ist mit jedem Texteditor, beispielsweise Notepad im Zubehör von Windows, zu lesen. `acFormatXLS`: Dabei handelt es sich um das Excel-Tabellenformat. `AcFormatDAP`: Bei dieser Konstante handelt es sich um Datenzugriffseiten.
`An`	Optional. Hier müssen Sie die Empfänger auflisten, deren Namen in die `An`-Zeile der E-Mail-Nachricht aufgenommen werden sollen. Die Empfängernamen in diesem Argument müssen durch Semikola (;) voneinander getrennt werden.

Tabelle 63: Die Argumente der Methode SendObject

Argument	Beschreibung
Cc	Optional. Gibt an, an welche E-Mail-Empfänger Sie die E-Mail als Kopie schicken möchten. Es gelten dabei dieselben Optionen wie auch beim Argument An.
Bcc	Optional. Hier können Sie E-Mail-Empfänger eine »blinde Kopie« der E-Mail schicken, ohne dass der eigentliche Empfänger der E-Mail, der unter dem Argument An angegeben wurde, etwas davon erfährt.
Betreff	Optional. Repräsentiert die Betreff-Zeile der E-Mail. Geben Sie dort einen Betreff in doppelten Anführungsstrichen an.
Nachricht	Optional. Geben Sie den Text an, der in die E-Mail eingefügt werden soll. Wenn Sie dieses Argument nicht angeben, wird nur das Objekt, jedoch kein Text in die E-Mail aufgenommen.
NachrichtBearbeiten	Optional. Hier können Sie entscheiden, ob Sie die E-Mail direkt absenden oder zur weiteren Bearbeitung vorher öffnen möchten. Setzen Sie dieses Argument auf den Wert False, um die Nachricht direkt zu versenden. Setzen Sie das Argument auf den Wert True, um die E-Mail zur weiteren Bearbeitung zu öffnen.
Vorlagedatei	Optional. Hier handelt es sich um einen optionalen Wert, der den vollständigen Namen und Pfad der Datei angibt, die als Vorlage für eine HTML-Datei verwendet werden soll.

Tabelle 63: Die Argumente der Methode SendObject (Forts.)

Im folgenden Beispiel aus Listing 168 wird eine E-Mail ohne Dateianhang versendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TextMailVersenden()
   On Error GoTo fehler
   DoCmd.SendObject , , , _
      "Held-office@t-online.de", _
      "Machero@aol.com", , "Feedback erwünscht", _
      "Sehr geehrter KundeXY," & vbCrLf & _
      "Bitte schicken Sie mir ein Feedback" & _
      vbCrLf & "zu meiner Anfrage vom 20.07.2004" & _
      vbCrLf & vbCrLf & _
      "Viele Grüße" & vbCrLf & _
      "Bernd Held", True
      Exit Sub
```

Listing 168: Eine einfache Text-E-Mail versenden

```
    fehler:
       MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 168: Eine einfache Text-E-Mail versenden (Forts.)

Da diese Art von E-Mail keinen Dateianhang hat, lassen Sie die ersten drei Argumente der Methode `SendObject` einfach leer. Danach geben Sie den Adressaten sowie den Kopieempfänger der E-Mail, den Titel und den eigentlichen Text der E-Mail an. Mithilfe der Konstanten `vbCrLf` können Sie im E-Mail-Text einen Zeilenumbruch erzeugen. Über das letzte Argument, das Sie auf `True` oder `False` setzen können, bestimmen Sie, ob die E-Mail direkt versendet oder noch einmal vor dem Versand zur Kontrolle angezeigt werden soll.

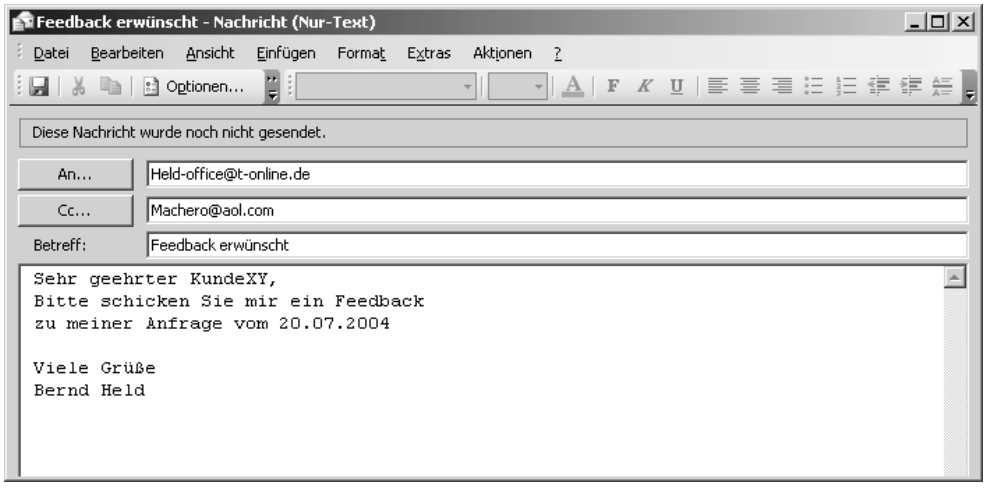


Abbildung 92: Eine Text-E-Mail ohne Anhang erstellen

Im nächsten Beispiel aus Listing 169 wird die Tabelle `Artikel` im Excel-Format gleich an mehrere Anwender per E-Mail verschickt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=======================================================

Sub MailMitAnhangVersenden()
   On Error GoTo fehler
   DoCmd.SendObject acSendTable, _
      "Artikel", acFormatXLS, _
      "Held-office@t-online.de; Machero@aol.com", _
      , , "Die aktuellen Top-Artikel", _
```

Listing 169: E-Mail mit Dateianhang versenden

```
        "Hallo Kollegen, " & vbCrLf & _
        "Anbei die aktuell gut laufenden Artikel!" & _
        vbCrLf & "Viele Grüße" & vbCrLf & _
        "Fritz Meier", True
    Exit Sub


    fehler:
       MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 169: E-Mail mit Dateianhang versenden (Forts.)

Legen Sie in den ersten drei Argumenten der Methode `SendObject` den Objekttyp, den Objektnamen und das gewünschte Ausgabeformat an. Danach geben Sie den Adressaten sowie den Kopieempfänger der E-Mail, den Titel und den eigentlichen Text der E-Mail an.

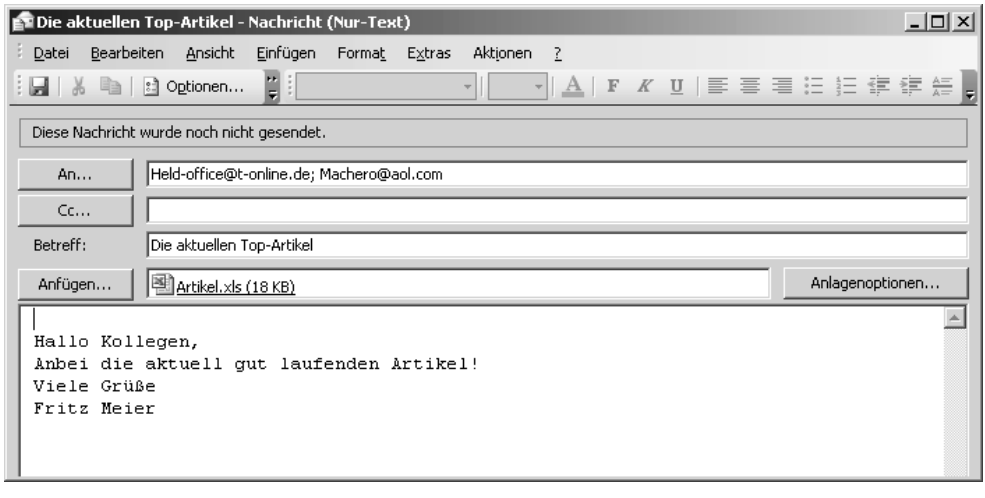


Abbildung 93: Die Tabelle Artikel wird im Excel-Format verschickt.

152 Feld aktivieren im Formular

Über die Methode `GoToControl` können Sie den Fokus in einem Formular auf ein bestimmtes Feld setzen. Die Syntax dieser Methode lautet:

```
Ausdruck.GoToControl(Steuerelementname)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Steuerelementname`	Erforderlich. Gibt den Namen des Steuerelements im aktiven Formular oder Datenblatt an.

Tabelle 64: Die Argumente der Methode GoToControl

Im folgenden Beispiel aus Listing 170 wird das Formular KUNDEN geöffnet und der Fokus in das Feld `Firma` gesetzt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=================================================

Sub FeldAnspringen()
   DoCmd.OpenForm "Kunden"
   DoCmd.GoToControl "Firma"
End Sub
```

Listing 170: Ein bestimmtes Feld im Formular anspringen

Über die Methode `OpenForm` öffnen Sie das Formular KUNDEN. Danach wenden Sie die Methode `GoToControl` an, um das gewünschte Feld zu aktivieren.

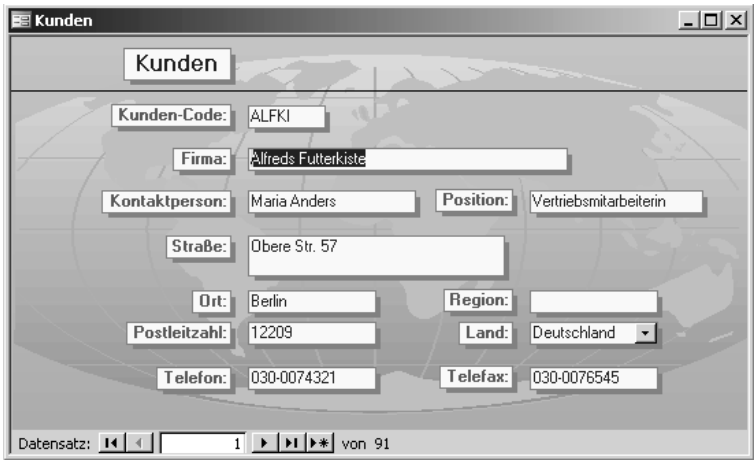


Abbildung 94: Der Fokus wurde auf das Feld Firma gesetzt.

153 Datensatz in Tabelle aktivieren

Über die Methode `GoToRecord` können Sie einen bestimmten Satz in einem Formular oder einer Tabelle vorab einstellen. Die Syntax dieser Methode lautet:

```
Ausdruck.GoToRecord(Objekttyp, Objektname, Datensatz, Offset)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
Objekttyp	Gibt den Objekttyp an. Es gelten dabei: acActiveDataObject (Standard), acDataForm, acDataFunction, acDataQuery, acDataServerView, acDataStoredProcedure oder acDataTable.
Objektname	Optional. Gibt den Namen des Objekts an.
Datensatz	Optional. Dabei gelten folgende Konstanten: acFirst, acGoTo, acLast, acNewRec, acNext (Standard), acPrevious
Offset	Optional. Eine Zahl, die die Anzahl der Datensätze angibt, um die vorwärts oder rückwärts geblättert werden soll, wenn Sie für das Argument Datensatz die Optionen acNext oder acPrevious angegeben haben, oder der Datensatz, zu dem Sie wechseln möchten, wenn Sie für das Argument Datensatz acGoTo angegeben haben. Der Ausdruck muss eine gültige Datensatznummer ergeben.

Tabelle 65: Die Argumente der Methode GoToRecord

Im Beispiel aus Listing 171 wird die Tabelle Artikel geöffnet und der letzte Satz voreingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub LetzterSatzEinstellen()
   DoCmd.OpenTable "Artikel"
   DoCmd.GoToRecord acDataTable, "Artikel", acLast
End Sub
```

Listing 171: Den letzten Satz einer Tabelle voreinstellen

154 Menübefehle ausführen

Mithilfe der Methode RunCommand können Sie jeden möglichen Befehl in Access, den Sie über die Menüs ansteuern können, auch per Makro ausführen lassen. Diese Methode hat demzufolge mehrere hundert Konstanten, über die Sie steuern können, welcher Menübefehl ausgeführt werden soll. Die Syntax dieser Methode lautet:

```
Ausdruck.RunCommand(Command)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
Command	Über eine Konstante aus Tabelle 67 können Sie den gewünschten Befehl ausführen.

Tabelle 66: Die Argumente der Methode RunCommand

Entnehmen Sie aus der folgenden Tabelle exemplarisch ein paar wichtige Command-Konstanten.

Konstante	Beschreibung
acCmdAboutMicrosoftAccess	Ruft den Info-Dialog auf.
acCmdAnswerWizard	Zeigt die Direkthilfe an.
acCmdCloseWindow	Schließt das aktuelle Fenster.
acCmdDatabaseProperties	Zeigt die Datenbankeigenschaften an.
acCmdDocMinimize	Minimiert das Datenbankfenster.
acCmdDocMaximize	Maximiert das Datenbankfenster.
acCmdExit	Beendet Microsoft Access.
acCmdFind	Das Suchenfenster wird angezeigt.
acCmdNewDatabase	Legt eine neue Datenbank an.
acCmdOpenDatabase	Zeigt den ÖFFNEN-Dialog an.
acCmdOpenTable	Öffnet eine Tabelle.
acCmdOptions	Zeigt den Dialog OPTIONEN an.
acCmdPrint	Zeigt den Dialog DRUCKEN an.
acCmdQuickPrint	Druckt sofort, ohne den Dialog DRUCKEN anzuzeigen.
acCmdRedo	Wiederholt die letzte Aktion.
acCmdUndo	Widerruft die letzte Aktion.
acCmdRelationships	Öffnet das Fenster, welches die Beziehungen der Tabellen anzeigt.
acCmdSend	Öffnet den E-Mail-Dialog.

Tabelle 67: Die wichtigsten Command-Konstanten der Methode RunCommand

Im folgenden Beispiel aus Listing 172 wird der Dialog DOKUMENTEIGENSCHAFTEN aufgerufen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub DokumenteigenschaftenAnzeigen()
   DoCmd.RunCommand acCmdDatabaseProperties
End Sub
```

Listing 172: Dokumenteigenschaften aufrufen

155 Symbolleisten ein- und ausblenden

Über die Methode `ShowToolbar` können Sie eine Symbolleiste einblenden. Die Syntax dieser Methode lautet:

```
Ausdruck.ShowToolbar(Symbolleistenname, Anzeigen)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Symbolleistenname`	Erforderlich. Gibt den Namen einer integrierten Symbolleiste von Microsoft Access oder einer von Ihnen erstellten benutzerdefinierten Symbolleiste an.
`Anzeigen`	Optional. Folgende Konstanten stehen zur Verfügung: `acToolbarNo`, `acToolbarWhereApprop` und `acToolbarYes` (Standard).

Tabelle 68: Die Argumente der Methode ShowToolbar

Im folgenden Beispiel aus Listing 173 wird die Symbolleiste WEB ein- und wieder ausgeblendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub SymbolleisteEinblenden()
   DoCmd.ShowToolbar "Web", acToolbarYes
End Sub


Sub SymbolleisteAusblenden()
   DoCmd.ShowToolbar "Web", acToolbarNo
End Sub
```

Listing 173: Symbolleiste ein- und ausblenden

156 SQL-Anweisungen absetzen

Über die Methode RunSQL können Sie eine SQL-Anweisung absetzen, um Abfragen zu starten. Dabei sind Anfügeabfragen, Löschabfragen, Tabellenerstellungsabfragen sowie Aktualisierungsabfragen möglich. Die Syntax dieser Methode lautet:

```
Ausdruck.RunSQL(SQLAnweisung, TransaktionVerwenden)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
SQLAnweisung	Erforderlich. Ein Zeichenfolgenausdruck, der einer gültigen SQL-Anweisung für eine Aktionsabfrage oder eine Datendefinitionsabfrage entspricht. Verwendet werden kann eine der folgenden Anweisungen: INSERT INTO, DELETE, SELECT...INTO, UPDATE, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX oder DROP INDEX. Fügen Sie eine in-Klausel hinzu, wenn Sie auf eine andere Datenbank zugreifen möchten.
TransaktionVerwenden	Optional. Verwenden Sie True, um diese Abfrage in eine Transaktion einzubinden, oder False, wenn Sie keine Transaktion durchführen möchten.

Tabelle 69: Die Argumente der Methode RunSQL

Im folgenden Beispiel aus Listing 174 wird die Methode RunSQL eingesetzt, um in der Tabelle Kunden im Feld Position alle Vorkommen von Inhaber in Chef umzusetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub SQLAnweisungStarten()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "UPDATE Kunden " & _
      "SET Kunden.Position = 'Chef' " & _
      "WHERE Kunden.Position = 'Inhaber'"
   DoCmd.RunSQL strSQL
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 174: Eine SQL-Anweisung absetzen

157 Fenstergröße festlegen

Mithilfe der Methode `MoveSize` können Sie die Fensterbreite des Access-Fensters einstellen. Die Syntax dieser Methode lautet:

```
Ausdruck.MoveSize(Rechts, Unten, Breite, Höhe)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Rechts`	Optional. Die neue horizontale Position der oberen linken Ecke des Fensters, gemessen vom linken Rand des umgebenden Fensters.
`Unten`	Optional. Die neue vertikale Position der oberen linken Ecke des Fensters, gemessen vom oberen Rand des umgebenden Fensters.
`Breite`	Optional. Legt die neue Breite des Fensters fest.
`Höhe`	Optional. Legt die neue Höhe des Fensters fest.

Tabelle 70: Die Argumente der Methode MoveSize

Hinweis

Die Maßeinheit für die Argumente lautet Twips. (Twip: eine Maßeinheit, die 1/20 eines Punkts oder 1/1440 eines Zolls entspricht. 567 Twips sind ein Zentimeter.)

Im folgenden Beispiel aus Listing 175 wird das Formular KUNDEN beim Aufruf in der Größe angepasst. Dazu stellen Sie das Ereignis `Form_Open` ein.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Formular   Form_Kunden
'======================================================

Private Sub Form_Open(Cancel As Integer)
   DoCmd.MoveSize 100, 100, 8000, 6000
End Sub
```

Listing 175: Formulargröße anpassen

Zur Manipulation der Fenstergröße stehen noch weitere Methoden zur Verfügung: Mithilfe der Methode `Maximize` können Sie ein Fenster so weit vergrößern, dass es das ganze Access-Fenster ausfüllt. Diese Methode hat keine weiteren Argumente.

Über die Methode `Minimize` können Sie das aktive Fenster auf eine kleine Titelleiste unten im Microsoft Access-Fenster reduzieren. Diese Methode hat keine weiteren Argumente.

Im folgenden Beispiel aus Listing 176 wird ein Bericht aufgerufen und das Fenster maximiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub FensterMaximieren()
On Error GoTo fehler
   DoCmd.OpenReport "Rechnngen", acPreview
   DoCmd.Maximize
   Exit Sub


  fehler:
     MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 176: Fenster maximieren

Eine weitere Methode ist im gleichen Atemzug zu nennen: Sie können die Methode `Restore` verwenden, um die vorherige Größe eines maximierten oder minimierten Fensters wiederherzustellen. Diese Methode hat keine weiteren Argumente.

Im Beispiel aus Listing 177 wird folgendes Standardverhalten von Access deaktiviert. Wenn Sie ein beliebiges Formular öffnen, dieses dann maximieren und anschließend schließen, dann werden auch andere Formulare, die Sie danach öffnen, in der Maximalansicht angezeigt. Um dies zu verhindern, wenden Sie das `Close`-Ereignis des Formulars an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Formular   Form_Kunden
'=====================================================

Private Sub Form_Close()
   DoCmd.Restore
End Sub
```

Listing 177: Formulargröße beim Schließen wieder zurücksetzen

158 Datenbankinhalte transferieren

Mithilfe der Methode `TransferDatabase` können Sie Daten zwischen der aktuellen Datenbank und einer anderen Datenbank austauschen. Die Syntax dieser Methode lautet:

```
Ausdruck.TransferDatabase(Transfertyp, Datenbanktyp, Datenbankname, Objekttyp, Quelle,
Ziel, NurStruktur, AnmeldungSpeichern)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
Transfertyp	Optional. Legt den Transfertyp fest. Es stehen dabei folgende Typen zur Verfügung: acExport, acImport (Standard), acLink
Datenbanktyp	Optional. Gibt den Namen der Datenbanktypen an, die zum Importieren, Exportieren oder Verknüpfen von Daten verwendet werden können. Standardwert ist hier »Microsoft Access«. Weitere mögliche Typen können Sie der Online-Hilfe entnehmen.
Datenbankname	Optional. Gibt den vollständigen Namen und Pfad der Datenbank an, die zum Importieren, Exportieren oder Verknüpfen von Daten verwendet werden soll.
Objekttyp	Optional. Legt den Objekttyp fest.
Quelle	Optional. Gibt den Namen des Objekts an, dessen Daten importiert, exportiert oder verknüpft werden sollen.
Ziel	Optional. Gibt den Namen des importierten, exportierten oder verknüpften Objekts in der Zieldatenbank an.
NurStruktur	Optional. Verwenden Sie True, um nur die Struktur einer Datenbanktabelle zu importieren oder zu exportieren. Verwenden Sie False, um die Struktur der Tabelle sowie deren Daten zu importieren oder zu exportieren.
AnmeldungSpeichern	Optional. Verwenden Sie True, um in der Verbindungszeichenfolge einer verknüpften Tabelle den Benutzernamen (ID) und das Kennwort für eine ODBC-Datenbank zu speichern, zu der die Tabelle gehört. Auf diese Weise entfällt die Anmeldung beim Öffnen der Tabelle. Verwenden Sie False, wenn Sie den Benutzernamen und das Kennwort nicht speichern möchten.

Tabelle 71: Die Argumente der Methode TransferDatabase

Im folgenden Beispiel aus Listing 178 wird der Bericht RECHNUNG aus der Datenbank Nordwind.mdb in die aktuell geöffnete Datenbank transferiert und unter dem Namen RechnungNeu gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub BerichtTransferieren()
   On Error GoTo fehler
```

Listing 178: Einen Bericht in eine andere Datenbank transferieren

```
    DoCmd.TransferDatabase acImport, _
       "Microsoft Access", _
       "C:\Eigene Dateien\Nordwind.mdb", acReport, _
       "Rechnung", "RechnungNeu"
    Exit Sub


    fehler:
       MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 178: Einen Bericht in eine andere Datenbank transferieren (Forts.)

Im nächsten Beispiel aus Listing 179 wird aus der Datenbank `Nordwind.mdb` die Tabelle `Personal` in die aktuell geöffnete Datenbank verlinkt und mit dem Namen `PersonalGelinkt` gesichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleTransferierenUndLinken()
   On Error GoTo fehler
   DoCmd.TransferDatabase acLink, _
      "Microsoft Access", _
      "C:\Eigene Dateien\Nordwind.mdb", acTable, _
      "Personal", "PersonalGelinkt"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 179: Tabelle transferieren und linken

159 Tabellen transferieren

Mithilfe der Methode `TransferSpreadsheet` können Sie Daten aus einer Access-Tabelle beispielsweise in eine Excel-Tabelle übertragen. Die Syntax dieser Methode lautet:

```
Ausdruck.TransferSpreadsheet(Transfertyp, Dateiformat, Tabellenname, Dateiname,
BesitztFeldnamen, Bereich)
```

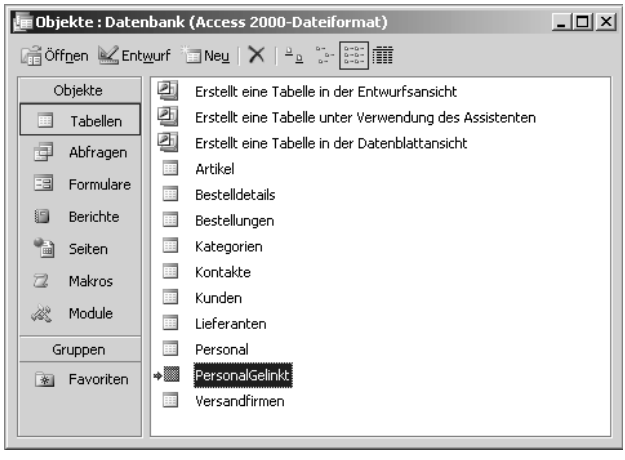


Abbildung 95: Die verlinkte Tabelle erkennen Sie am Pfeilsymbol.

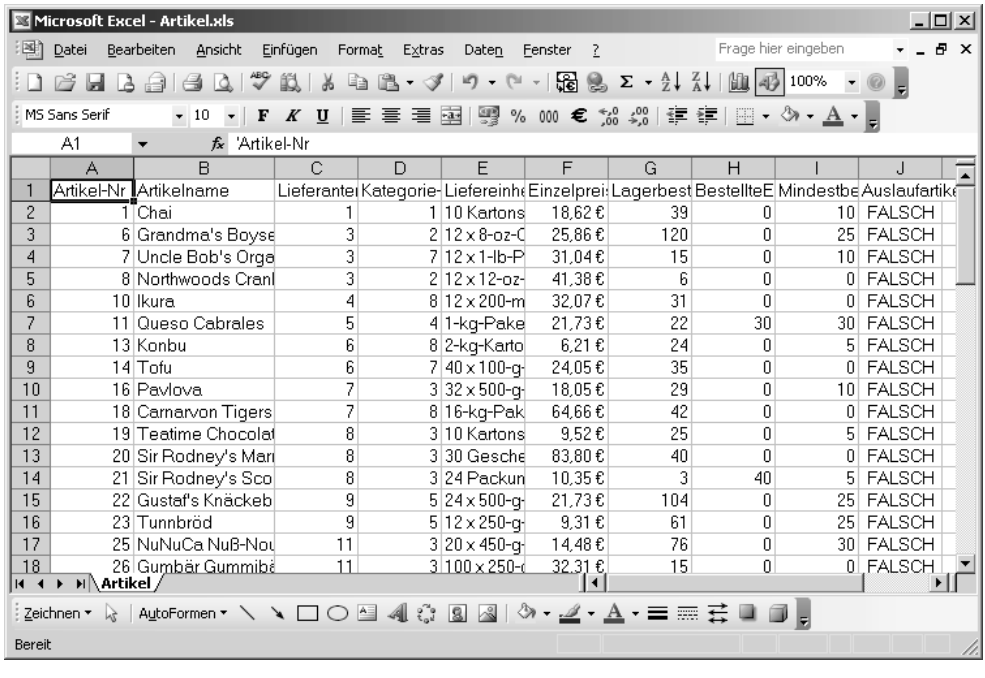
Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der ein `DoCmd`-Objekt zurückgibt.
`Transfertyp`	Optional. Gibt an, welchen Transfer Sie genau durchführen möchten. Sie haben dabei die Auswahl zwischen dem Export (`acExport`), dem Import (`acImport`) oder einer Verknüpfung (`acLink`).
`Dateiformat`	Optional. Gibt an, in welcher Excel-Version (bzw. Lotus-Version) Sie die Access-Tabelle exportieren möchten.
`Tabellenamen`	Optional. Gibt den Namen der zu exportierenden Access-Tabelle an.
`Dateinamen`	Optional. Gibt das Ziel für den Datenexport bekannt. Dabei muss die angegebene Datenquelle nicht einmal existieren. Access legt diese neu für Sie an.
`BesitztFeldnamen`	Optional. Verwenden Sie den Wert `True`, um die erste Zeile der Kalkulationstabelle beim Importieren, Exportieren oder Verknüpfen zur Angabe der Feldnamen zu verwenden. Verwenden Sie hingegen den Wert `False`, wenn die erste Zeile als normale Datenzeile gelten soll. Wenn Sie dieses Argument nicht angeben, wird der Standardwert `False` verwendet.
`Bereich`	Diese Argument gilt nur für Importoperationen und darf beim Export nicht angegeben werden. Beim Import werden standardmäßig alle Datensätze importiert, sofern dieses Argument nicht gesetzt wird.

Tabelle 72: Die Argumente der Methode TransferSpreadsheet

Im folgenden Beispiel aus Listing 180 wird die Access-Tabelle `Artikel` in eine Excel 2000-Tabelle überführt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleTransferieren()
   On Error GoTo fehler
   DoCmd.TransferSpreadsheet acExport, _
      acSpreadsheetTypeExcel9, "Artikel", _
      "C:\Eigene Dateien\Artikel.xls", True
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 180: Eine Datentabelle als Excel-Tabelle exportieren



	A	B	C	D	E	F	G	H	I	J
1	Artikel-Nr	Artikelname	Lieferanter	Kategorie-	Liefereinhe	Einzelprei	Lagerbest	BestellteE	Mindestbe	Auslaufartik
2	1	Chai	1	1	10 Kartons	18,62 €	39	0	10	FALSCH
3	6	Grandma's Boyse	3	2	12 x 8-oz-G	25,86 €	120	0	25	FALSCH
4	7	Uncle Bob's Orga	3	7	12 x 1-lb-P	31,04 €	15	0	10	FALSCH
5	8	Northwoods Cranl	3	2	12 x 12-oz-	41,38 €	6	0	0	FALSCH
6	10	Ikura	4	8	12 x 200-m	32,07 €	31	0	0	FALSCH
7	11	Queso Cabrales	5	4	1-kg-Pake	21,73 €	22	30	30	FALSCH
8	13	Konbu	6	8	2-kg-Karto	6,21 €	24	0	5	FALSCH
9	14	Tofu	6	7	40 x 100-g-	24,05 €	35	0	0	FALSCH
10	16	Pavlova	7	3	32 x 500-g-	18,05 €	29	0	10	FALSCH
11	18	Carnarvon Tigers	7	8	16-kg-Pak	64,66 €	42	0	0	FALSCH
12	19	Teatime Chocolat	8	3	10 Kartons	9,52 €	25	0	5	FALSCH
13	20	Sir Rodney's Mari	8	3	30 Gesche	83,80 €	40	0	0	FALSCH
14	21	Sir Rodney's Sco	8	3	24 Packun	10,35 €	3	40	5	FALSCH
15	22	Gustaf's Knäckeb	9	5	24 x 500-g-	21,73 €	104	0	25	FALSCH
16	23	Tunnbröd	9	5	12 x 250-g-	9,31 €	61	0	25	FALSCH
17	25	NuNuCa Nuß-Nou	11	3	20 x 450-g-	14,48 €	76	0	30	FALSCH
18	26	Gumbär Gummibä	11	3	100 x 250-g	32,31 €	15	0	0	FALSCH

Abbildung 96: Das Ergebnis liegt in einer Excel-Tabelle vor.

160 Texte exportieren und importieren

Mithilfe der `TransferText`-Methode können Sie Text zwischen der aktuellen Microsoft Access-Datenbank und einer Textdatei importieren oder exportieren. Die Syntax dieser Methode lautet:

```
Ausdruck.TransferText(Transfertyp, Spezifikationsname, Tabellenname, Dateiname,
BesitztFeldnamen, HTML-Tabellenname, Codepage)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der ein DoCmd-Objekt zurückgibt.
Transfertyp	Optional. Es stehen folgende Typen zur Verfügung: acExportDelim, acExportFixed, acExportHTML, acExportMerge, acImportDelim (Standard), acImportFixed, acImportHTML, acLinkDelim, acLinkFixed, acLinkHTML
Spezifikationsname	Optional. Ein Zeichenfolgenausdruck, der den Namen einer Import- oder Exportspezifikation angibt, die Sie in der aktuellen Datenbank erstellt und gespeichert haben. Bei einer Textdatei mit fester Zeilenlänge müssen Sie entweder ein Argument angeben oder eine Schemadatei (*Schema.ini*) verwenden, die in demselben Ordner wie die importierte, verknüpfte oder exportierte Textdatei gespeichert sein muss.
Tabellenname	Optional. Legt den Namen einer Microsoft Access-Tabelle zum Importieren, Exportieren oder Verknüpfen von Textdaten fest.
Dateiname	Optional. Legt den vollständigen Namen und den Pfad der Textdatei an, die zum Importieren, Exportieren oder Verknüpfen von Daten verwendet werden soll.
BesitztFeldnamen	Optional. Verwenden Sie True, um die erste Zeile der Textdatei beim Importieren, Exportieren oder Verknüpfen zur Angabe der Feldnamen zu verwenden. Verwenden Sie False, wenn die erste Zeile als normale Datenzeile gelten soll.
HTML-Tabellenname	Optional. Gibt den Namen der Tabelle oder Liste in der HTML-Datei an, die Sie importieren oder verknüpfen möchten. Dieses Argument wird nur dann beachtet, wenn das Argument Transfertyp auf acImportHTML oder acLinkHTML festgelegt ist.
Codepage	Optional. Ein Wert des Typs Long, der den Zeichensatz der Codepage angibt.

Tabelle 73: Die Argumente der Methode TransferText

Im folgenden Beispiel aus Listing 181 wird eine Tabelle als Textdatei exportiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================
```

Listing 181: Tabelle in Textdatei exportieren

```
Sub TextExportieren()
   On Error GoTo fehler
   DoCmd.TransferText acExportDelim, , _
      "Artikel", "C:\Eigene Dateien\Artikel.txt"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 181: Tabelle in Textdatei exportieren (Forts.)

Beim Import von Textdateien müssen Sie vorher eine Importspezifikation erstellen. Dazu sind folgende Schritte notwenig:

1. Wählen Sie aus dem Menü DATEI den Befehl EXTERNE DATEN/IMPORTIEREN.
2. Im Dialogfeld IMPORTIEREN wählen Sie die Textdatei aus, die Sie in eine Access-Tabelle importieren möchten.
3. Klicken Sie auf die Schaltfläche IMPORTIEREN.
4. Aktivieren Sie im Textimport-Assistenten die gewünschte Option. Da bei der Textdatei *Artikel.txt* keine feste Feldbreite vorliegt und Sie das Semikolon als Trennzeichen vorfinden, aktivieren Sie in diesem Beispiel die erste Option.
5. Klicken Sie danach auf die Schaltfläche WEITERE (bitte nicht verwechseln mit WEITER!).
6. Im Gruppenfeld FELDINFORMATION hat Access anhand der Textdatei *Artikel.txt* bereits die Felder vom Datentyp her gesehen für Sie automatisch vordefiniert. Diese Information können Sie natürlich noch anpassen, wenn es nötig ist. In der Spalte ÜBERSPRINGEN können Sie einzelne Datenfelder der Textdatei überspringen, d.h., wenn Sie einzelne Felder dort aktivieren, werden diese nicht mit in die Tabelle übernommen.
7. Klicken Sie auf die Schaltfläche SPEICHERN, um die Importspezifikation zu sichern.
8. Geben Sie der Importspezifikation einen Namen und bestätigen Sie mit OK. Merken Sie sich diesen Namen, den Sie später für Ihr Makro benötigen.
9. Klicken Sie danach auf die Schaltfläche OK, um das Dialogfeld ARTIKEL IMPORTSPEZIFIKATIONEN zu schließen.
10. Im Textimport-Assistenten wieder angekommen, klicken Sie auf WEITER, um zum nächsten Importschritt zu gelangen.
11. Übergehen Sie auch den nächsten Schritt mit einem Klick auf die Schaltfläche WEITER.
12. Um die Textdatei in einer neuen Tabelle zu speichern, aktivieren Sie die Option IN EINER NEUEN TABELLE.
13. Klicken Sie danach auf WEITER.

14. In diesem Schritt können Sie noch einmal festlegen, wie die einzelnen Felder definiert werden sollen. Da Sie diese Aufgabe aber bereits vorher über die Importspezifikation erledigt haben, klicken Sie auf die Schaltfläche WEITER.
15. Im nächsten Schritt des Assistenten können Sie festlegen, ob Sie einen Primärschlüssel anlegen möchten. Dabei können Sie diese Aufgabe von Access selbst ausführen lassen. Klicken Sie danach auf WEITER.
16. Geben Sie an, in welche Tabelle Sie die Textdatei einfügen möchten. Sollte diese Tabelle noch nicht existieren, wird diese von Access automatisch angelegt.
17. Klicken Sie abschließend auf die Schaltfläche FERTIG STELLEN.

Jetzt können Sie das Makro für den Import einer Textdatei schreiben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TextImportieren()
   On Error GoTo fehler
   DoCmd.TransferText acImportDelim, _
      "ArtikelImportspezifikation", _
      "ArtikelX", _
      "C:\Eigene Dateien\Artikel.txt", False
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 182: Textdatei importieren

161 Das Reference-Objekt

Das Objekt `Reference` stellt einen Verweis auf die Typbibliothek einer anderen Anwendung oder eines anderen Projekts dar.

162 Bibliotheken auslesen

Über die `References`-Auflistung können Sie alle eingebundenen Bibliotheken in der Entwicklungsumgebung abarbeiten.

Im folgenden Beispiel aus Listing 183 werden alle eingebundenen Bibliotheken der Entwicklungsumgebung im Direktfenster aufgelistet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlRef
'=====================================================

Sub BibliothekenDokumentieren()
   Dim intZ As Integer

   For intZ = 1 To References.Count
      Debug.Print References(intZ).FullPath
   Next intZ
End Sub
```

Listing 183: Eingebundene Bibliotheken auslesen

In einer Schleife arbeiten Sie alle Bibliotheken ab. Innerhalb der Schleife ermitteln Sie den Namen sowie den kompletten Pfad der Bibliothek, indem Sie die Eigenschaft `FullName` einsetzen.

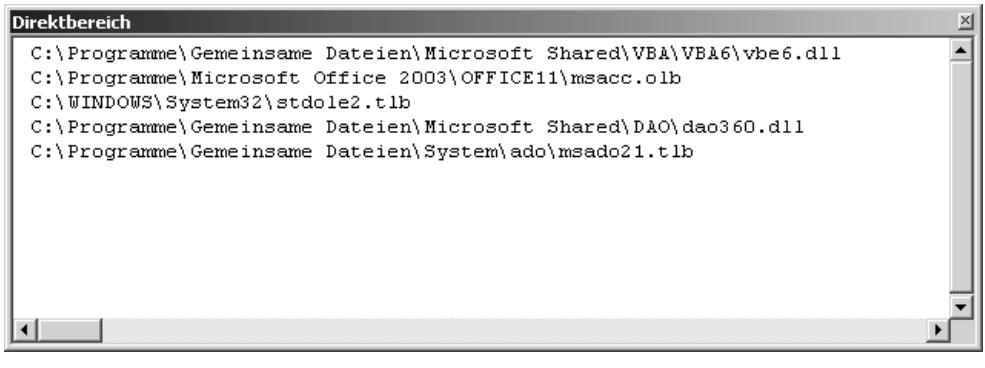


Abbildung 97: Alle aktiven Bibliotheken auslesen

Hinweis

In Kapitel 9 können Sie nachschlagen, wie Sie mithilfe der VBE-Programmierung noch weiter gehende Aufgaben schnell erledigen können.

163 Office-Assistenten einsetzen

Der Office-Assistent wird im Makro aus Listing 184 dafür eingesetzt, um den Ablauf einer VBA-Schulung darzustellen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlBal
'=====================================================

Sub OfficeAssistentenAufrufen()
   Dim OffAss As Balloon

   Set OffAss = Assistant.NewBalloon

   With OffAss
      .Heading = "Office-Assistent"
      .Icon = msoIconTip
      .Mode = msoModeAutoDown
      .BalloonType = msoBalloonTypeButtons
      .Labels(1).Text = "Tag 1: VBA-Grundlagen"
      .Labels(2).Text = "Tag 2: Einfache Programmiertechnken"
      .Labels(3).Text = "Tag 3: Komplexere Programmiertechniken"
      .Animation = msoAnimationGreeting
      .Button = msoButtonSetOK
      .Show
   End With
   Set OffAss = Nothing
End Sub
```

Listing 184: Der Aufruf des Office-Assistenten

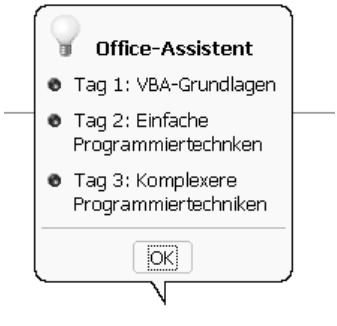


Abbildung 98: Der Office-Assistent wird aufgerufen.

Im Makro aus Listing 184 werden einige Methoden und Eigenschaften eingesetzt, die jetzt erläutert werden sollen.

Über die Eigenschaft `Heading` können Sie den Titel in einem `Balloon`-Objekt festlegen.

Die Eigenschaft `Icon` gibt den Symboltyp zurück, der im unteren linken Abschnitt der Sprechblase des Office-Assistenten erscheint. Dabei stehen folgende Konstanten aus Tabelle 74 zur Verfügung:

Konstante	Symbol
`msoIconAlert`	Gelbes Ausrufezeichen
`msoIconAlertCritical`	Rotes X-Symbol
`msoIconAlertInfo`	Info
`msoIconAlertQuery`	Fragezeichen
`msoIconAlertWarning`	Gelbes Ausrufezeichen
`msoIconNone`	Kein Symbol
`msoIconTip`	Glühbirne

Tabelle 74: Die Konstanten für das Symbol eines Ballons

Die Eigenschaft `Mode` gibt das modale Verhalten der Sprechblase des Office-Assistenten zurück oder legt es fest. Dabei stehen die folgenden Konstanten aus Tabelle 75 zur Verfügung.

Konstante	Beschreibung
`msoModeAutoDown`	Die Sprechblase wird geschlossen, sobald der Benutzer auf eine beliebige Stelle des Bildschirms klickt.
`msoModeModal`	Der Anwender muss die Sprechblase schließen, bevor er die Arbeit in der Anwendung fortsetzen kann.
`msoModeModeless`	Der Anwender kann in der Anwendung weiterarbeiten, während die Sprechblase sichtbar ist.

Tabelle 75: Die Konstanten der Eigenschaft Mode

Die Eigenschaft `BalloonType` gibt den Sprechblasentyp zurück, den der Office-Assistent verwendet, oder legt ihn fest. Dabei werden die Aufzählungszeichen festgelegt. Sie können folgende Konstanten aus der Tabelle 76 einsetzen.

Konstante	Aufzählungszeichen
`msoBalloonTypeBullets`	Kleine Punkte
`msoBalloonTypeButtons`	Größere Punkte
`msoBalloonTypeNumbers`	Numerische

Tabelle 76: Die Konstanten der Eigenschaft BalloonType

Die Eigenschaft `Labels` legt die Beschriftungen für die Schaltflächen, Nummerierungen und Aufzählungen in einer Sprechblase des Office-Assistenten fest.

Die Eigenschaft `Animation` gibt die Animationsaktion des Office-Assistenten zurück oder legt sie fest. Insgesamt stehen eine ganze Reihe von Animationen zur Verfügung. Je nach eingestelltem Assistenten wird die eine oder andere Animation nicht angezeigt.

Konstante
msoAnimationAppear
msoAnimationBeginSpeaking
msoAnimationCharacterSuccessMajor
msoAnimationCheckingSomething
msoAnimationDisappear
msoAnimationEmptyTrash
msoAnimationGestureDown
msoAnimationGestureLeft
msoAnimationGestureRight
msoAnimationGestureUp
msoAnimationGetArtsy
msoAnimationGetAttentionMajor
msoAnimationGetAttentionMinor
msoAnimationGetTechy
msoAnimationGetWizardy
msoAnimationGoodbye
msoAnimationGreeting
msoAnimationIdle
msoAnimationListensToComputer
msoAnimationLookDown
msoAnimationLookDownLeft
msoAnimationLookDownRight
msoAnimationLookLeft
msoAnimationLookRight
msoAnimationLookUp
msoAnimationLookUpLeft
msoAnimationLookUpRight
msoAnimationPrinting
msoAnimationRestPose
msoAnimationSaving
msoAnimationSearching
msoAnimationSendingMail
msoAnimationThinking

Tabelle 77: Die Konstanten für die Eigenschaft Animation

Konstante
msoAnimationWorkingAtSomething
msoAnimationWritingNotingSomething

Tabelle 77: Die Konstanten für die Eigenschaft Animation (Forts.)

Die Eigenschaft `Button` gibt die Art der unten in der Sprechblase des Office-Assistenten angezeigten Schaltfläche zurück oder legt sie fest. Dabei können Sie folgende Konstanten aus der Tabelle 78 verwenden.

Argument	Schaltflächen
msoButtonSetAbortRetryIgnore	Abbrechen, Ignorieren und Wiederholen
msoButtonSetBackClose	Zurück und Schliessen
msoButtonSetBackNextClose	Zurück, Weiter, Schliessen
msoButtonSetBackNextSnooze	Zurück, Weiter, Erneut erinnern
msoButtonSetCancel	Abbrechen
msoButtonSetNextClose	Weiter, Schliessen
msoButtonSetNone	Keine Schaltfläche
msoButtonSetOK	OK
msoButtonSetOkCancel	OK und Abbrechen
msoButtonSetRetryCancel	Wiederholen, Abbrechen
msoButtonSetSearchClose	Suchen, Schliessen
msoButtonSetTipsOptionsClose	Tipps, Optionen, Schliessen
msoButtonSetYesAllNoCancel	Ja, Ja alle, Abbrechen, Schliessen
msoButtonSetYesNo	Ja, Nein
msoButtonSetYesNoCancel	Ja, Nein, Abbrechen

Tabelle 78: Die Konstanten für die Schaltflächen eines Ballons

Die Methode `Show` zeigt das angegebene `Balloon`-Objekt an.

Die Frage, welche Schaltfläche im Office-Assistenten angeklickt wurde, beantwortet die Tabelle 79.

Konstante	Schaltfläche
msoBalloonButtonAbort	Abbrechen
msoBalloonButtonBack	Zurück
msoBalloonButtonCancel	Abbrechen
msoBalloonButtonClose	Schliessen

Tabelle 79: Die Rückgabekonstanten der Methode Show

Konstante	Schaltfläche
msoBalloonButtonIgnore	Ignorieren
msoBalloonButtonNext	Nächste
msoBalloonButtonNo	Nein
msoBalloonButtonNull	Null
msoBalloonButtonOK	OK
msoBalloonButtonOptions	Optionen
msoBalloonButtonRetry	Wiederholen
msoBalloonButtonSearch	Suchen
msoBalloonButtonSnooze	Erneut erinnern
msoBalloonButtonTips	Tipps
msoBalloonButtonYes	Ja
msoBalloonButtonYesToAll	Alle Ja

Tabelle 79: Die Rückgabekonstanten der Methode Show (Forts.)

Beim folgenden Beispiel aus Listing 185 soll ermittelt werden, welche der Schaltflächen vom Anwender angeklickt wurde.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlBal
'=====================================================

Sub Entscheidung()
   Dim OffAss As Balloon

   Set OffAss = Assistant.NewBalloon
   With OffAss
      .Heading = "Wählen Sie eine Variante aus"
      .Labels(1).Text = "VarianteA"
      .Labels(2).Text = "VarianteB"
      .Button = msoButtonSetOkCancel

      If .Show = msoBalloonButtonOK Then
         MsgBox "Schaltfläche OK geklickt!"
      Else
         MsgBox "Schaltfläche Abbrechen geklickt!"
      End If
   End With
End Sub
```

Listing 185: Schaltfläche beim Office-Assistenten auswerten

Abbildung 99: Die Auswertung der Schaltfläche findet direkt nach dem Klick statt.

164 Datei kopieren

Mithilfe der Methode `CopyFile` können Sie eine oder mehrere Dateien von einem Ort an einen anderen kopieren. Die Syntax dieser Methode lautet:

```
Objekt.CopyFile Quelle, Ziel[, überschreiben]
```

Teil	Beschreibung
`Objekt`	Erforderlich. Das Objekt ist immer der Name eines `FileSystemObject`.
`Quelle`	Erforderlich. Zeichenfolge für die Dateispezifikation für eine oder mehrere zu kopierende Dateien, die Platzhalterzeichen enthalten kann.
`Ziel`	Erforderlich. Zeichenfolge für das Ziel, an das die Datei(en) aus `Quelle` kopiert werden soll(en). Platzhalterzeichen sind nicht zulässig.
`überschreiben`	Optional. Boolescher Wert, der angibt, ob vorhandene Dateien überschrieben werden sollen. Wenn er `True` ist, werden die Dateien überschrieben; ist er `False`, werden sie nicht überschrieben. Die Voreinstellung ist `True`. Beachten Sie, dass `CopyFile` unabhängig vom Wert für `überschreiben` fehlschlägt, wenn für `Ziel` das schreibgeschützte Attribut festgelegt wurde.

Tabelle 80: Die Argumente der Methode CopyFile

Im folgenden Beispiel aus Listing 186 wird die Excel-Arbeitsmappe *Katalog.xls* aus dem Route-Verzeichnis der Festplatte in das Verzeichnis *C:\Eigene Dateien* kopiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================
```

Listing 186: Einzelne Datei kopieren

```
Sub DateiKopieren()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.CopyFile "C:\Katalog.xls", "C:\Eigene Dateien\"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 186: Einzelne Datei kopieren (Forts.)

Sollen gleich mehrere Dateien, beispielsweise alle Excel-Mappen, von einem Verzeichnis in ein anderes kopiert werden, dann starten Sie das Makro aus Listing 187.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DateienKopieren()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.CopyFile "C:\*.xls", "C:\Eigene Dateien\"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 187: Mehrere Dateien kopieren

165 Datei verschieben

Mithilfe der Methode `MoveFile` können Sie eine oder mehrere Dateien von einem Ort an einen anderen verschieben. Die Syntax dieser Funktion lautet:

```
Objekt.MoveFile Quelle, Ziel
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject`.
`Quelle`	Erforderlich. Der Pfad der zu verschiebenden Datei(en). Die Zeichenfolge des Arguments `Quelle` kann nur in der letzten Pfadkomponente Platzhalterzeichen enthalten.
`Ziel`	Erforderlich. Der Pfad zu dem Ort, an den die Datei(en) verschoben werden soll(en). Das Argument `Ziel` kann keine Platzhalterzeichen enthalten.

Tabelle 81: Die Argumente der Methode MoveFile

Im Makro aus Listing 188 wird die Datei *Katalog.xls* von einem Verzeichnis in ein anderes verschoben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DateiVerschieben()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.MoveFile "C:\Katalog.xls", "C:\Eigene Dateien\"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 188: Einzelne Datei verschieben

Auch hier können wieder mehrere Dateien von einem Verzeichnis in ein anderes verschoben werden, wie Listing 189 zeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================
```

Listing 189: Mehrere Dateien verschieben

```
Sub DateienVerschieben()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.MoveFile "C:\*.xls", "C:\Eigene Dateien\"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 189: Mehrere Dateien verschieben (Forts.)

166 Datei löschen

Über den Einsatz der Methode `DeleteFile` können Sie eine angegebene Datei löschen. Die Syntax dieser Methode lautet:

```
Objekt.DeleteFile Dateispez[, erzwingen]
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject`.
`Dateispez`	Erforderlich. Der Name der zu löschenden Datei. Die `Dateispez` kann in der letzten Pfadkomponente Platzhalterzeichen enthalten.
`Erzwingen`	Optional. Boolescher Wert, der `True` ist, wenn Dateien mit Schreibschutzattribut gelöscht werden sollen, bzw. `False` (Voreinstellung), wenn sie nicht gelöscht werden sollen.

Tabelle 82: Die Argumente der Methode DeleteFile

Im Makro aus Listing 190 wird eine Datei ohne Rückfrage von der Festplatte entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DateiEntfernen()
   Dim fs As Object
   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.DeleteFile "C:\Eigene Dateien\Katalog.xls"
   Exit Sub
```

Listing 190: Einzelne Datei löschen

```
   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 190: Einzelne Datei löschen (Forts.)

Im folgenden Beispiel aus Listing 191 werden alle Dateien aus dem temporären Verzeichnis *C:\Windows\Temp* gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DateienEntfernen()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.DeleteFile "C:\Windows\Temp\*.*"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 191: Alle Dateien in einem Verzeichnis löschen

167 Dateiexistenz prüfen

Mithilfe der Methode `FileExists` können Sie überprüfen, ob eine bestimmte Datei in einem Verzeichnis überhaupt existiert.

Im Beispiel aus Listing 192 wird überprüft, ob die Datei *Katalog.xls* im Verzeichnis *C:\Eigene Dateien* enthalten ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DateiDa()
   Dim fs As Object

   Set fs = CreateObject("Scripting.FileSystemObject")
```

Listing 192: Dateiexistenz überprüfen

```
    If fs.FileExists("C:\Eigene Dateien\Katalog.xls") = True Then
       MsgBox "Datei existiert!"
    Else
       MsgBox "Datei existiert nicht!"
    End If
End Sub
```

Listing 192: Dateiexistenz überprüfen (Forts.)

Die Methode `FileExists` meldet den Wert `True` zurück, wenn die Datei im angegebenen Verzeichnis gefunden werden kann.

168 Dateiinfo abfragen

Im folgenden Beispiel aus Listing 193 ermitteln Sie mithilfe des `FileSystem`-Objekts die Dateieigenschaften Erstellungsdatum, letzte Änderung und letzter Zugriff auf die Datei. Dabei kommen die Eigenschaften `DateCreated`, `DateLastModified` und `DateLastAccessed` zum Einsatz.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub DatenbankInfosErmitteln()
   Dim fs As Object
   Dim f As Object
   Dim strDatei As String

   strDatei = "C:\Eigene Dateien\objekte.mdb"
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFile(strDatei)

   Debug.Print "Erstellt       : " & f.DateCreated
   Debug.Print "Letzter Zugriff: " & f.DateLastAccessed
   Debug.Print "Letzte Änderung: " & f.DateLastModified
End Sub
```

Listing 193: Dateiinformationen abfragen

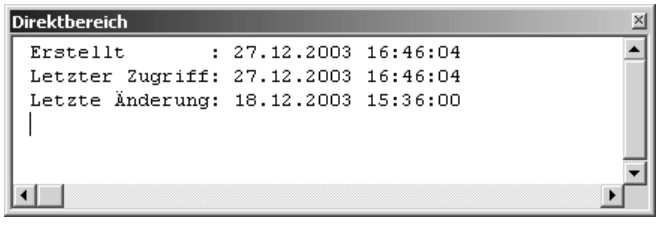


Abbildung 100: Die wichtigsten Informationen zur Datei

169 Verzeichnis anlegen

Über die Methode `CreateFolder` können Sie ein neues Verzeichnis anlegen. Im Makro aus Listing 194 wird der neue Ordner *Kopie* angelegt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=================================================

Sub OrdnerErstellen()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.CreateFolder "C:\Eigene Dateien\Kopie"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 194: Neuen Ordner anlegen

170 Verzeichnis löschen

Die Methode `DeleteFolder` löscht einen angegebenen Ordner und seinen Inhalt. Die Syntax dieser Methode lautet:

```
Objekt.DeleteFolder Ordnerspez[, erzwingen]
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject`.
`Ordnerspez`	Erforderlich. Der Name des zu löschenden Ordners. Die `Ordnerspez` kann in der letzten Pfadkomponente Platzhalterzeichen enthalten.
`Erzwingen`	Optional. Boolescher Wert, der `True` ist, wenn Ordner mit Schreibschutzattribut gelöscht werden sollen, bzw. `False` (Voreinstellung), wenn sie nicht gelöscht werden sollen

Tabelle 83: Die Argumente der Methode DeleteFolder

Im folgenden Beispiel aus Listing 195 wird der Ordner *Kopie* ohne Rückfrage entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub OrdnerEntfernen()
   Dim fs As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   fs.DeleteFolder "C:\Eigene Dateien\Kopie"
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 195: Ordner entfernen

171 Ordnerexistenz prüfen

Über den Einsatz der Methode `FolderExists` können Sie testen, ob ein Verzeichnis existiert. Die Methode gibt den Wert `True` zurück, wenn ein angegebener Ordner vorhanden ist, bzw. den Wert `False`, wenn dies nicht der Fall ist. Die Syntax dieser Methode lautet:

```
Objekt.FolderExists(Ordnerspez)
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject`.
`Ordnerspez`	Erforderlich. Der Name des Ordners, von dem Sie wissen möchten, ob er existiert. Sie müssen eine (entweder absolute oder relative) vollständige Pfadangabe machen, wenn Sie davon ausgehen, dass der Ordner nicht im aktuellen Ordner vorhanden ist.

Tabelle 84: Die Argumente der Methode FolderExists

Im Makro aus Listing 196 wird überprüft, ob der Ordner *C:\Eigene Dateien\Kopie* existiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================
```

Listing 196: Ordnerexistenz überprüfen

```
Sub OrdnerDa()
   Dim fs As Object

   Set fs = CreateObject("Scripting.FileSystemObject")
   If fs.FolderExists("C:\Eigene Dateien\Kopie") = True Then
      MsgBox "Ordner existiert!"
   Else
      MsgBox "Ordner existiert nicht!"
   End If
End Sub
```

Listing 196: Ordnerexistenz überprüfen (Forts.)

172 Ordnergröße ermitteln

Soll die Größe eines Ordners ermittelt werden, um beispielsweise festzustellen, ob der Inhalt des Ordners auf eine Diskette bzw. einen Datenstick passt, dann setzen Sie die Eigenschaft `Size` ein.

Im Beispiel aus Listing 197 wird die Größe des Ordners *C:\Eigene Dateien* in Mbyte ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub OrdnerGrößeErmitteln()
   Dim fs As Object
   Dim fsOrdner As Object

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set fsOrdner = fs.GetFolder("C:\Eigene Dateien\Kopie")

   MsgBox "Der Ordner hat eine Größe von " & _
   Round(fsOrdner.Size / 1024 / 1024, 2) & " MByte", vbInformation
   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 197: Die Ordnergröße ermitteln

Die Größenangabe wird in Byte ausgegeben. Um daraus Kilobyte zu machen, teilen Sie das Ergebnis durch 1024. Sollen daraus Mbyte gemacht werden, dann teilen Sie wiederum durch 1024. Runden Sie das Ergebnis auf zwei Stellen hinter dem Komma, indem Sie die Funktion `Round` einsetzen.

173 Ordner listen

Wenn Sie mit Verzeichnissen arbeiten, dann sollten Sie auch eine Makrolösung haben, die überprüft, ob in einem Verzeichnis noch weitere Unterverzeichnisse existieren. Diese Aufgabe können Sie beispielsweise über eine Funktion erledigen. Übergeben Sie dieser Funktion den Namen des Verzeichnisses, das Sie auslesen möchten und schreiben Sie alle Namen der Unterverzeichnisse in den Direktbereich der Entwicklungsumgebung von Access. Sehen Sie sich dazu das Listing 198 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Function OrdnerDurchsuchen(strV As String) As Boolean
   Dim fs As Object
   Dim objOrdner As Object
   Dim Ordner As Object
   Dim intz As Integer

   intz = 0
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set objOrdner = fs.GetFolder(strV)

   For Each Ordner In objOrdner.SubFolders
      Debug.Print "Ordner " & intz & ": " & Ordner.Name
      intz = intz + 1
   Next

   Set objOrdner = Nothing
   Set objFSO = Nothing
End Function


Sub VerzeichnisStrukturAuslesen()
   Dim b As Boolean

   b = OrdnerDurchsuchen("C:\windows")
End Sub
```

Listing 198: Unterordner aus einem Verzeichnis auflisten lassen

Über das Makro `VerzeichnisstrukturAuslesen` wird die Funktion `OrdnerDuchsuchen` aufgerufen. Dabei wird das gewünschte Verzeichnis an die Funktion übergeben. Innerhalb der Funktion wird eine Schleife durchlaufen, in der jedes Unterverzeichnis ermittelt wird. Mit der Anweisung `Debug.Print` wird der Name des jeweiligen Unterverzeichnisses in den Direktbereich der Entwicklungsumgebung geschrieben.

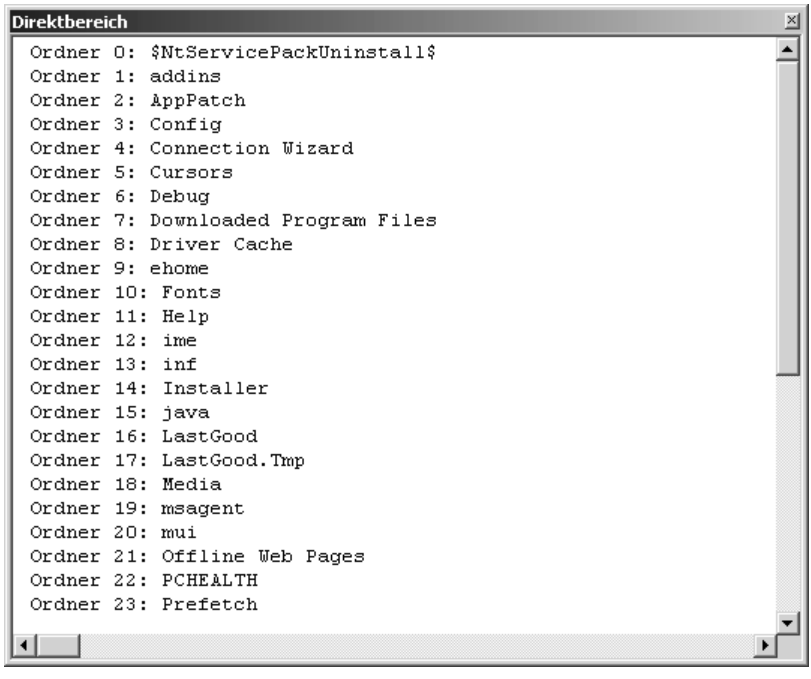


Abbildung 101: Alle Windows-Unterordner werden aufgelistet.

174 Dateien auflisten

Möchten Sie alle Dateien eines Verzeichnisses ermitteln, dann erfassen Sie das Makro aus Listing 199.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Function OrdnerDurchsuchen2(strV As String) As Boolean
   Dim fs As Object
   Dim objOrdner As Object
   Dim Datei As Object
   Dim intz As Integer

   intz = 1
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set objOrdner = fs.GetFolder(strV)

   For Each Datei In objOrdner.Files
      Debug.Print "Datei " & intz & ": " & Datei.Name
```

Listing 199: Alle Dateien aus einem Ordner auflisten

```
        intz = intz + 1
    Next
    Set objOrdner = Nothing
    Set objFSO = Nothing
End Function


Sub DateienListen()
    Dim b As Boolean

    b = OrdnerDurchsuchen2("C:\windows")
End Sub
```

Listing 199: Alle Dateien aus einem Ordner auflisten (Forts.)

Über das Makro `DateienListen` wird die Funktion `OrdnerDurchsuchen2` aufgerufen. Dabei wird das gewünschte Verzeichnis an die Funktion übergeben. Innerhalb der Funktion wird eine Schleife durchlaufen, in der jede Datei ermittelt wird. Mit der Anweisung `Debug.Print` wird der Name des jeweiligen Unterverzeichnisses in den Direktbereich der Entwicklungsumgebung geschrieben.

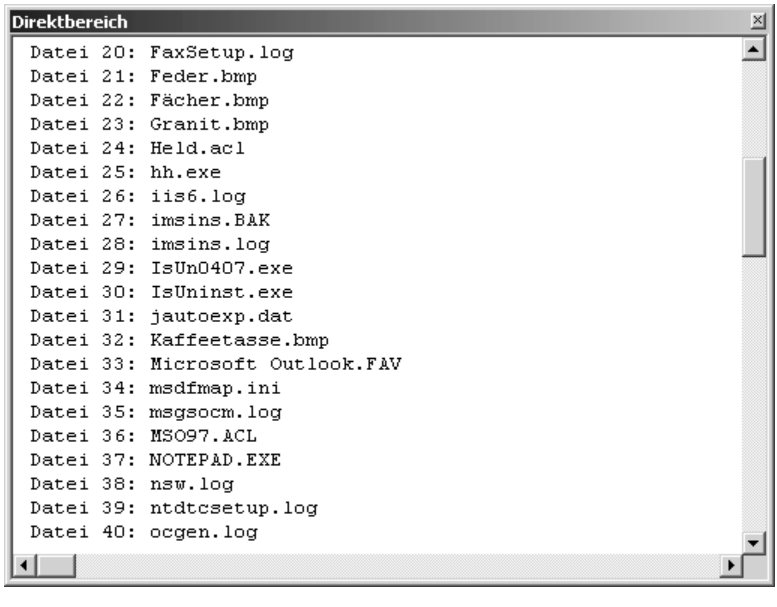


Abbildung 102: Alle Dateien werden aufgelistet.

175 Laufwerke auslesen

Neben einer API-Funktion, die Sie im vorherigen Kapitel nachschlagen können, gibt es auch die Möglichkeit, die Laufwerkstypen über das Objekt `FileSystemObject` zu ermitteln.

Im Makro aus Listing 200 werden alle Laufwerke Ihres Computers ausgelesen und deren Laufwerkstypen festgestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub LaufwerkeAuslesen()
   Dim fs As Object
   Dim fsObj As Object
   Dim strText As String

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")

   For Each fsObj In fs.Drives
      Select Case fsObj.DriveType
         Case 1
            strText = "Disketten-Laufwerk/USB-Datenstick"
         Case 2
            strText = "Festplatte"
         Case 3
            strText = "Netzlaufwerk"
         Case 4
            strText = "CDROM-Laufwerk"
         Case 5
            strText = "RAM-Disk"
         Case Else
            strText = "nicht ermittelbar"
      End Select
      Debug.Print fsObj.DriveLetter & " ist ein(e) " & strText
   Next fsObj

   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 200: Laufwerkstypen ermitteln und ausgeben

Mithilfe der Auflistung `Drives`, in der alle Laufwerke Ihres PCs automatisch verzeichnet sind, können Sie mittels einer `For each next`-Schleife den Laufwerkstyp des jeweiligen Laufwerks über die Eigenschaft `DriveType` ermitteln. Dabei gelten folgende Zuordnungen.

Nummer	Beschreibung
1	Diskette, USB-Datenstick oder etwas ähnlich Auswechselbares
2	Festplatte (fest eingebaut)
3	Netzlaufwerk
4	CD-ROM
5	RAM-Disk

Tabelle 85: Die DriveType-Eigenschaften im Überblick

Mithilfe der Eigenschaft `DriveLetter` können Sie den Laufwerksbuchstaben abfragen. Geben Sie diese beiden Informationen über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

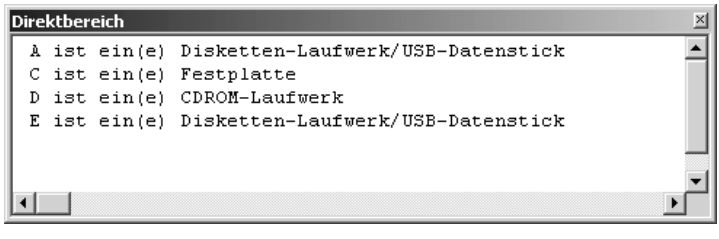


Abbildung 103: Die Laufwerkstypen wurden ermittelt.

176 Weitere Laufwerkseigenschaften

Im Makro aus Listing 201 können Sie weitere Laufwerkseigenschaften wie beispielsweise der verfügbare bzw. der Gesamtspeicherplatz Ihrer Festplatte ermitteln.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub LaufwerksEigenschaften()
   Dim fs As Object
   Dim fsObj As Object
   Dim strText As String

   On Error GoTo fehler
   Set fs = CreateObject("Scripting.FileSystemObject")

   For Each fsObj In fs.Drives
      With fsObj
```

Listing 201: Weitere Laufwerkseigenschaften abfragen

```
         Select Case .DriveType
            Case 1
               strText = "Disketten-Laufwerk/USB-Datenstick"
            Case 2
               strText = "Festplatte"
            Case 3
               strText = "Netzlaufwerk"
            Case 4
               strText = "CDROM-Laufwerk"
            Case 5
               strText = "RAM-Disk"
            Case Else
               strText = "nicht ermittelbar"
         End Select

         If .DriveType = 2 Then
            Debug.Print "Laufwerksbuchstabe:   " & _
               .DriveLetter & " ----> " & strText
            Debug.Print "Laufwerksname:        " & .VolumeName
            Debug.Print "Verfügbarer Speicher: " & _
            Round(.TotalSize / 1048576, 2) & " MByte"
            Debug.Print "Freier Speicher:      " & _
               Round(.FreeSpace / 1048576, 2) & " MByte"
            Debug.Print "Stammverzeichnis:     " & .RootFolder
            Debug.Print "Dateisystem:          " & .FileSystem
         End If
      End With
   Next fsObj

   Exit Sub


   fehler:
      MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 201: Weitere Laufwerkseigenschaften abfragen (Forts.)

Den Namen des Laufwerks, sofern einer vergeben wurde, können Sie über die Eigenschaft `VolumeName` abfragen. Die Gesamtkapazität der Festplatte ermitteln Sie mithilfe der Eigenschaft `TotalSize`. Den momentan freien Speicher liefert die Eigenschaft `FreeSpace`. Beide Angaben werde in Byte ausgegeben. Diese können Sie in Mbyte umrechnen, indem Sie das Ergebnis durch 1048576 teilen. Da das Ergebnis noch diverse Nachkommastellen aufweist, runden Sie dieses auf zwei Stellen hinter dem Komma mithilfe der Funktion `Round`. Das Stammverzeichnis Ihrer Festplatte können Sie über die Eigenschaft `RootFolder` abfragen. Die Info über das Dateisystem liefert Ihnen die Eigenschaft `FileSystem`.

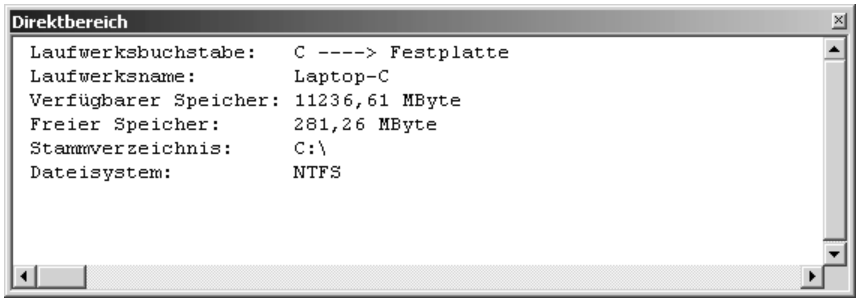


Abbildung 104: Die wichtigsten Eigenschaften der Festplatte

177 Textdatei einlesen

Über das `FileSystemObject` lassen sich auch Textdateien einlesen. Im folgenden Beispiel aus Listing 202 wird die Textdatei *Artikel.txt* in das Direktfenster der Entwicklungsumgebung Zeile für Zeile eingelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Sub TextdateiEinlesen()
   Dim fs As Object
   Dim objts As Object
   Dim strSatz As String

   Set fs = CreateObject("Scripting.FileSystemObject")
   Set objts = fs.OpenTextFile("C:\Eigene Dateien\Artikel.txt")

   Do Until objts.AtEndOfStream
      strSatz = objts.ReadLine
      Debug.Print strSatz
   Loop
   objts.Close
End Sub
```

Listing 202: Textdatei in das Direktfenster einlesen

Mithilfe der Methode `OpenTextFile` öffnen Sie die Textdatei. Die so geöffnete Textdatei steht Ihnen danach in der Objektvariablen `objts` zur Verfügung. In einer Schleife lesen Sie nun zeilenweise die `Textstream`-Variable aus. Übertragen Sie dabei den Inhalt eines Satzes zunächst in das Direktfenster der Entwicklungsumgebung. Über die Eigenschaft `AtEndOfStream` können Sie dabei abfragen, wann das Ende der eingelesenen Textdatei erreicht ist. Über die Methode `Close` schließen Sie die noch geöffnete Textdatei.

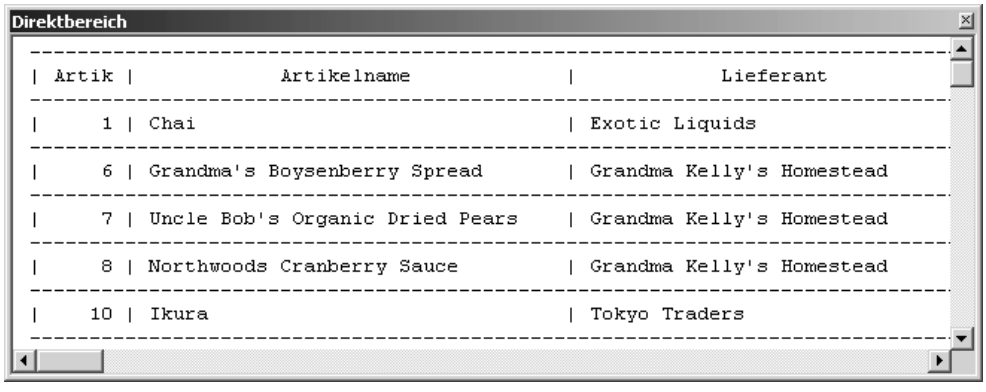


Abbildung 105: Die Textdatei wurde komplett eingelesen.

178 Textdatei schreiben

Mithilfe der Methode `CreateTextfile` können Sie eine Textdatei anlegen und über die Methode `WriteLine` erfassen Sie einzelne Zeilen. Die Syntax der Methode `CreateFile` sieht wie folgt aus:

```
Objekt.CreateTextFile(Dateiname[, überschreiben[, Unicode]])
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject` oder `Folder`-Objekts.
`Dateiname`	Erforderlich. Zeichenfolge, die die zu erstellende Datei identifiziert.
`Überschreiben`	Optional. Boolescher Wert, der angibt, ob eine vorhandene Datei überschrieben werden kann. Wenn er `True` ist, kann die Datei überschrieben werden; ist er `False`, kann sie nicht überschrieben werden. Wenn er ausgelassen wird, werden vorhandene Dateien nicht überschrieben.
`Unicode`	Optional. Boolescher Wert, der angibt, ob die Datei als Unicode- oder ASCII-Datei erstellt wird. Der Wert ist `True`, wenn die Datei als Unicode-Datei erstellt wird, bzw. `False`, wenn sie als ASCII-Datei erstellt wird. Wird für den Wert keine Angabe gemacht, wird von einer ASCII-Datei ausgegangen.

Tabelle 86: Die Argumente der Methode CreateTextFile

Im Makro aus Listing 203 wird eine Textdatei angelegt und eine Zeile erfasst. Dabei werden das aktuelle Datum sowie die aktuelle Uhrzeit über die Funktion `Now` festgehalten.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=================================================

Sub TextdateiSchreiben()
   Dim fs As Object
   Dim obj As Object

   Set fs = CreateObject("Scripting.FileSystemObject")
   Set obj = fs.CreateTextFile("C:\iniHeld.txt", True)
   obj.WriteLine (Now)
   obj.Close
End Sub
```

Listing 203: Eine Textdatei anlegen und füllen

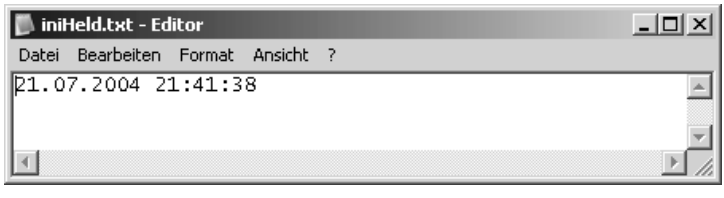


Abbildung 106: Eine »Ini-Datei« anlegen

179 Das FileSearch-Objekt

VBA stellt mit dem `FileSearch`-Objekt ein sehr mächtiges Objekt für Suchfunktionen zur Verfügung.

Bevor Sie an das Programmieren dieser Aufgabe herangehen, binden Sie die Bibliothek MICROSOFT SCRIPTING RUNTIME in der Entwicklungsumgebung ein. Diese Bibliothek wird dazu benötigt, die Datumsinformationen der Datenbanken abzufragen. Das vorherige Einbinden einer zusätzlichen Bibliothek, um an zusätzliche Befehle zu kommen, wird als early-binding bezeichnet. In Kapitel 10 erfahren Sie mehr zu diesem Thema.

Im nächsten Beispiel aus Listing 204 soll Ihre Festplatte nach Datenbanken abgesucht werden. Die Ergebnisse der Suche werden in das Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlFileS
'=================================================
```

Listing 204: Datenbanken finden und Dokumenteigenschaften auslesen

```
Function DB_Infos(FileName As Variant, DatNr As Integer) As String
   Dim fso As Scripting.FileSystemObject

   Set fso = New Scripting.FileSystemObject
   With fso.GetFile(FileName)
      If DatNr = 1 Then DB_Infos = .Name
      If DatNr = 2 Then DB_Infos = .DateCreated
      If DatNr = 3 Then DB_Infos = .DateLastModified
      If DatNr = 4 Then DB_Infos = .Size
   End With
End Function


Sub DBsSuchen()
   With Application.FileSearch
      .NewSearch
      .LookIn = "C:\Eigene Dateien\"
      .FileName = "*.mdb"
      .SearchSubFolders = True
      If .Execute() > 0 Then
         For Each varfile In .FoundFiles
            Debug.Print varfile
            Debug.Print DB_Infos(varfile, 1)
            Debug.Print DB_Infos(varfile, 2)
            Debug.Print DB_Infos(varfile, 3)
            Debug.Print DB_Infos(varfile, 4) & vbLf
         Next varfile
      End If
   End With
End Sub
```

Listing 204: Datenbanken finden und Dokumenteigenschaften auslesen (Forts.)

Über die Eigenschaft `Name` ermitteln Sie den Namen der jeweils gefundenen Datenbank.

Mithilfe der Eigenschaft `DateCreated` können Sie das Anlagedatum einer Datenbank bzw. auch eines Ordners ermitteln. Die Eigenschaft `DateLastModified` liefert Ihnen den genauen Zeitpunkt der letzten Änderung einer Datenbank. Die Eigenschaft `Size` liefert Ihnen die Größe der Datenbank in Byte.

Mithilfe des Objekts `FileSearch` kann nach bestimmten Dateien in Verzeichnissen gesucht werden. Dieses Objekt bietet einige Eigenschaften an, die angegeben werden können. Die Eigenschaft `FileName` bestimmt, nach welchen Dateien gesucht werden soll. Dabei kann die Endung von Access-Dateien *mdb* mit einem Sternchen angegeben werden. Es können somit alle Access-Datenbanken gesucht und gefunden werden, egal, wie diese auch immer heißen mögen. Über die Eigenschaft `LookIn` kann festgelegt werden, wo Access nach den Dateien suchen soll. Hier können Sie das Laufwerk sowie das Verzeichnis angeben. Dabei dürfen Sie nicht den letzten Backslash nach dem Verzeichnisnamen vergessen, da Access hierauf empfindlich reagiert. Sollen noch darunter liegende Verzeichnisse durchsucht werden, dann kann dies über die Eigenschaft `SearchSubFolders` festgelegt werden. Diese Eigenschaft wird auf den Wert `True` gesetzt, wenn die angegebene Suche alle Unterordner im durch die `LookIn`-Eigen-

schaft angegebenen Ordner einschließen soll. Die Methode `Execute` führt die jetzt näher spezifizierte Suche anschließend aus.

Nach der Suche sind alle gefundenen Dateien im Objekt `FoundFiles` verzeichnet. Diese gefundenen Dateien werden in einer anschließenden Schleife nacheinander verarbeitet. Innerhalb der Schleife werden die Dateien ausgewertet, indem die Funktion `DB_Info` eingesetzt wird. Mit der Methode `AddNew` wird dabei jeweils ein neuer Datensatz angelegt, danach gefüllt und über die Methode `Update` letztendlich gespeichert. Am Ende des Makros schließen Sie die Tabelle über die Methode `Close` und heben die Objektverweise auf, um den Arbeitsspeicher wieder freizugeben.

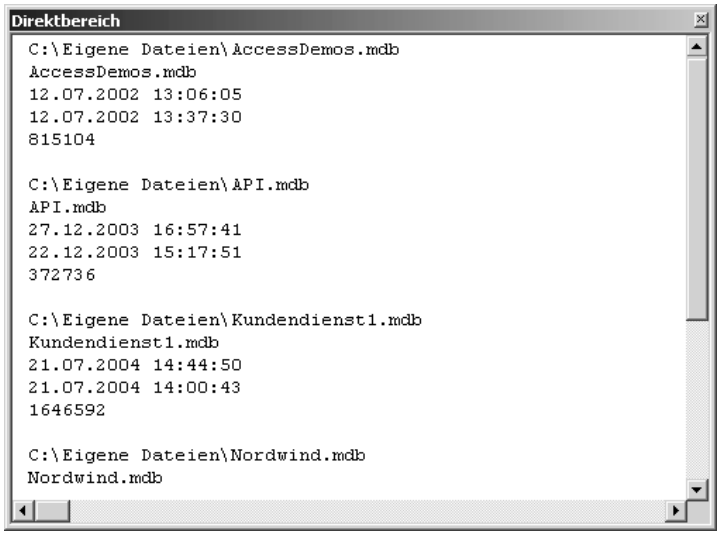


Abbildung 107: Datenbankinformationen über FileSearch ermitteln

180 Das CommandBar-Objekt

Alle Leisten in Access werden durch das Objekt `CommandBars` beschrieben. Die einzelnen Leisten können Sie über die Eigenschaft `Type` unterscheiden; so liefert eine Symbolleiste den Index 0, eine Menüleiste den Index 1 und ein Kontextmenü den Index 2. Weiterhin können Leisten ganz gezielt über einen eindeutigen ID-Wert angesprochen werden.

181 Leisten identifizieren

Das folgende Makro in Listing 205 schreibt alle Leisten von Access mit ihrem eindeutigen ID-Wert, Namen und Typ in den Direktbereich.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub LeistenIdentifizieren()
   Dim intZ As Integer
   Dim strText As String

   For intZ = 1 To CommandBars.Count

      Select Case CommandBars(intZ).Type
         Case 0
            strText = "Symbolleiste"
         Case 1
            strText = "Menüleiste"
         Case 2
            strText = "Kontextmenü"
         Case Else
            strText = "nicht ermittelbar"
      End Select

      Debug.Print intZ & " --> " & _
      CommandBars(intZ).Name & " --> " & strText
   Next intZ
End Sub
```

Listing 205: Leisten über die Eigenschaft Type identifizieren

Ermitteln Sie im ersten Schritt alle vorhandenen Leisten von Access, indem Sie diese mithilfe der Methode `Count` zählen. Über die Eigenschaft `Type` geben Sie die Art der Befehlsleiste zurück. Entnehmen Sie die möglichen Konstanten bzw. Indexwerte der Tabelle 87.

Index	Konstante	Befehlsleiste
0	`msoBarTypeNormal`	Symbolleiste
1	`msoBarTypeMenuBar`	Menüleiste
2	`msoBarTypePopup`	Kontextmenü

Tabelle 87: Die möglichen Typen von Leisten in Access

Mit der Eigenschaft `Name` ermitteln Sie den Namen der Befehlsleiste. Den eindeutigen Index haben Sie bereits durch das Hochzählen der Variablen `i` herausgefunden. Über die Anweisung `Debug.Print` geben Sie alle Informationen über die vorhandenen Befehlsleisten im Direktbereich aus.

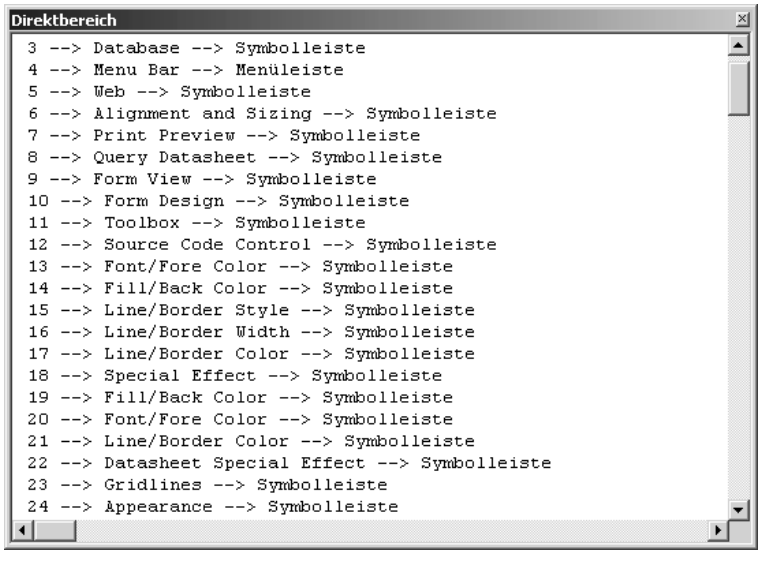


Abbildung 108: Alle Access-Leisten im Überblick

182 Symbolleiste ein- und ausblenden

Im folgenden Makro aus Listing 207 blenden Sie die Symbolleiste DATENBANK ein.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'======================================================

Sub SymbolleisteDatenbankAnzeigen()
   Application.CommandBars("Database").Visible = True
End Sub
```

Listing 206: Symbolleiste einblenden – Variante 1

Im folgenden Makro aus Listing 207 blenden Sie die Symbolleiste DATENBANK aus.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'======================================================
Sub SymbolleisteDatenbankAusblenden()
 Application.CommandBars("Database").Visible = False
End Sub
```

Listing 207: Symbolleiste ausblenden – Variante 1

Mithilfe der Eigenschaft `Visible` können Sie Symbolleisten ein- und ausblenden. Setzen Sie diese Eigenschaft auf den Wert `True`, wenn Sie eine Symbolleiste einblenden möchten. Verwenden Sie den Wert `False`, um eine Symbolleiste auszublenden.

Die Alternative

Alternativ zu der gerade vorgestellten Vorgehensweise können Sie auch mit der Methode `ShowToolbar` arbeiten, um eine Symbolleiste einzublenden. Im folgenden Beispiel aus Listing 208 blenden Sie die Symbolleiste WEB ein.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub SymbolleisteEinblendenAlternative()
   DoCmd.ShowToolbar "Web", acToolbarYes
End Sub
```

Listing 208: Symbolleiste einblenden – Variante 2

Um die Symbolleiste WEB wieder auszublenden, benutzen Sie dieselbe Methode, übergeben aber eine andere Konstante.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub SymbolleisteAusblendenAlternative()
   DoCmd.ShowToolbar "Web", acToolbarNo
End Sub
```

Listing 209: Symbolleiste ausblenden – Variante 2

183 Alle Symbolleisten ein- und ausblenden

Möchten Sie sich alle existierenden Symbolleisten anzeigen lassen, gehen Sie noch einen Schritt weiter. Den Code für diese Aufgabe sehen Sie in Listing 210.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================
```

Listing 210: Alle Symbolleisten anzeigen

```
Sub AlleSymbolleistenEinblenden()
   Dim intz As Integer

   For intz = 1 To Application.CommandBars.Count
      On Error Resume Next
      CommandBars(intz).Visible = True
   Next intz
End Sub
```

Listing 210: Alle Symbolleisten anzeigen (Forts.)

Um fast alle Symbolleisten auszublenden, starten Sie das Makro aus Listing 211.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub AlleSymbolleistenAusblenden()
   Dim intz As Integer

   For intz = 3 To Application.CommandBars.Count
      On Error Resume Next
      CommandBars(intz).Visible = False
   Next intz
End Sub
```

Listing 211: Fast alle Symbolleisten ausblenden

Lassen Sie dabei aber die Symbolleiste DATENBANK unberührt, indem Sie die Schleife erst mit dem Index 3 beginnen lassen.

184 Symbolleisten-IDs ermitteln

Jede einzelne Symbolschaltfläche in Access hat eine eindeutige ID, über die Sie die Symbolschaltfläche ansprechen können. Wie aber können Sie genau diese ID in Access ermitteln? Im Makro aus Listing 212 werden alle Symbole der Symbolleiste DATENBANK mit ihrem Namen und ihrem eindeutigen Menü in den Direktbereich geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================
```

Listing 212: Alle Symbol-IDs auflisten

```
Sub SymbolIDsAuflisten()
   Dim intz As Integer
   Dim Cmd As CommandBar

   Set Cmd = Application.CommandBars("Database")

   For intz = 1 To Cmd.Controls.Count
      Debug.Print Cmd.Controls(intz).Caption & " --> " & _
         Cmd.Controls(intz).Id
   Next intz
End Sub
```

Listing 212: Alle Symbol-IDs auflisten (Forts.)

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `CommandBar`. Danach geben Sie über die Anweisung `Set` den Namen der Symbolleiste bekannt, die Sie bearbeiten möchten. In einer `For each next`-Schleife durchlaufen Sie alle Symbole der Symbolleiste, ermitteln die Beschriftung der Symbolleiste, die standardmäßig nicht angezeigt wird, über die Eigenschaft `Caption` und geben diese über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus. Mithilfe der Eigenschaft `ID` kann das einzelne Symbol direkt angesprochen und identifiziert werden.

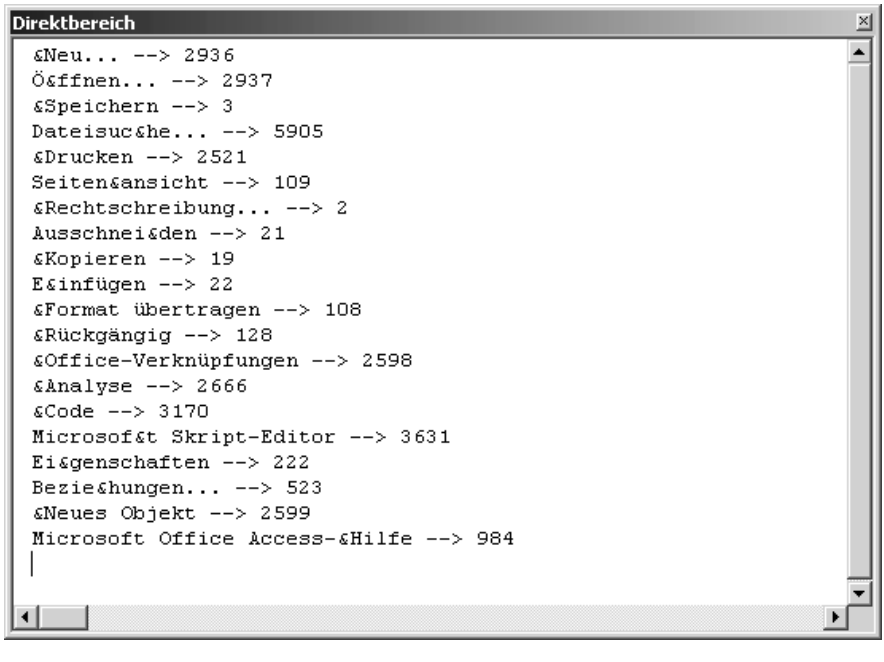


Abbildung 109: Alle IDs der Symbolleiste Datenbank werden dokumentiert.

185 Leisten und Befehle ermitteln

Im nächsten Listing 213 schreiben Sie die Namen aller Leisten und ihrer dazugehörigen Befehle in den Direktbereich.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub CommandBarsAuflisten()
   Dim intz As Integer
   Dim intSz As Integer
   Dim Cmd As Object

   Set Cmd = Application.CommandBars

   For intz = 1 To Cmd.Count
      Debug.Print "Das ist die Leiste: " & Cmd(intz).Name
      For intSz = 1 To Cmd(intz).Controls.Count
         Debug.Print Cmd(intz).Controls(intSz).Caption
      Next intSz
      Debug.Print vbLf
   Next intz
End Sub
```

Listing 213: Alle Symbolleisten sowie Symbole werden ermittelt.

Im Auflistungsobjekt `CommandBars` sind alle Leisten automatisch verzeichnet. In einer Schleife arbeiten Sie nacheinander eine Leiste nach der anderen ab, indem Sie über die Funktion `Count` die Anzahl der insgesamt zur Verfügung stehenden Leisten ermitteln.

In einer weiteren Schleife durchlaufen Sie alle Befehle der jeweiligen Leiste und ermitteln die Befehlsbeschriftung über die Eigenschaft `Caption`.

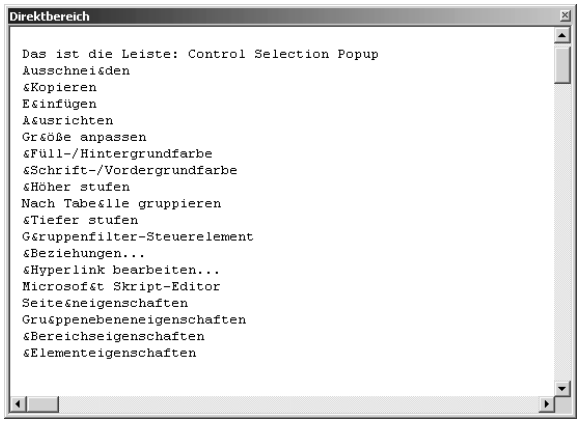


Abbildung 110: Die Namen sowie Symbole aller Symbolleisten dokumentieren

186 Symbolschaltflächen (de)aktivieren

Einzelne Symbole können Sie direkt ansprechen, indem Sie die genaue Anordnung in der Symbolleiste über die Eigenschaft `Controls` angeben.

Im Beispiel von Listing 214 wird in der Symbolleiste DATENBANK das Symbol NEU deaktiviert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub SymbolDeaktivieren()
   Dim cmdCtrl As CommandBarControl

   Application.CommandBars("Database").Controls(1).Enabled = False
End Sub
```

Listing 214: Symbol deaktivieren

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `CommandBarControl`. Damit stehen Ihnen alle Methoden und Eigenschaften für Symbole zur Verfügung, wie auch die Eigenschaft `Enabled`, die Sie auf den Wert `False` setzen, um das erste Symbol zu deaktivieren.

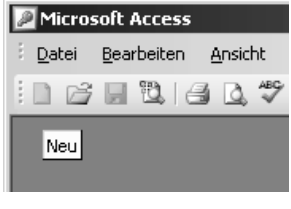


Abbildung 111: Das Symbol Neu ist deaktiviert worden.

Um diese Symbolschaltfläche wieder zu aktivieren, starten Sie das Makro aus Listing 215.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub SymbolAktivieren()
   Dim cmdCtrl As CommandBarControl
```

Listing 215: Das Symbol Neu wieder aktivieren

```
    Application.CommandBars("Database").Controls(1).Enabled = True
End Sub
```

Listing 215: Das Symbol Neu wieder aktivieren (Forts.)

187 Neue Symbolleiste erstellen

Wenn Sie Ihre eigene Symbolleiste anlegen möchten, die für Sie wichtige Funktionen enthält, dann können Sie eine Symbolleiste zuerst anlegen und später mit den gewünschten Symbolen belegen. Im ersten Schritt legen Sie nun eine neue, noch leere Symbolleiste an. Den Code für diese Aufgabe entnehmen Sie Listing 216.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub NeueSymbolleisteAnlegen()
   Dim cmd As CommandBar

   On Error Resume Next
   Set cmd = CommandBars.Add("Neue Symbolleiste")

   With cmd
     .Visible = True
     .Top = 400
     .Left = 70
   End With
End Sub
```

Listing 216: Eine neue Symbolleiste anlegen

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `CommandBar`. Mit der Methode `Add` fügen Sie eine neue Symbolleiste ein und weisen Sie der Objektvariablen über die Anweisung `Set` zu. Dabei bestimmen Sie über die Eigenschaft `Visible`, dass die Symbolleiste auf dem Bildschirm angezeigt wird. Mit den Eigenschaften `Top` und `Left` legen Sie die exakte Anzeigeposition (linke obere Ecke) fest. Sollte die Symbolleiste bereits angelegt worden sein, sorgt die Anweisung `On Error Resume Next` dafür, dass es zu keinem Makrofehler kommt. Die Methode `Add` wird in diesem Fall einfach ignoriert.

188 Symbolleiste löschen

Um eine Symbolleiste zu löschen, setzen Sie die Methode `Delete` aus Listing 217 ein. Sehen Sie sich dazu wieder die Anweisung `On Error Resume Next` an, die einen Makrofehler verhindert, wenn versucht wird, die bereits gelöschte Symbolleiste beispielsweise erneut zu löschen.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'====================================================

Sub SymbolleisteLöschen()
   On Error Resume Next
   Application.CommandBars("Neue Symbolleiste").Delete
End Sub
```

Listing 217: Symbolleiste löschen

189 Symbolschaltflächen-FaceIDs ermitteln

Im nächsten Schritt sollten Sie die noch leere Symbolleiste mit Ihren Wunschsymbolen befüllen. Jede einzelne Symbolschaltfläche in Access hat eine eindeutige ID, über die Sie die Symbolschaltfläche ansprechen können. Mit dieser ID ist eine eindeutige Funktion hinterlegt.

Neben der ID gibt es auch noch die so genannte `FaceId`. Diese `FaceId` ist ebenso eindeutig und gibt das Aussehen des Symbols wieder.

Wie aber können Sie genau diese `FaceId` in Access ermitteln? In der folgenden Aufgabe aus Listing 218 werden Sie die ersten 1000 Symbole in eine eigene Symbolleiste einfügen.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'====================================================

Sub SymbolFaceIDsErmitteln()
   Dim cmd As CommandBar
   Dim cmdctrl As CommandBarControl
   Dim intz As Integer

   On Error Resume Next
   Set cmd = Application.CommandBars.Add("Symbole", msoBarFloating)

   For intz = 1 To 1000
      Set cmdctrl = cmd.Controls.Add(msoControlButton)

      With cmdctrl
         .FaceId = intz
         .TooltipText = intz
      End With
   Next intz
```

Listing 218: Symbolleiste mit Symbolen bestücken

```
    cmd.Visible = True
End Sub
```

Listing 218: Symbolleiste mit Symbolen bestücken (Forts.)

Deklarieren Sie im ersten Schritt zwei Objektvariablen: Die Objektvariable `cmd` repräsentiert später die neue Symbolleiste. Über die Objektvariable `cmdCtrl` werden später die einzelnen Symbole in die Symbolleiste eingefügt. Über die Methode `Add` wird eine neue Symbolleiste eingefügt und benannt. Danach wird eine Schleife aufgesetzt, die genau 1000 Mal durchlaufen wird. Innerhalb der Schleife wenden Sie die Methode `Add` an, um die einzelnen Symbole einzufügen. Über die Eigenschaft `FaceId` wird das Aussehen des Symbols bestimmt. Mithilfe der Eigenschaft `ToolTipText` können Sie diese eindeutige Nummer als QuickInfo ausgeben.

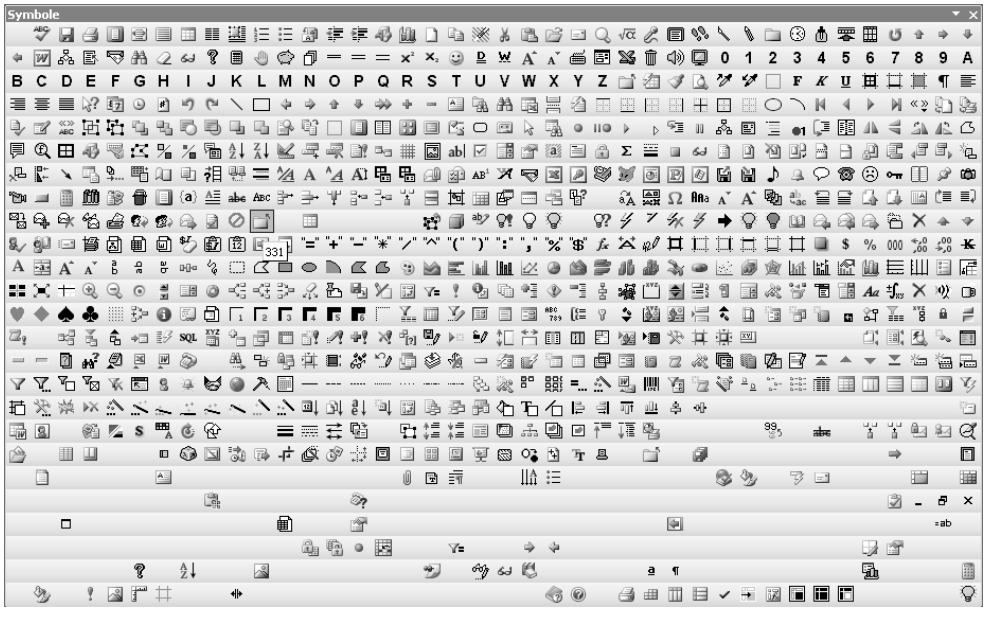


Abbildung 112: Die ersten 1000 Symbole

190 Symbolschaltflächen-IDs ermitteln

Sie haben nun das Aussehen der einzelnen Symbole über die FaceId herausgefunden. Es fehlt Ihnen noch die ID der einzelnen Symbole, die die Funktion der Symbolschaltfläche festlegt.

Im folgenden Beispiel aus Listing 219 wird die Symbolleiste DATENBANK untersucht. Alle IDs der Symbole dieser Schaltfläche werden in den Direktbereich geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub IDsErmitteln()
   Dim cmd As CommandBar
   Dim cmdctrl As CommandBarControl

   On Error Resume Next
   Set cmd = Application.CommandBars("Database")

   For Each cmdctrl In cmd.Controls
      Debug.Print cmdctrl.TooltipText & " --> " & cmdctrl.Id
   Next cmdctrl
End Sub
```

Listing 219: Die eindeutigen IDs von Symbolen ermitteln

Mithilfe der Eigenschaft `ToolTipText` können Sie die QuickInfo der einzelnen Symbole der Symbolleiste DATENBANK ermitteln. Mit der Eigenschaft `ID` ist die Aktion gemeint, die ausgeführt wird, wenn Sie eine Symbolschaltfläche anklicken.

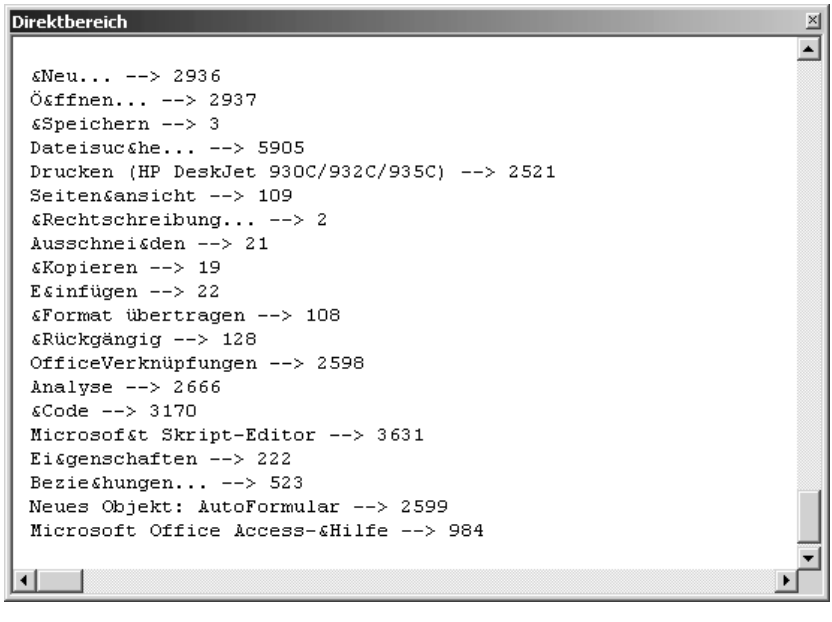


Abbildung 113: Die IDs der Symbole wurden dokumentiert.

191 Symbolschaltflächen einfügen

Da Sie nun alle erforderlichen Informationen haben, um die noch leere Symbolleiste mit den gewünschten Symbolen auszustatten, gehen Sie ans Werk.

Die Symbolleiste soll folgende Symbole enthalten:

- Das SPEICHERN-Symbol
- Das NEU-Symbol
- Das ÖFFNEN-Symbol
- Ein benutzerdefiniertes Symbol, dem Sie ein eigenes Makro zuweisen

Im folgenden Makro aus Listing 220 wird die Symbolleiste NEUE SYMBOLLEISTE angelegt und die gerade beschriebenen Funktionen werden integriert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteEinfügenUndFüllen()
   Dim cmd As CommandBar
   Dim cmdctrl As CommandBarControl

   On Error GoTo fehler
   Set cmd = CommandBars.Add("Neue Symbolleiste")

   'Speichern-Symbol einfügen
   Set cmdctrl = cmd.Controls.Add(msoControlButton, 3)

   'Symbol Neu einfügen
   Set cmdctrl = cmd.Controls.Add(msoControlButton, 2936)

   'Symbol Öffnen einfügen
   Set cmdctrl = cmd.Controls.Add(msoControlButton, 2937)

   'Benutzerdefiniertes Symbol einfügen
   Set cmdctrl = cmd.Controls.Add(msoControlButton)
   With cmdctrl
      .FaceId = 125
      .TooltipText = "Aktuelles Datum"
      .OnAction = "DatumAusgeben"
   End With

   Set cmd = Nothing
   Set cmdctrl = Nothing
   Exit Sub
```

Listing 220: Eine neue Symbolleiste anlegen und mit Symbolen bestücken

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 220: Eine neue Symbolleiste anlegen und mit Symbolen bestücken (Forts.)

Geben Sie in der Anweisung `Set` im Zusammenspiel mit der Methode `Add` an, um die neue Symbolleiste anzulegen, in welche Sie die Symbole einfügen möchten. Danach fügen Sie mit der Methode `Add` die einzelnen Symbole ein. Um dabei auf bereits fertige Funktionen von Access zurückzugreifen, genügt es, wenn Sie bei der Methode `Add` die dazugehörige `ID` mit angeben, die Sie vorher bereits über das Makro aus Listing 219 ermittelt haben. Damit werden das Aussehen sowie die dazugehörige QuickInfo mit der Beschreibung der Funktion automatisch mit angelegt.

Abbildung 114: Die neue Symbolleiste wurde angelegt.

Bei eigenen Symbolen müssen Sie die Eigenschaft `FaceId` einsetzen, um das Aussehen des Symbols festzulegen. Um eine QuickInfo zu definieren, die angezeigt werden soll, sobald der Anwender den Mauszeiger auf dem Symbol positioniert, setzen Sie die Eigenschaft `ToolTipText` ein. Damit überhaupt etwas passiert, wenn der Anwender auf das benutzerdefinierte Symbol klickt, legen Sie über die Eigenschaft `OnAction` fest, welches Makro beim Klick auf das Symbol ausgeführt werden soll. Sie müssen zu diesem Zweck das Makro noch erfassen. Dieses Makro sehen Sie in Listing 221.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub DatumAusgeben()
   MsgBox "Heute ist der " & Date
End Sub
```

Listing 221: Das aktuelle Datum ausgeben

192 Symbolleisten schützen

Sie können Symbolleisten jederzeit anpassen, d.h., Sie können neue Symbole in die Symbolleiste aufnehmen oder Symbole aus den Leisten herausnehmen. Des Weiteren können Sie die

Position von Symbolleisten auf dem Bildschirm frei bestimmen. Möchten Sie all dies verhindern, so haben Sie die Möglichkeit, Ihre Symbolleisten zu schützen.

Im nächsten Makro aus Listing 222 wird die Symbolleiste DATENBANK geschützt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'====================================================

Sub SymbolleisteSchützen()
   With Application.CommandBars("Database")
      .Protection = msoBarNoChangeVisible + msoBarNoCustomize
      .Visible = True
   End With
End Sub
```

Listing 222: Symbolleisten schützen

Setzen Sie die Eigenschaft `Protection` ein, um Ihre Symbolleisten zu schützen. Die Konstante `msoBarNoChangeVisible` bewirkt, dass die Symbolleiste nicht im Kontextmenü erscheint, wenn Sie eine beliebige Symbolleiste mit der rechten Maustaste anklicken. Die Konstante `msoBarNoCustomize` verhindert ein Anpassen der Symbolleiste. Sie haben dadurch keine Möglichkeit, neue Symbole hinzuzufügen bzw. Symbole aus der Symbolleiste herauszunehmen.

Entnehmen Sie Tabelle 88 die Möglichkeiten, die Sie mit der Eigenschaft `Protection` haben.

Konstante	Bedeutung
`msoBarNoChangeDock`	Die Symbolleiste kann nicht aus ihrer Verankerung herausgelöst werden.
`msoBarNoChangeVisible`	Die Symbolleiste können Sie weder im Kontextmenü der Symbolleisten noch im Dialogfeld ANPASSEN sehen.
`msoBarNoCustomize`	Das Hinzufügen bzw. Löschen von Symbolen aus der Symbolleiste ist nicht möglich, ebenso wenig das Verschieben der Symbole.
`msoBarNoHorizontalDock`	Die Symbolleiste kann weder am oberen noch am unteren Bildschirm angedockt werden.
`msoBarNoVerticalDock`	Die Symbolleiste kann weder rechts noch links am Bildschirm angedockt werden.
`msoBarNoMove`	Die Symbolleiste kann nicht auf dem Bildschirm frei bewegt werden.
`msoBarNoResize`	Die Symbolleiste kann in ihrer Form nicht verändert werden.

Tabelle 88: Die Möglichkeiten zum Schutz von Symbolleisten

Heben Sie den Schutz wieder auf, indem Sie das folgende Makro aus Listing 223 starten.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleistenschutzAufheben()
   With Application.CommandBars("Database")
      .Protection = False
      .Visible = True
   End With
End Sub
```

Listing 223: Symbolleistenschutz wieder aufheben

Setzen Sie die Eigenschaft `Protection` auf den Wert `False`, um den Schutz der Symbolleiste wieder aufzuheben.

193 Symbolschaltflächen (de)aktivieren

Sie können in Access auch bestimmte Symbolschaltflächen deaktivieren und bei Bedarf wieder aktivieren. Dies empfiehlt sich u.a., wenn Sie auf bestimmte Anlässe in Access reagieren möchten. So können Sie z.B. eine Symbolschaltfläche standardmäßig deaktivieren und erst dann aktivieren, wenn Sie z.B. eine Kopieraktion durchführen. So sind Sie in der Lage, am Status eines Symbols abzulesen, ob Sie sich gerade im Kopiermodus befinden.

In der nächsten Aufgabe werden Sie zwei Symbole aus der Symbolleiste NEUE SYMBOLLEISTE deaktivieren. Es handelt sich um die Symbole SPEICHERN und NEU. Starten Sie für diese Aufgabe das Makro aus Listing 224.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================
Sub DeaktivierenSymbole()
   Dim cmd As CommandBar

   Set cmd = CommandBars("Neue Symbolleiste")
   cmd.Visible = True
   With cmd
      .Controls(1).Enabled = False
      .Controls(2).Enabled = False
   End With
End Sub
```

Listing 224: Einzelne Symbole deaktivieren

Die Deaktivierung einzelner Symbole bewirkt, dass diese abgeblendet dargestellt werden. Wenn Sie auf eines dieser Symbole klicken, erfolgt keine Reaktion. Damit Sie das richtige Symbol deaktivieren, arbeiten Sie am besten über die eindeutige ID. Geben Sie vorher an, in welcher Symbolleiste Sie das gewünschte Symbol deaktivieren möchten, und deaktivieren Sie es schließlich, indem Sie die Eigenschaft `Enabled` auf den Wert `False` setzen.

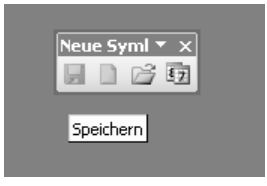


Abbildung 115: Die ersten beiden Symbole wurden deaktiviert.

Um diese beiden Symbole wieder zu aktivieren, setzen Sie die Eigenschaft `Enabled` für diese Symbole wieder auf `True`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub AktivierenSymbole()
   Dim cmd As CommandBar

   Set cmd = CommandBars("Neue Symbolleiste")
   cmd.Visible = True
   With cmd
      .Controls(1).Enabled = True
      .Controls(2).Enabled = True
   End With
End Sub
```

Listing 225: Symbole wieder aktivieren

194 Adaptive Menüs ausschalten

Bei der Ermittlung der korrekten Einfügeposition kann Ihnen aber die seit der Access-Version 2000 bestehende Funktion mit den adaptiven Menüs einen Strich durch die Rechnung machen. Bei diesem Feature handelt es sich um ein sich ständig selbst anpassendes System, nach dem Menüleisten angeordnet werden. Die am häufigsten verwendeten Funktionen werden in den Menüs ganz oben angeordnet. Die seltener verwendeten Access-Funktionen rutschen immer weiter in den Menüs nach unten und werden erst nach längerem Verweilen mit der Maus im Menü dynamisch eingeblendet. Es empfiehlt sich daher, dieses neue Feature auszuschalten. Das Ausschalten der adaptiven Menüs können Sie manuell vornehmen,

indem Sie mit der rechten Maustaste auf eine beliebige Symbolleiste klicken und aus dem Kontextmenü den Befehl ANPASSEN auswählen. Wechseln Sie danach auf die Registerkarte OPTIONEN und deaktivieren Sie das Kontrollkästchen MENÜS ZEIGEN ZULETZT VERWENDETE BEFEHLE ZUERST AN (Access 2000) bzw. aktivieren Sie das Kontrollkästchen MENÜS IMMER VOLLSTÄNDIG ANZEIGEN (Access 2002 und Access 2003).

Diese manuelle Einstellung können Sie aber auch über das Makro aus Listing 226 deaktivieren.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub AdaptiveMenüsAusschalten()
   Application.CommandBars.AdaptiveMenus = False
End Sub
```

Listing 226: Adaptive Menüs ausschalten

Setzen Sie die Eigenschaft `AdaptiveMenus` auf den Wert `False`, um die personalisierten Menüs abzuschalten. Dadurch werden die Menüs wie gewohnt in Access angezeigt.

Hinweis

Möchten Sie ausgiebig mit Menü- und Symbolleisten arbeiten, müssen Sie noch die Bibliothek MICROSOFT OFFICE 10 OBJECT LIBRARY für Office XP bzw. MICROSOFT OFFICE 11 OBJECT LIBRARY für Office 2003 über den Menübefehl EXTRAS/VERWEISE aktivieren. Damit stehen Ihnen dann alle Methoden und Eigenschaften zur Verfügung, die Ihnen Office anbietet.

195 Neues Menü einfügen

In der nächsten Aufgabe soll ein zusätzliches Menü genau vor dem HILFE-Menü eingefügt werden, das ein paar nützliche Funktionen aufnehmen soll. Denken Sie daran, dass Sie vorher die Bibliothek MICROSOFT OFFICE 11 OBJECT LIBRARY in Ihrer Entwicklungsumgebung einbinden.

Den dazu notwendigen Code können Sie aus Listing 227 entnehmen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================
```

Listing 227: Neues Menü einfügen

```
Sub NeuesMenüEinfügen()
   Dim intz As Integer
   Dim intPosHilfe As Integer
   Dim cmdctrl As CommandBarControl

   On Error GoTo fehler
   intz = Application.CommandBars("Menu Bar").Controls.Count
   intPosHilfe = Application.CommandBars("Menu Bar").Controls(intz).Index

   Set cmdctrl = Application.CommandBars("Menu Bar"). _
   Controls.Add(Type:=msoControlPopup, Before:=intPosHilfe, Temporary:=True)
   cmdctrl.Caption = "&Eigene Funktionen"

   Set cmdctrl = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 227: Neues Menü einfügen (Forts.)

Definieren Sie im ersten Schritt zwei Integer-Variablen, die zum einen die Anzahl der Menüs ermitteln, die momentan in der Menüleiste eingebunden sind, und zum anderen die Position des HILFE-Menüs ermitteln. Eine weitere Objektvariable vom Typ `CommandBarControl` wird gebraucht, um den neuen Menüpunkt einzufügen. Über die Methode `Count` zählen Sie die Anzahl der Menüs in der Menüleiste und speichern sie in der Variablen `intz`. Im nächsten Schritt ermitteln Sie die Position des HILFE-Menüs, das standardmäßig ganz rechts in der Arbeitsblatt-Menüleiste steht. Die Menüleiste können Sie über das Objekt `CommandBars("Menu Bar")`ansprechen. Über die Eigenschaft `Controls` bekommen Sie alle Steuerelemente der angegebenen Menüleiste angezeigt.

Mithilfe der Methode `Add` fügen Sie ein neues Menü ein. Die Methode `Add` hat die Syntax:

```
Add(Type, Id, Before, Temporary)
```

Beim Argument `Type` geben Sie an, um welche Art von Steuerelement es sich handeln soll. Zur Auswahl stehen die Konstanten aus der folgenden Tabelle.

Konstante	Beschreibung
`msoControlButton`	Fügt ein Schaltflächenelement ein.
`msoControlEdit`	Fügt ein Eingabefeld ein.
`msoControlDropdown`	Fügt ein Dropdown-Feld ein,
`msoControlComboBox`	Fügt ebenso ein Dropdown-Feld ein.
`msoControlPopup`	Fügt ein Dropdown-Menü ein.

Tabelle 89: Alle möglichen Konstanten für Steuerelemente in Menüleisten

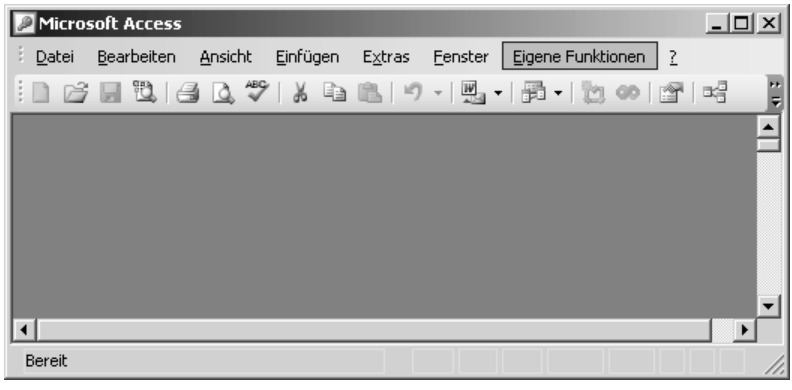


Abbildung 116: Das neue Menü wurde eingefügt.

Beim Argument `ID` können Sie sich entscheiden, ob Sie zusätzlich zum Menütext auch noch ein Symbol anzeigen möchten. Dieses Argument funktioniert jedoch nur innerhalb eines Menüs, also für einen Menübefehl. Mit dem Argument `Before` legen Sie die genaue Position des Menüs fest. Übergeben Sie dem Argument die vorher ermittelte Position des Hilfe-Menüs. Setzen Sie das letzte Argument `Temporary` auf den Wert `True`, wenn das neue Steuerelement temporär sein soll. Temporäre Steuerelemente werden automatisch gelöscht, wenn die Containeranwendung geschlossen wird.

196 Menüleiste zurücksetzen

Haben Sie ein wenig mit Ihrer Menüleiste herumexperimentiert und möchten Sie diese ganz schnell wieder in ihren ursprünglichen Zustand zurücksetzen, dann starten Sie das Makro aus Listing 228.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================
Sub MenüLeisteZurücksetzen()
   On Error Resume Next
   Application.CommandBars("Menu Bar").Reset
End Sub
```

Listing 228: Menüleiste zurücksetzen

Die Methode `Reset` setzt die angegebene, integrierte Befehlsleiste auf die Standardkonfiguration der Steuerelemente zurück.

197 Menü löschen

Wenn Sie noch weitere zusätzliche Menüs in Ihrer Menüleiste haben, dann macht es keinen Sinn, die komplette Menüleiste zurückzusetzen. Manche Hersteller von Access-Tools integrieren ebenso weitere Menüs bzw. Menüunterpunkte in die Menüleiste von Access. Für den Fall, dass Sie nun die Methode `Reset` einsetzen, sind auch diese erwünschten Einträge weg. In diesem Fall müssen Sie ganz gezielt Ihr eigenes Menü herauslöschen. Sehen Sie sich dazu das Makro aus Listing 229 an.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub MenüLöschen()
   On Error Resume Next
   Application.CommandBars("Menu Bar").Controls("Eigene Funktionen").Delete
End Sub
```

Listing 229: Menü löschen

Wenden Sie die Methode `Delete` an, um das neu eingefügte Menü wieder zu löschen.

198 Menübefehle einfügen

Das gerade eingefügte Menü enthält noch keine Menübefehle. Diese fügen Sie jetzt ein. Erweitern Sie das vorher bereits erfasste Makro aus Listing 227 wie folgt:

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=================================================

Sub NeuesMenüEinfügen()
   Dim intz As Integer
   Dim intPosHilfe As Integer
   Dim cmdctrl As CommandBarControl
   Dim cmdctrl2 As CommandBarControl

   On Error GoTo fehler
```

Listing 230: Neues Menü und Menübefehle einfügen

```
    intz = Application.CommandBars("Menu Bar").Controls.Count
    intPosHilfe = Application.CommandBars("Menu Bar").Controls(intz).Index

    Set cmdctrl = Application.CommandBars("Menu Bar"). _
    Controls.Add(Type:=msoControlPopup, Before:=intPosHilfe, Temporary:=True)
    cmdctrl.Caption = "&Eigene Funktionen"

    Set cmdctrl2 = cmdctrl.Controls.Add(Type:=msoControlButton)
    With cmdctrl2
       .Caption = "Speicherort der Datei anzeigen"
       .Style = msoButtonIconAndCaption
       .OnAction = "Verzeichniszurückgeben"
       .FaceId = 23
    End With

    Set cmdctrl2 = cmdctrl.Controls.Add(Type:=msoControlButton)
    With cmdctrl2
       .Caption = "Wo&chentag, Datum und Uhrzeit anzeigen"
       .Style = msoButtonCaption
       .OnAction = "DatumUndUhrzeit"
       .BeginGroup = True
    End With

    Set cmdctrl2 = Nothing
    Set cmdctrl = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 230: Neues Menü und Menübefehle einfügen (Forts.)

Für die Menübefehle im Menü EIGENE FUNKTIONEN benötigen Sie eine weitere Objektvariable vom Typ `CommandBarControl`. Wenden Sie die Methode `Add` auf das neu eingefügte Menü an, um nun die einzelnen Menübefehle hinzuzufügen. Mit der Anweisung `With` legen Sie übersichtlich weitere Eigenschaften der neuen Menübefehle fest.

Mit der Eigenschaft `Caption` legen Sie die Beschriftung des Menübefehls fest. Verwenden Sie das kaufmännische Zeichen &, um den Shortcut für diesen Befehl zu bestimmen. Ist das Menü einmal aktiviert, können Sie durch die Eingabe des unterstrichenen Buchstabens innerhalb des Menübefehls die dahinter liegende Funktion bzw. das Makro starten.

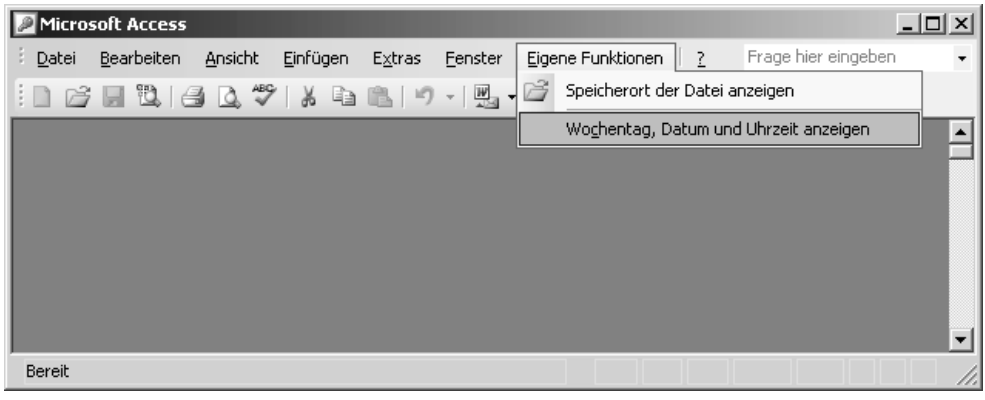


Abbildung 117: Das neue Menü wurde angelegt und mit Menübefehlen ausgestattet.

Hinweis

Über die Eigenschaft `FaceId` können Sie dem Menübefehl auch noch ein Symbol hinzufügen. Allerdings muss dabei dann die Eigenschaft `Styles` mit der Konstante `msoButtonIconAndCaption` angegeben werden.

199 Menübefehle gruppieren

Fassen Sie Menübefehle optisch zusammen, indem Sie eine Trennlinie zwischen einzelnen Menübefehlen ziehen. Diesen Effekt erreichen Sie über die Eigenschaft `BeginGroup`. Setzen Sie diese Eigenschaft auf den Wert `True`, wenn sich der angegebene Menübefehl am Anfang einer Gruppe von Menübefehlen im Menü befindet.

200 Menü auslesen

Um speziell die Menüleiste von Access auszulesen, setzen Sie den Code aus Listing 231 ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=======================================================

Sub MenüNamenAusgeben()
   Dim cmd As CommandBar
   Dim Ctrl As CommandBarControl
   Set cmd = Application.CommandBars("Menu Bar")

   For Each Ctrl In cmd.Controls
      Debug.Print Ctrl.Caption
   Next ctrl
End Sub
```

Listing 231: Menüs auslesen

Deklarieren Sie zuerst zwei Objektvariablen – eine davon vom Typ `CommandBar` und die zweite als `CommandBarControl`. Über die Anweisung `Set` weisen Sie der Objektvariablen `cmd` die Menüleiste zu. Danach setzen Sie eine `For each next`-Schleife auf und geben dabei alle Menüs der Menüleiste über die Eigenschaft `Caption` und die Anweisung `Debug.Print` aus.

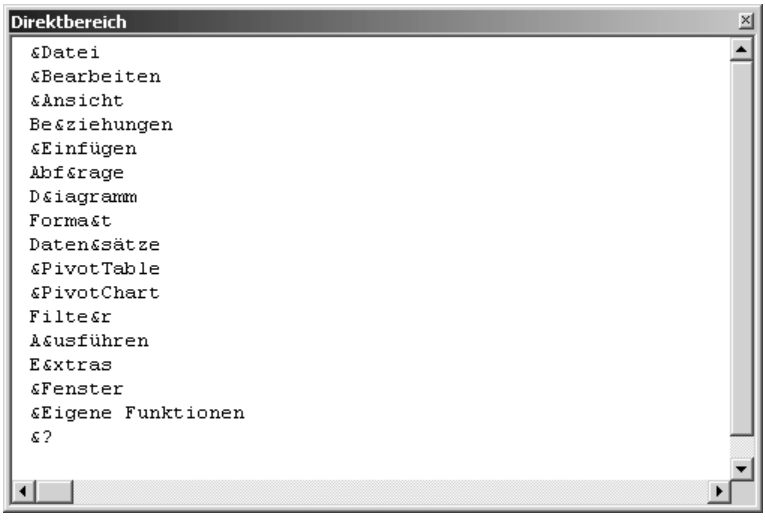


Abbildung 118: Die Menünamen wurden dokumentiert.

201 Menübefehle auslesen

Möchten Sie mehr über ein bestimmtes Menü erfahren, hilft Ihnen das Makro aus Listing 232. Es schreibt alle Menübefehle des ersten Menüs in den Direktbereich.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub MenüsAuslesen()
   Dim cmdctrl As CommandBarControl
   Dim cmdctrl2 As CommandBarControl

   Set cmdctrl = Application.CommandBars("Menu Bar").Controls(1)

   For Each cmdctrl2 In cmdctrl.Controls
      Debug.Print cmdctrl2.Caption
   Next cmdctrl2
End Sub
```

Listing 232: Menübefehle eines Menüs auslesen

Indem Sie die Eigenschaft `Controls` auf den Wert 1 setzen, bestimmen Sie, dass das Menü DATEI gemeint ist.

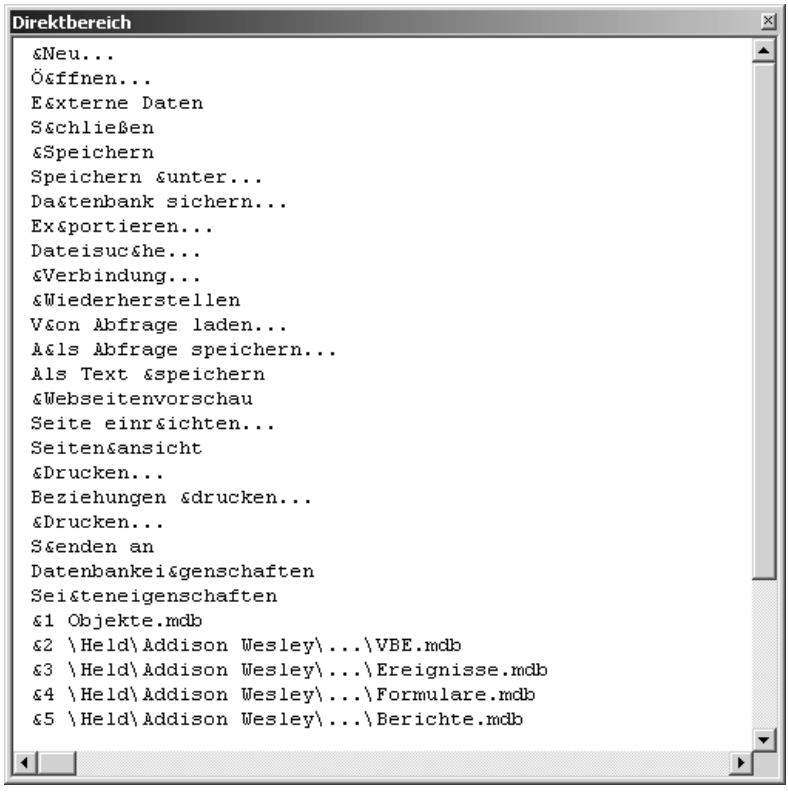


Abbildung 119: Alle Befehle des ersten Menüs wurden dokumentiert.

202 Menüs (de)aktivieren

Neben dem Ansprechen einzelner Menüs mit einem Index haben Sie ebenso die Möglichkeit, ganz gezielt bestimmte Menübefehle zu deaktivieren. Diese Menübefehle werden dann abgeblendet und können nicht mehr ausgewählt werden.

Im folgenden Listing 233 werden alle Menübefehle im Menü EINFÜGEN deaktiviert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================
```

Listing 233: Alle Menübefehle deaktivieren

```
Sub AlleMenübefehleInLeisteDeaktivieren()
   Dim cmdctrl As CommandBarControl
   Dim cmdCtrl2 As CommandBarControl

   Set cmdctrl = Application.CommandBars("Menu Bar").Controls(5)

   For Each cmdCtrl2 In cmdctrl.Controls
      cmdCtrl2.Enabled = False
   Next cmdCtrl2
End Sub
```

Listing 233: Alle Menübefehle deaktivieren (Forts.)

Setzen Sie die Eigenschaft `Enabled` auf den Wert `False`, um die einzelnen Menübefehle zu deaktivieren.

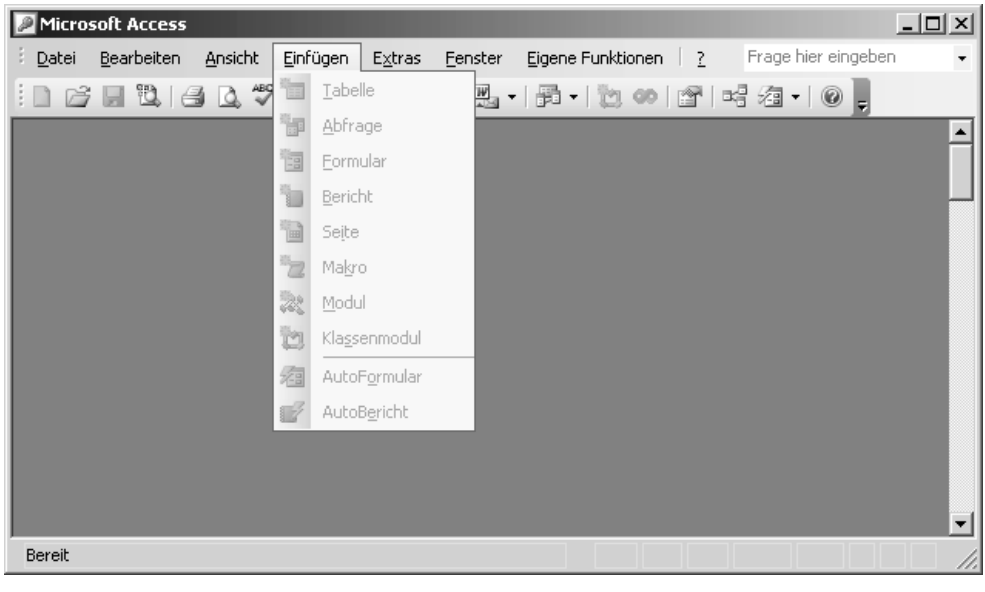


Abbildung 120: Alle Menübefehle im Menü Einfügen wurden deaktiviert.

Um die Menübefehle wieder zu aktivieren, starten Sie das Makro aus Listing 234.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub AlleMenübefehleInLeisteAktivieren()
```

Listing 234: Alle Menübefehle eines Menüs wieder aktivieren

```
    Dim cmdctrl As CommandBarControl
    Dim cmdCtrl2 As CommandBarControl

    Set cmdctrl = Application.CommandBars("Menu Bar").Controls(5)

    For Each cmdCtrl2 In cmdctrl.Controls
       cmdCtrl2.Enabled = True
    Next cmdCtrl2
End Sub
```

Listing 234: Alle Menübefehle eines Menüs wieder aktivieren (Forts.)

203 Menübefehle (de)aktivieren

Natürlich können Sie auch gleich ganz oben im Menü den Menüpunkt deaktivieren. Dadurch kann das entsprechende Menü nicht mehr heruntergeklappt werden. Den Code für diese Anweisung entnehmen Sie Listing 235:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap03
' Dateiname  Objekte.mdb
' Modul      MdlCmd
'=====================================================

Sub MenüDeaktivieren()
   'Menü Einfügen deaktivieren
   Application.CommandBars("Menu Bar").Controls(5).Enabled = False
End Sub
```

Listing 235: Ganzes Menü deaktivieren

Setzen Sie die Eigenschaft `Enabled` auf den Wert `True`, um das Menü wieder zur Verfügung zu stellen.

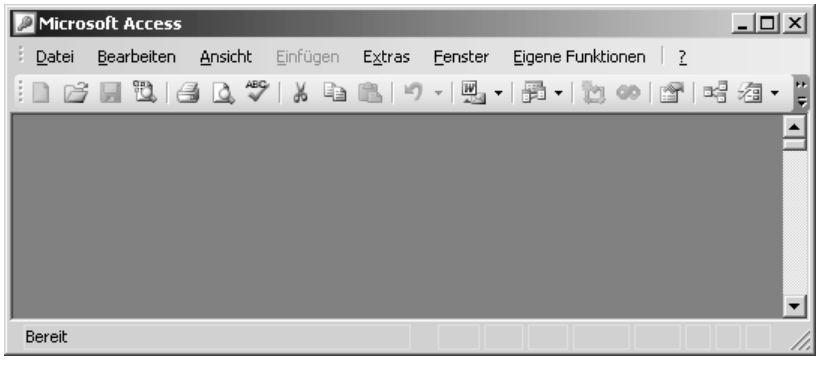


Abbildung 121: Ein ganzes Menü wurde deaktiviert.

Tabellen programmieren

Einfachere Aufgaben rund um das Thema Tabellen können Sie mithilfe des Objekts `DoCmd` erledigen. Wenn es etwas mehr sein soll, dann greifen Sie auf die Datenzugriffsmethode ADO (ACTIVEX DATA OBJECTS) zurück. Diese Methode zeichnet sich durch eine hohe Verarbeitungsgeschwindigkeit, eine benutzerfreundliche Bedienung sowie wenig Verbrauch an Arbeitsspeicher und Festplattenspeicher aus. Neben dieser Methode gibt es noch eine zweite, etwas ältere und sehr verbreitete Methode, genannt DAO (DATA ACCESS OBJECTS).

In diesem Kapitel können Sie neben ausgewählten Aufgaben mit dem Objekt `DoCmd` auch Objekte, Methoden und Eigenschaften von ADO/DAO nachschlagen und anhand praktischer Beispiele deren Einsatz nachvollziehen.

204 Tabellen öffnen

Als erste Aufgabe öffnen Sie im Makro Listing 236 über die Methode `OpenTable` die Tabelle KUNDEN.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=======================================================

Sub TabelleÖffnen()
   On Error GoTo fehler
   DoCmd.OpenTable "Kunden", acViewNormal
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 236: Tabelle öffnen

Mithilfe der Methode `OpenTable` öffnen Sie eine Tabelle in Access. Diese Methode hat folgende Syntax:

```
OpenTable(Tabellenname, Ansicht, Datenmodus)
```

Im Argument `Tabellenname` geben Sie den Namen der Tabelle an, die Sie öffnen möchten.

Beim Argument `Ansicht` können Sie entscheiden, wie Sie Ihre Tabelle anzeigen möchten. Es stehen Ihnen dabei folgende Konstanten zur Verfügung:

- `acViewDesign`: Öffnet die Tabelle in der Entwurfsansicht.
- `acViewNormal`: Öffnet die Tabelle in gewohnter Weise in der Datenblattansicht (Standardeinstellung).
- `acViewPivotChart`: Stellt die Tabelle für ein Pivot-Diagramm zur Verfügung.
- `acViewPivotTable`: Hiermit können Sie die Felder der Tabelle für eine Pivot-Tabelle verwenden.
- `acViewPreview`: Zeigt die Tabelle in der Seitenansicht an.

Beim letzten Argument `Datenmodus` legen Sie fest, ob Änderungen an der Tabelle durchgeführt werden dürfen oder nicht. Dabei können Sie folgende Konstanten festlegen:

- `acAdd`: Der Anwender kann neue Datensätze hinzufügen, jedoch keine bestehenden Datensätze bearbeiten.
- `acEdit`: Der Anwender kann bestehende Datensätze bearbeiten und neue Datensätze hinzufügen.
- `acReadOnly`: Der Anwender kann die Datensätze nur ansehen.

205 Datensatz finden

Über die Methode `FindRecord` haben Sie die Möglichkeit, schon beim Aufruf der Tabelle einen bestimmten Datensatz anzeigen zu lassen. Im Makro aus Beispiel Listing 237 wird die Tabelle `Kunden` geöffnet und der Datensatz

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=================================================

Sub TabelleÖffnenUndSatzAktivieren()
   On Error GoTo fehler
   DoCmd.OpenTable "Kunden", acViewNormal
   DoCmd.FindRecord "Ernst Handel", acEntire, True, acSearchAll, True, acAll
Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 237: Einen Datensatz aktivieren

Die Methode `FindRecord` hat folgende Syntax:

```
FindRecord(SuchenNach, Vergleichen, GroßKlein, Suchen, WieFormatiert,
NurAktuellesFeld, AmAnfangBeginnen)
```

Im Argument SuchenNach geben Sie an, nach welchem Text Sie in der Tabelle suchen möchten.

Das Argument Vergleichen gibt an, wo sich die Daten im Feld befinden. Sie können eine Suche nach Daten in einem beliebigen Teil des Felds (Teil des Feldinhalts = acAnywhere), Daten, die das gesamte Feld ausfüllen (gesamter Feldinhalt = acEntire), oder Daten, die sich am Anfang des Felds befinden (Anfang des Feldinhalts = acStart), angeben. Als Standardeinstellung ist immer der gesamte Feldinhalt, also die Konstante acEntire, vorgesehen.

Im Argument GroßKlein geben Sie an, ob Access bei der Suche zwischen Groß- und Kleinschreibung unterscheiden soll. Wenn ja, dann setzen Sie dieses Argument auf den Wert True.

Das Argument Suchen legt die Suchreihenfolge fest. Sie können dabei die folgenden Konstanten einsetzen:

- acDown: Suche vom aktuellen Datensatz bis zum Ende der Tabelle.
- acUp: Suche vom aktuellen Datensatz bis zum Anfang der Tabelle.
- acSearchAll: Suche bis zum Ende der Datensätze und dann vom Anfang der Datensätze bis zum aktuellen Datensatz. Somit werden alle Datensätze durchsucht. Bei dieser Einstellung handelt es sich um die Standardeinstellung.

Mithilfe des Arguments WieFormatiert können Sie bestimmen, ob die Suche auch formatierte Daten umfasst. Setzen Sie dieses Argument auf den Wert True, um nach Daten zu suchen sowie nach Informationen, wie diese formatiert sind und im Feld angezeigt werden. Setzen Sie dieses Argument hingegen auf den Wert False, so sucht Access nach Daten, die in der Datenbank gespeichert sind. Die Standardeinstellung lautet False.

Beim Argument NurAktuellesFeld können Sie bestimmen, ob Access seine Suche nur auf die momentan aktive Spalte beziehen soll. In diesem Fall geben Sie die Konstante acCurrent an. Möchten Sie den angegebenen Suchbegriff in allen Zellen der Tabelle suchen, dann setzen Sie die Konstante acAll ein.

Das letzte Argument AmAnfangBeginnen bestimmt, ob die Suche beim ersten Satz oder beim aktuellen Datensatz beginnen soll. Geben Sie diesem Argument den Wert True, um die Suche beim ersten Datensatz zu beginnen. Verwenden Sie den Wert False, um die Suche im Datensatz zu beginnen, der auf den aktuellen Datensatz folgt. Wenn Sie dieses Argument nicht angeben, wird der Standardwert (True) verwendet.

206 Datensatz aktivieren

Alternativ zur Methode FindRecord können Sie auch die Methode GoToRecord wie in Listing 238 einsetzen, um einen bestimmten Datensatz, im Beispiel den letzten Datensatz der Tabelle, zu aktivieren.

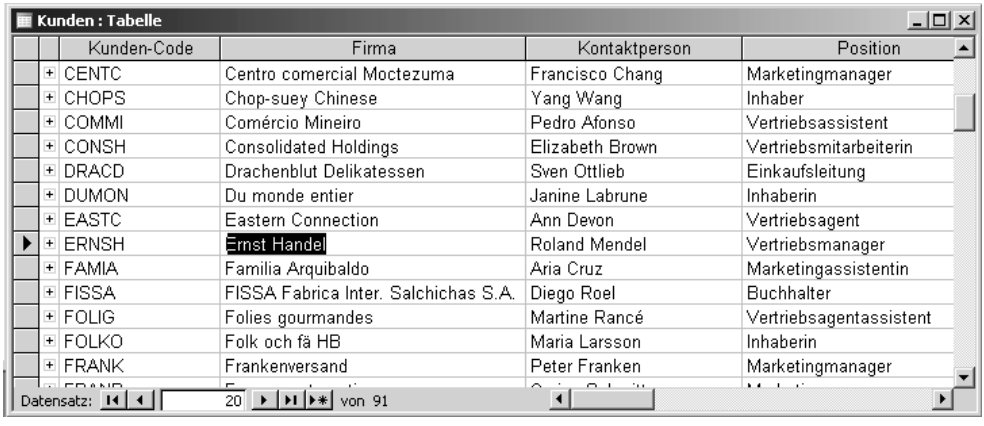


Abbildung 122: Tabelle öffnen und Datensatz vor einstellen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleÖffnenUndSatzAktivierenVar2()
   On Error GoTo fehler

   DoCmd.OpenTable "Kunden", acViewNormal
   DoCmd.GoToRecord acDataTable, "Kunden", acLast
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 238: Den letzten Datensatz aktivieren

Die Methode `GoToRecord` hat dabei folgende Syntax:

```
GoToRecord(Objekttyp, Objektname, Datensatz, Offset)
```

Im Argument `Objekttyp` müssen Sie angeben, um welchen Objekttyp es sich handelt. Es stehen Ihnen hierfür folgende Konstanten zur Verfügung:

- `acActiveDataObject`: Bei dieser Konstanten handelt es sich um die Standardeinstellung, die vorgibt, dass das aktive Objekt verwendet wird. In unserem Beispiel ist hiermit die Tabelle gemeint.
- `acDataForm`: Die Methode `GoToRecord` wird in einem Formular eingesetzt.
- `acDataFunction`: Die Methode wird anhand einer Funktion verwendet.

- acDataQuery: Hier kommt die Methode bei einer Abfrage zum Einsatz.
- acDataServerView: Einsatz der Methode GoToRecord unter der Serversicht.
- acDataStoredProcedure: Die Methode wird anhand einer gespeicherten Prozedur ausgeführt.
- AcDatTable: Die Methode wird anhand einer Tabelle ausgeführt.

Beim Argument Objektname müssen Sie den Namen der Objekte angeben. Auf unser Beispiel bezogen ist das die Tabelle Artikel.

Im Argument Datensatz bestimmen Sie über eine Konstante, welcher Datensatz der aktuelle Datensatz sein soll, d.h., auf welchem Datensatz der Mauszeiger stehen soll. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- acFirst: Der erste Datensatz wird zum aktuellen Datensatz.
- acGoTo: Mit dieser Konstante können Sie über einen numerischen Wert festlegen, auf welchen Datensatz positioniert werden soll.
- acLast: Der letzte Datensatz in der Tabelle wird zum aktuell ausgewählten.
- acNewRec: Damit machen Sie den neu eingefügten Datensatz zum aktuellen Datensatz.
- acNext: Hierbei handelt es sich um die Standardeinstellung von Access, die vorgibt, dass der nächste Datensatz zum aktuellen Datensatz gemacht werden soll. Dabei müssen Sie wie schon bei der Konstanten acGoTo einen numerischen Wert angeben, um die genaue Position zu bestimmen.
- acPrevious: Bei dieser Konstanten wird der vorherige Satz zum aktuellen Datensatz gemacht. Auch hier muss noch ein numerischer Wert angegeben werden.

Beim letzten Argument Offset geben Sie einen numerischen Wert an. Dieser numerische Wert nennt die Anzahl der Datensätze, um die vorwärts oder rückwärts geblättert werden soll, wenn Sie für das Argument Datensatz acNext oder acPrevious angegeben haben, oder den Datensatz, zu dem Sie wechseln möchten, wenn Sie für das Argument Datensatz acGoTo angegeben haben.

207 Tabellen verknüpfen

Wenn Sie eine Tabelle aus einer anderen Datenbank mit Ihrer aktiven Datenbank verknüpfen möchten, dann können Sie das über den Menübefehl DATEI/EXTERNE DATEN/TABELLEN VERKNÜPFEN erledigen oder ein kleines Makro schreiben, das diese Aufgabe schneller durchführt.

Im folgenden Makro aus Listing 239 wird die Tabelle Personal aus der Datenbank *Nordwind.mdb* im Verzeichnis *C:\Eigene Dateien* in die aktuelle Datenbank verlinkt.

Abbildung 123: Der letzte Datensatz in der Tabelle wurde aktiviert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabellenLinken()
   On Error GoTo fehler
   DoCmd.TransferDatabase acLink, "Microsoft Access", _
      "C:\Eigene Dateien\Nordwind.mdb", acTable, "Personal", "PersonalGelinkt"
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 239: Tabelle verlinken

Geben Sie bei der Methode `TransferDatabase` im ersten Argument die Konstante `acLink` an. Damit wird festgelegt, dass eine Verknüpfung vorgenommen werden soll. Beim zweiten Argument handelt es sich eigentlich um einen Standardwert, der angibt, dass es sich in diesem Fall um eine Access-Datenbank handeln soll. Im dritten Argument geben Sie den Pfad sowie den Dateinamen der Datenbank an, die die Tabelle enthält, die Sie in Ihre Datenbank verknüpft einfügen möchten. Im nächsten Argument legen Sie mithilfe der Konstante `acTable` fest, dass es sich um eine zu verknüpfende Tabelle handelt. Das darauf folgende Argument bestimmt, wie der Name der Tabelle lautet. Das letzte Argument beinhaltet den Namen der Tabelle für Ihre Datenbank.

208 Tabellen im HTML/XML-Format speichern

Access bietet einige Möglichkeiten an, wie Sie Ihre Tabelle in Internetformate überführen können. Dabei können Sie Ihre Tabellen sowohl in das HTML-Format wie auch in das XML-Format transportieren.

Tabelle ins HTML-Format übertragen

Im folgenden Beispiel aus Listing 240 wird die Tabelle `Artikel` im HTML-Format gespeichert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=================================================

Sub TabelleInHTMLkonvertieren()
   DoCmd.OpenTable "Artikel", acViewNormal
   DoCmd.OutputTo acOutputTable, "Artikel", acFormatHTML, "Artikel.htm"
   DoCmd.Close acTable, "Artikel"
End Sub
```

Listing 240: Tabelle ins HTML-Format exportieren

Mit der Methode `OutPutTo` können Sie eine Tabelle im HTML-Format speichern, indem Sie die Konstante `acFormatHTML` verwenden.

Alle Tabellen der Datenbank ins HTML-Format übertragen

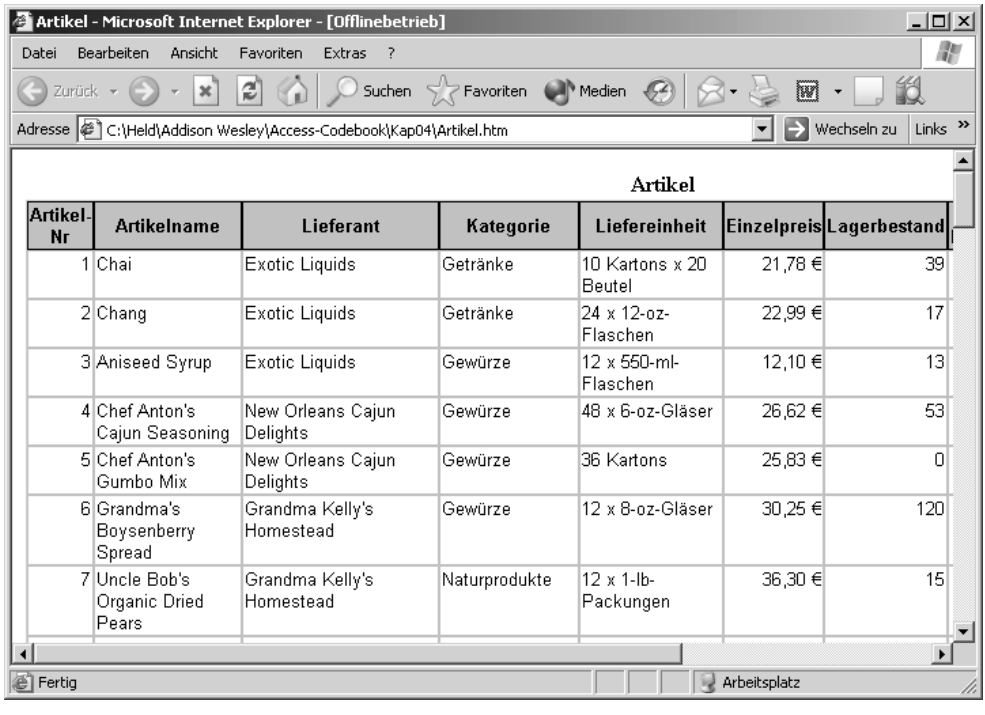
Wenn Sie das Makro aus Listing 240 noch ein wenig erweitern, dann können Sie alle Ihre Tabellen im HTML-Format abspeichern. Dazu starten Sie das Makro aus Listing 241:

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=================================================

Sub AlleTabellenKonvertieren()
   Dim obj As AccessObject
   Dim dbs As Object

   Set dbs = Application.CurrentData
   DoCmd.Echo False

   For Each obj In dbs.AllTables
      DoCmd.OpenTable obj.Name, acViewNormal
      DoCmd.OutputTo acOutputTable, obj.Name, acFormatHTML, obj.Name & ".htm"
      DoCmd.Close acTable, obj.Name
```

Listing 241: Alle Tabellen ins HTML-Format umwandeln

```
    Next obj

    DoCmd.Echo True
End Sub
```

Listing 241: Alle Tabellen ins HTML-Format umwandeln (Forts.)



Artikel

Artikel-Nr	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbestand
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €	39
2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	22,99 €	17
3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	12,10 €	13
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	26,62 €	53
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €	0
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	30,25 €	120
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	36,30 €	15

Abbildung 124: Die Tabelle wurde im HTML-Format gespeichert.

Setzen Sie das Auflistungsobjekt `AllTables` ein, um an alle Tabellen heranzukommen, die in der aktuellen Datenbank angelegt wurden. Öffnen Sie danach jede einzelne Tabelle, wandeln Sie diese um und schließen Sie die Tabelle wieder. Damit der Bildschirm während dieser Aktion ruhig bleibt, setzen Sie die Methode `Echo` auf den Wert `False`.

Tabelle ins XML-Format übertragen

Auch für das neuere Internetformat XML existiert eine Schnittstelle in Access, über die Sie Ihre Tabellen in das XML-Format überführen können.

Im folgenden Beispiel aus Listing 242 wird die Tabelle `Artikel` im XML-Format gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleInXMLKonvertieren()
   Application.ExportXML acExportTable, "Artikel", "Artikel.xml", "artikel.xsd"
End Sub
```

Listing 242: Tabelle ins XML-Format umwandeln

Mithilfe der Methode `ExportXML` können Sie Access-Objekte in das XML-Format umwandeln. Diese Methode hat eine ganze Reihe optionaler Argumente, die Sie in der Online-Hilfe von Access-VBA nachlesen können.

209 Komplexere Aufgaben mit ADO/DAO und SQL

Um Aufgaben durchzuführen, die über das Öffnen, Filtern und Schließen von Tabellen hinausgehen, arbeiten Sie über die Zugriffsmethode ADO/DAO und den Einsatz von SQL-Statements, die Sie im Zusammenspiel gerade für Access-Tabellen gewinnbringend einsetzen können. Auf den folgenden Seiten können Sie spezielle Lösungen nachschlagen, die Sie mehr oder weniger 1:1 für Ihre eigenen Anwendungen übernehmen können.

210 Datenbank öffnen und schließen (ADO)

Zu Beginn der »Lösungsstaffel« sehen Sie im Makro aus Listing 243, wie eine Datenbank über ADO-Methoden geöffnet und wieder geschlossen wird.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DBUndTabelleStarten()
   Dim Conn As ADODB.Connection

   On Error GoTo fehler
   Set Conn = New ADODB.Connection
   Conn.Provider = "Microsoft.Jet.OLEDB.4.0"
   Conn.ConnectionString = _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=C:\Eigene Dateien\nordwind.mdb;"
   Conn.Open
```

Listing 243: Datenbank öffnen und wieder schließen

```
    'Weitere Aktionen

    Conn.Close
    Set Conn = Nothing
    Exit Sub


    fehler:
       Msgbox Err.Number & " " & Err.Description
End Sub
```

Listing 243: Datenbank öffnen und wieder schließen (Forts.)

Das Objekt `Connection` stellt die Verbindung zur Datenbank bzw. zum Daten-Provider her. Die Eigenschaft `Provider` gibt den Namen des Providers für die Verbindung bekannt. Über die Eigenschaft `ConnectionString` können Sie die Datenquelle der Verbindung zuweisen. Über den Einsatz der Methode `Open` stellen Sie die Verbindung des Providers zur Datenquelle her. Mithilfe der Methode `Close` beenden Sie die Verbindung zum Provider.

211 Tabelle öffnen und Dump ziehen (ADO)

Beim folgenden Beispiel aus Listing 244 wird die Tabelle `Artikel` geöffnet und der komplette Inhalt als »Dump« im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub TabelleÖffnenUndDumpErstellen()
   Dim DBS As New ADODB.Recordset

   On Error GoTo fehler
   DBS.Open "Artikel", CurrentProject.Connection
   Debug.Print DBS.GetString
   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 244: Tabelle öffnen und auslesen

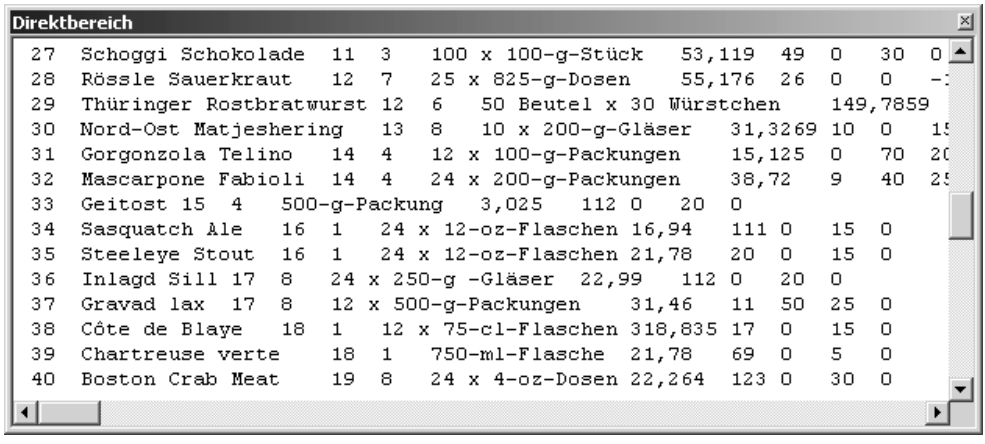


Abbildung 125: Inhalt 1:1 ausgeben

Über den Einsatz der Methode `Open` können Sie eine Tabelle öffnen. Die Syntax dieser Methode lautet:

```
recordset.Open Source, ActiveConnection, CursorType, LockType, Options
```

Argument	Beschreibung
`Source`	Optional. Gibt den Namen der Tabelle an, die Sie öffnen möchten. Auch der Einsatz einer SQL-Anweisung ist hier möglich.
`ActiveConnection`	Optional. Gibt die Verbindung an.
`CursorType`	Optional. Bestimmt die Art des Cursors. Unter anderem wird dadurch der Zugriff auf Ihre Daten festgelegt. Dabei stehen Ihnen folgende Konstanten zur Verfügung: `adOpenForwardOnly` (Voreinstellung): Öffnet einen Vorwärtscursor. Mithilfe dieses Cursors können Sie nur nach vorne blättern. `adOpenKeyset`: Öffnet einen Cursor vom Typ »Schlüsselgruppen«. Dieser Cursor ist vergleichbar mit dem dynamischen Cursor. Jedoch werden bei diesem Cursor Änderungen in der Tabelle, die von anderen Anwendern durchgeführt werden, nicht angezeigt. `adOpenDynamic`: Öffnet einen dynamischen Cursor. Damit haben Sie die Möglichkeit, Tabelleneinträge anzuzeigen, zu ändern und zu löschen. Alle Änderungen werden regelmäßig aktualisiert und angezeigt. `adOpenStatic`: Öffnet einen statischen Cursor. Bei diesem Cursor können die Daten nur angezeigt, jedoch nicht geändert werden. Die Datenansicht ist als Momentaufnahme des Zustands zu verstehen, der zum Zeitpunkt des Öffnens der Tabelle vorgelegen hat.

Tabelle 90: Die Argumente der Methode Open

Argument	Beschreibung
LockType	Optional. Bestimmt, welches Sperrverfahren der Provider beim Öffnen der Tabelle einsetzen soll. Dabei stehen Ihnen folgende Konstanten zur Verfügung: adLockReadOnly: Bei dieser Standardeinstellung können Sie die Daten in der Tabelle nicht ändern. adLockPessimistic: Der Datensatz wird vollständig gesperrt. Dabei wird das erfolgreiche Bearbeiten der Datensätze sichergestellt, indem der Provider Datensätze in der Datenquelle sofort beim Bearbeiten sperrt. adLockOptimistic: Diese Einstellung sorgt dafür, dass die Tabelle teilweise gesperrt wird, d.h., ein Datensatz wird nur dann gesperrt, wenn Sie die Update-Methode aufrufen. adLockBatchOptimistic: Diese Einstellung lässt eine teilweise Stapelaktualisierung zu.
Options	Optional. Legt fest, wie der Provider die Daten auswerten soll.

Tabelle 90: Die Argumente der Methode Open (Forts.)

Über die Methode `GetString` können Sie alle Daten aus der Tabelle abrufen, sofern Sie keine weiteren Argumente angeben. Über die Methode `Close` schließen Sie die Tabelle.

212 Eine bestimmte Anzahl von Sätzen extrahieren

Sollen beispielsweise nur die ersten zehn Datensätze der Tabelle als »Dump« in das Direktfenster der Entwicklungsumgebung geschrieben werden, dann starten Sie das Makro aus Listing 245.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub TabelleÖffnenUndDumpErstellen2()
   Dim DBS As New ADODB.Recordset
   On Error GoTo fehler
   DBS.Open "Artikel", CurrentProject.Connection
   Debug.Print DBS.GetString(NumRows:=10, ColumnDelimeter:=";"))
   DBS.Close
   Exit Sub

   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 245: Nur eine Auswahl von Sätzen schreiben

Über das Argument `NumRows` der Methode `GetString` können Sie festlegen, wie viele Datensätze Sie ausgeben möchten. Wenn dieses Argument nicht angegeben wird oder mehr Datensätze angegeben werden, als die Tabelle überhaupt enthält, dann werden standardmäßig alle Datensätze der Tabelle ausgegeben.

Im Argument `ColumnDelimeter` können Sie das Trennzeichen zwischen den einzelnen Feldern angeben. Wird dieses Argument nicht angegeben, dann wird standardmäßig ein Tabulatorschritt zwischen den einzelnen Feldern eingefügt.

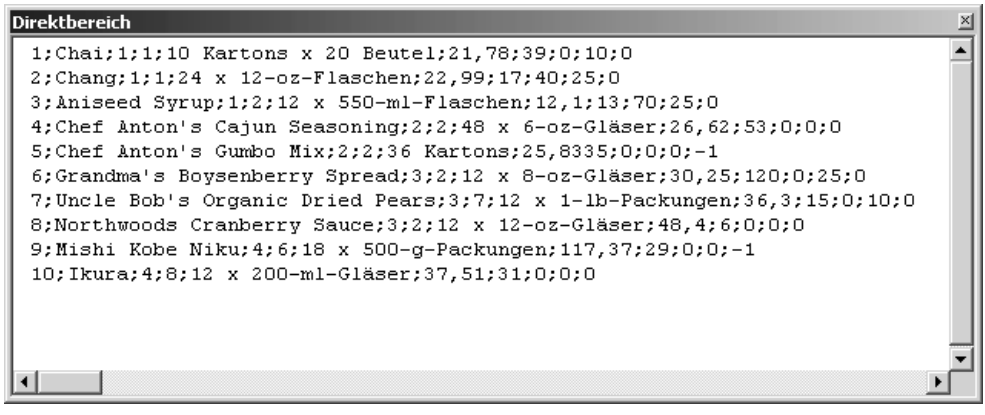


Abbildung 126: Ein anderes Trennzeichen sowie eine Vorauswahl an Sätzen treffen

213 Nur bestimmte Felder einer Tabelle ausgeben (ADO)

Die Abfrage aus Listing 244 lässt sich noch weiter einschränken, indem Sie mithilfe der SQL-Anweisung `Select` nur bestimmte Felder ausgeben.

Im Beispiel aus Listing 246 werden im Direktfenster der Entwicklungsumgebung nur die Datenfelder LAND, FIRMA und KONTAKTPERSON ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub NurBestimmteDatenExtrahieren()
   Dim DBS As New ADODB.Recordset

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler
   DBS.Open "SELECT Land, Firma, Kontaktperson " & _
      " FROM Kunden ORDER BY Land", _
      CurrentProject.Connection
```

Listing 246: Nur bestimmte Felder ausgeben

```
    Debug.Print DBS.GetString
    DBS.Close
    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 246: Nur bestimmte Felder ausgeben (Forts.)

Geben Sie bei der SQL-Anweisung die gewünschten Informationen an, jeweils getrennt durch ein Komma. Über die SQL-Klausel `ORDER BY` können Sie eine Sortierreihenfolge festlegen. Durch diese Vorauswahl greift die Methode `GetSting` nur noch die so angegebenen Felder ab.

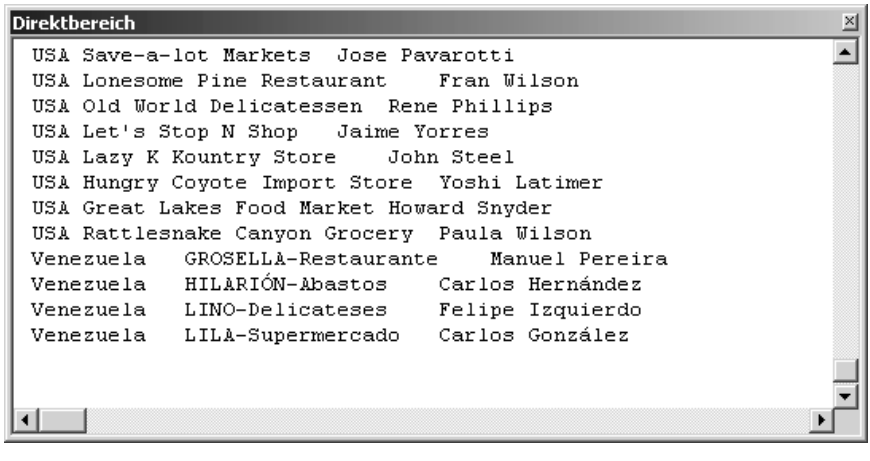


Abbildung 127: Es werden nur die Felder Land, Format und Kontaktperson ausgegeben.

214 Tabelle öffnen und Daten ausgeben (DAO)

Sollen Daten aus einer Tabelle über DAO ausgelesen werden, dann greifen Sie auf die DAO-Objekte `Database` und `RecordSet` zu.

Beim folgenden Beispiel aus Listing 247 wird die Tabelle `Kunden` geöffnet und einige Felder werden in das Direktfenster der Entwicklungsumgebung eingelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub TabelleStartenUndAuslesenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset
   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Kunden", dbOpenTable, dbReadOnly)

   rs.MoveFirst

   Do Until rs.EOF
      Debug.Print rs.Fields("Firma").Value
      Debug.Print rs.Fields("Kontaktperson").Value
      Debug.Print rs.Fields("Straße").Value
      Debug.Print rs.Fields("Ort").Value
      Debug.Print rs.Fields("Land").Value & vbLf
      rs.MoveNext
   Loop

   rs.Close
   Set rs = Nothing
   Set DBS = Nothing
   Exit Sub

   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 247: Tabelle öffnen und auslesen (DAO)

Deklarieren Sie im ersten Schritt zwei Objekte: eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Hinweis

Befindet sich die Tabelle nicht in der aktuellen Datenbank, dann setzen Sie beispielsweise folgende Codezeile ein:

```
Set DBS = OpenDataBase("c:\Eigene Dateien\DB1.mdb")
```

Führen Sie danach die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen. Wenden Sie die Methode `MoveFirst` an, um den ersten Satz in der Tabelle zu aktivieren.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze der Tabelle so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu und geben sie über den Befehl `Debug.Print` im Direktfenster der Entwicklungsumgebung aus. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf, indem Sie den Objektvariablen das Schlüsselwort `Nothing` zuweisen. Dadurch erfolgt die Trennung der Objektvariablen vom Objekt.

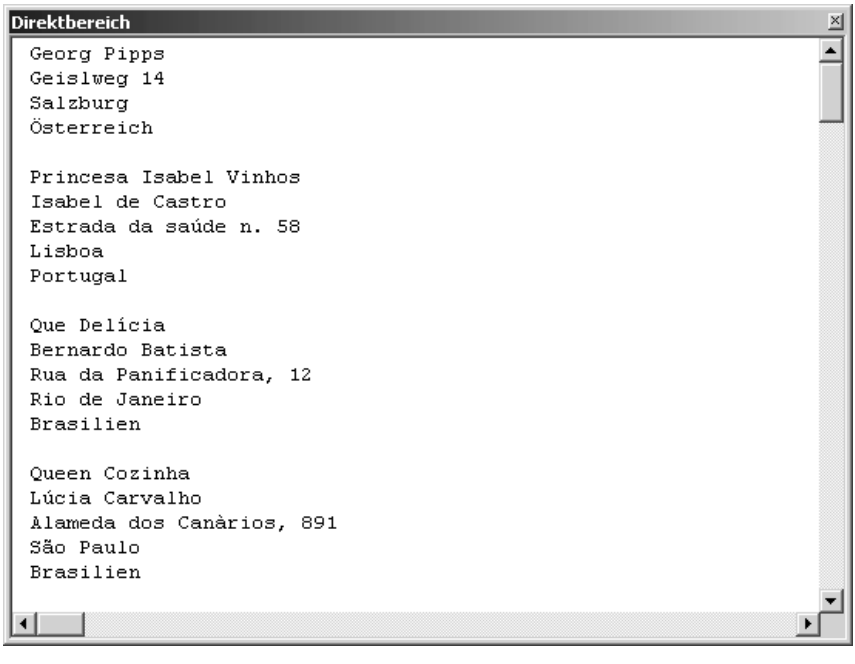


Abbildung 128: Alle Datensätze wurden ausgelesen.

215 Datensätze in Datenfeld einlesen (ADO)

Sollen nur bestimmte Spalten und Zeilen einer Tabelle ausgelesen werden, dann wenden Sie die Methode `GetRows` an. Über diese Methode können Sie mehrere Datensätze einer Tabelle in ein Datenfeld einlesen. Die Syntax dieser Methode lautet:

```
array = recordset.GetRows(Rows, Start, Fields )
```

Argument	Beschreibung
`recordset`	Gibt ein Recordset-Objekt zurück.
`Rows`	Optional. Legt die Anzahl an Datensätzen fest, die eingelesen werden sollen.

Tabelle 91: Die Argumente der Methode GetRows

Argument	Beschreibung
Start	Optional. Legt die Startposition fest, ab der Datensätze eingelesen werden sollen.
Fields	Legt die Spalten fest, die eingelesen werden sollen.

Tabelle 91: Die Argumente der Methode GetRows (Forts.)

Im folgenden Beispiel aus Listing 248 werden aus der Tabelle `Artikel` die ersten zehn Datensätze mit den ersten drei Spalteninhalten ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatenInFeldEinlesen()
   Dim dbs As New ADODB.Recordset
   Dim varDat As Variant
   Dim intZeilen As Integer
   Dim IntSpalten As Integer
   Dim IntGesamtZeilen As Integer

   IntGesamtZeilen = 10

   On Error GoTo fehler
   dbs.Open "Artikel", CurrentProject.Connection
   varDat = dbs.GetRows(IntGesamtZeilen)

   For intZeilen = 0 To IntGesamtZeilen - 1
      For IntSpalten = 0 To 2
         Debug.Print varDat(IntSpalten, intZeilen) & " ";
      Next IntSpalten
      Debug.Print vbLf
   Next intZeilen
   dbs.Close
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 248: Datensätze in ein Datenfeld einlesen (ADO)

Nach dem Öffnen der Tabelle `Artikel` übergeben Sie der Methode `GetRows` die Anzahl der gewünschten Zeilen. Danach durchlaufen Sie zwei Schleifen, in denen Sie neben den ersten zehn Zeilen auch noch die ersten drei Spalten in ein zweidimensionales Datenfeld einlesen.

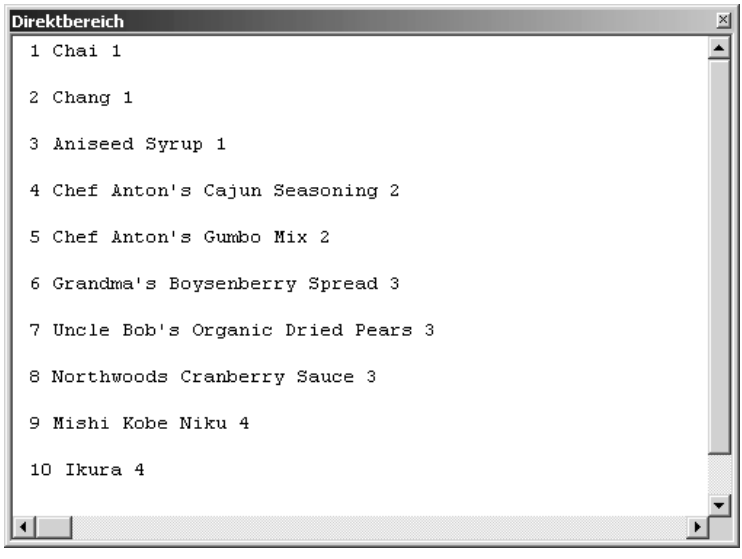


Abbildung 129: Die ersten zehn Datensätze einlesen

216 Datensätze in Datenfeld einlesen (DAO)

Die Methode `GetRows` ist ebenso auch in DAO einsetzbar. Im Makro aus Listing 249 werden die ersten zehn Datensätze sowie die ersten drei Spalten der Tabelle `Artikel` ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatenInFeldEinlesenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset
   Dim varDat As Variant
   Dim intZeilen As Integer
   Dim IntSpalten As Integer
   Dim IntGesamtZeilen As Integer

   IntGesamtZeilen = 10

   On Error GoTo fehler
   Set DBS = CurrentDb()
   Set rs = DBS.OpenRecordset("Artikel", dbOpenSnapshot)

   varDat = rs.GetRows(IntGesamtZeilen)
```

Listing 249: Datensätze in ein Datenfeld einlesen (DAO)

```
    For intZeilen = 0 To IntGesamtZeilen - 1
       For IntSpalten = 0 To 2
          Debug.Print varDat(IntSpalten, intZeilen) & " ";
       Next IntSpalten
       Debug.Print vbLf
    Next intZeilen

    rs.Close
    Set DBS = Nothing
    Set rs = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
 End Sub
```

Listing 249: Datensätze in ein Datenfeld einlesen (DAO) (Forts.)

Beim Makro aus Listing 249 wurde für die Bekanntgabe der aktuellen Datenbank die Kurzform

```
Set DBS = CurrentDb()
```

gewählt. Diese Kurzform ist gleichbedeutend mit der etwas längeren Form

```
Set DBS =  Application.CurrentDb.
```

217 SQL-Statements absetzen (ADO)

Über die Methode `Execute` können Sie eine SQL-Anweisung oder eine beliebige andere Abfrage absetzen. Die Syntax dieser Methode lautet:

```
connection.Execute CommandText, RecordsAffected, Options
```

Argument	Beschreibung
`Connection`	Optional. Gibt ein `Connection`-Objekt zurück.
`CommandText`	Erforderlich. Enthält den Text der SQL-Abfrage, der gespeicherten Prozedur, der Abfrage oder der URL.
`RecordsAffected`	Optional. Gibt die Anzahl der Datensätze zurück, die davon betroffen sind.
`Options`	Optional. Gibt an, wie der Provider das Argument `CommandText` behandeln soll.

Tabelle 92: Die Argumente der Methode Execute

Im folgenden Beispiel aus Listing 250 werden alle Datensätze aus der Tabelle `Kunden` extrahiert, bei denen die Kunden aus `Deutschland` kommen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=================================================

Sub DatensätzeAuslesen()
   Dim dbs As ADODB.Recordset

   Set dbs = _
      CurrentProject.Connection.Execute _
      ("SELECT * FROM Kunden Where Land ='Deutschland'" _
      , , adCmdText)

   Do Until dbs.EOF
      Debug.Print dbs!Firma & vbTab & _
         dbs!Kontaktperson & _
         vbTab & dbs!Land
      dbs.MoveNext
   Loop

   dbs.Close
   Set dbs = Nothing
End Sub
```

Listing 250: Datensätze auslesen

Beim Makro aus Listing 250 werden mithilfe der SQL-Anweisung `SELECT` die Felder `Firma`, `Kontaktperson` und `Land` aus der Tabelle `Kunden` ausgelesen. Über das Schlüsselwort `WHERE` kann dabei die Auswahl eingeschränkt werden. Die SQL-Anweisung wird an die Methode `Execute` übergeben. Das Ergebnis steht danach im `Recordset`-Objekt `dbs` zur Verfügung und kann ausgelesen werden. Das Auslesen erfolgt über eine `Do Until`-Schleife, bei der über die Eigenschaft `EOF` der letzte gefundene Satz im `Recordset`-Objekt festgestellt werden kann. Über die Methode `MoveNext` wird jeweils der nächste Datensatz im `Recordset`-Objekt ermittelt und über den Befehl `Debug.Print` im Direktfenster der Entwicklungsumgebung ausgegeben.

218 Unikatsliste aus Tabelle erstellen (ADO)

Um eine Unikatsliste aus einer Tabelle zu erstellen, bei der jeder Eintrag nur einmal vorkommt, setzen Sie den SQL-Befehl `SELECT DISTINCT` ein.

Im folgenden Beispiel aus Listing 251 wird die Tabelle `Kunden` ausgewertet. Ziel ist es, eine Liste zu erstellen, in der alle Länder der Tabelle nur einmal vorkommen.

Abbildung 130: Einige Datensätze auslesen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub KeineDoppelten()
   Dim DBS As ADODB.Recordset

   On Error Goto fehler
   Set DBS = _
      CurrentProject.Connection.Execute _
      ("SELECT DISTINCT LAND FROM KUNDEN" _
      , , adCmdText)

   Do Until DBS.EOF
      Debug.Print DBS!Land
      DBS.MoveNext
   Loop

   DBS.Close
   Set DBS = Nothing
   Exit Sub

   fehler:
      Msgbox Err.Number & " " & err.Description
End Sub
```

Listing 251: Eine Unikatsliste erstellen (ADO)

Über die Anweisung `DISTINCT`, welche Sie vor dem Feld angeben, wird sichergestellt, dass im Ergebnis jeder Satz nur einmal auftaucht.

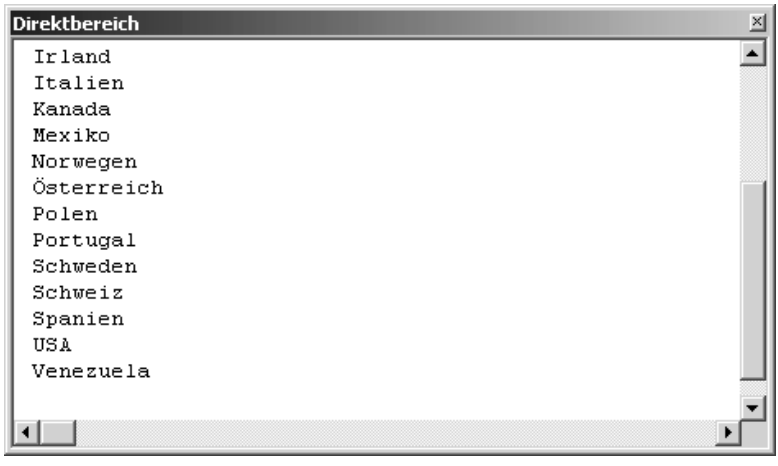


Abbildung 131: Die Unikatsländerliste wurde erstellt.

Im Makro aus Listing 251 wurde sichergestellt, dass jedes Land aus der Tabelle `Kunden` nur einmal im Direktfenster ausgegeben wird. Dieses Makro lässt sich insofern erweitern, als die Einmaligkeit für das Land sowie die Stadt gelten soll. Im Makro aus Listing 252 werden also nur Datensätze ausgegeben, die in Land und Stadt keine Dopplungen aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub KeineDoppelten2()
   Dim DBS As ADODB.Recordset

   Set DBS = _
      CurrentProject.Connection.Execute _
      ("SELECT DISTINCT LAND, ORT FROM KUNDEN" _
      , , adCmdText)

   Do Until DBS.EOF
      Debug.Print DBS!Land & vbTab & DBS!Ort
      DBS.MoveNext
   Loop

   DBS.Close
   Set DBS = Nothing
End Sub
```

Listing 252: Keine doppelten Länder und Orte (ADO)

Die Anweisung `DISTINCT` lässt sich erweitern, indem alle folgenden Felder, getrennt durch ein Komma, aufgeführt werden.

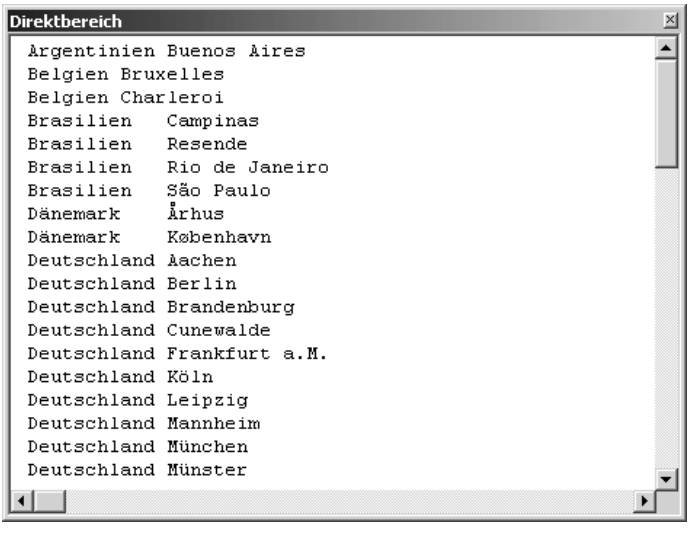


Abbildung 132: Jeder Ort in einem Land wird nur einmal genannt.

219 Unikatsliste aus Tabelle erstellen (DAO)

Um eine Unikatsliste über ein DAO-Makro zu erstellen, verwenden Sie ebenso das SQL-Schlüsselwort `Distinct`.

Im folgenden Makro aus Listing 253 wird die Tabelle `Kunden` angezapft und alle Ortsnamen werden ausgelesen. Dabei darf kein Ortsnamen doppelt im Ergebnis vorkommen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub KeineDoppeltenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("SELECT DISTINCT ORT FROM KUNDEN", dbOpenSnapshot)

   With rs
      Do Until .EOF
         Debug.Print .Fields("Ort")
         .MoveNext
      Loop
```

Listing 253: Eine Unikatsliste erstellen (DAO)

```
      .Close
   End With

   Set DBS = Nothing
   Set rs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 253: Eine Unikatsliste erstellen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte – eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Führen Sie die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen. Über die SQL-Anweisung `"SELECT DISTINCT ORT FROM KUNDEN"` wird jeder Ort nur einmal aus der Tabelle KUNDEN extrahiert..

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze, die jetzt als Ergebnis im `RecordSet`-Objekt `rs` zur Verfügung stehen, so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz im `RecordSet`-Objekt ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

220 Datensätze sortieren (ADO)

Um Datensätze sortiert auszugeben, setzen Sie die SQL-Klausel `ORDER BY` ein. Über das Schlüsselwort `ASC` (aufsteigend) bzw. `DESC` (absteigend) können Sie die Sortierreihenfolge festlegen. Wenn Sie aufsteigend sortieren möchten, dann können Sie dieses Schlüsselwort auch weglassen.

Im Beispiel aus Listing 254 wird die Tabelle `Artikel` geöffnet und es werden alle Datensätze ermittelt, die einen Einzelpreis von mehr als 20 Euro aufweisen. Die Ausgabe der Datensätze erfolgt aufsteigend nach dem Einzelpreis.

Abbildung 133: Jeder Ort taucht nur einmal auf.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeSortiertAuslesen()
   Dim DBS As ADODB.Recordset

   On Error Goto fehler
   Set DBS = CurrentProject.Connection.Execute _
      ("SELECT * FROM Artikel Where Einzelpreis > 20 ORDER BY Einzelpreis ASC" _
      , , adCmdText)

   Do Until DBS.EOF
      Debug.Print DBS!Artikelname & vbLf & _
         DBS!Einzelpreis & vbLf & _
         DBS!Lagerbestand & vbLf
      DBS.MoveNext
   Loop

   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 254: Datensätze sortiert ausgeben

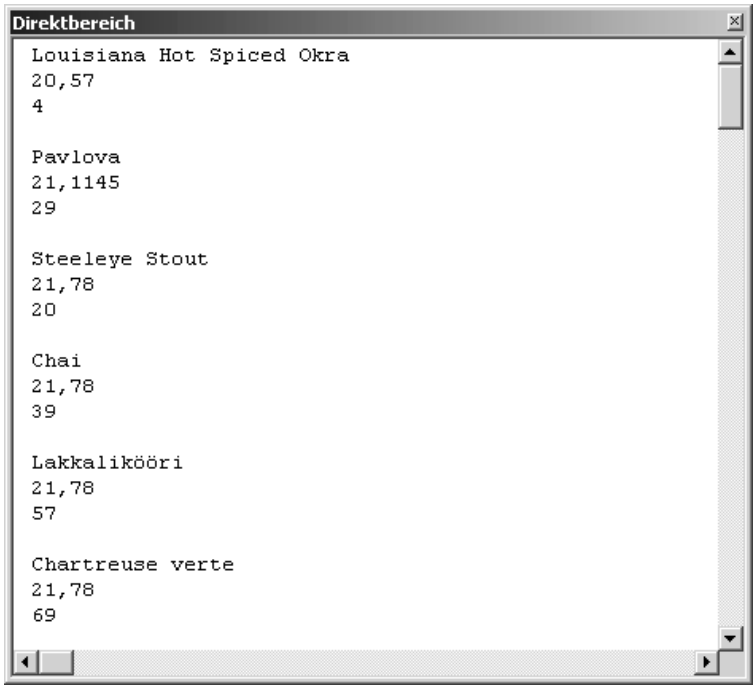


Abbildung 134: Der billigste Artikel steht ganz oben.

Eine alternative Möglichkeit, eine Liste zu sortieren, bietet der Einsatz der Eigenschaft `Sort`. Dabei haben Sie die Möglichkeit, Datensätze in einer Tabelle zu sortieren.

Im folgenden Beispiel aus Listing 255 wird die Tabelle `Artikel` geöffnet und die Datensätze werden nach dem Artikelnamen sortiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeSortieren()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .CursorLocation = adUseClient
```

Listing 255: Datensätze sortieren über die Eigenschaft Sort

```
        .Open "Artikel", Conn, adOpenKeyset, adLockOptimistic
        .Sort = "Artikelname ASC"

        Do While Not .EOF
           Debug.Print .Fields("Artikel-Nr.").Value & _
           " " & .Fields("Artikelname").Value
           .MoveNext
        Loop

        .Close
    End With

    Conn.Close
    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 255: Datensätze sortieren über die Eigenschaft Sort (Forts.)

Ganz wichtig ist beim Sortiervorgang, dass Sie die Eigenschaft `CursorLocation` mit der Konstanten `adUseClient` angeben. Damit wird der clientbasierte Cursor verwendet, der unter anderem den Sortiervorgang von Datensätzen unterstützt. Wenden Sie danach die Eigenschaft `Sort` an und geben Sie vor, nach welchen Kriterien sortiert werden soll. Haben Sie mehrere Sortierkriterien zur Auswahl, dann geben Sie diese entsprechend der Sortierreihenfolge getrennt durch Kommata ein. Bei der Sortierreihenfolge selbst können Sie entweder `ASC` für aufsteigende Sortierung oder `DESC` für absteigende Sortierung angeben. Dabei erfassen Sie nach dem Feldnamen ein Leerzeichen und hängen die gewünschte Sortierkonstante an. Wenn Sie diese Konstante nicht angeben, wird standardmäßig absteigend sortiert.

221 Datensätze sortieren (DAO)

Auch beim Anwenden von DAO müssen Sie auf SQL-Anweisungen nicht verzichten. Um Datensätze sortiert auszugeben, setzen Sie die SQL-Klausel `ORDER BY` ein. Über das Schlüsselwort `ASC` (aufsteigend) bzw. `DESC` (absteigend) können Sie die Sortierreihenfolge festlegen. Wenn Sie aufsteigend sortieren möchten, dann können Sie dieses Schlüsselwort auch weglassen.

Im Beispiel aus Listing 256 wird die Tabelle `Artikel` sortiert nach `Einzelpreis` im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=================================================

Sub DatensätzeSortiertAuslesenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset _
      ("SELECT * FROM Artikel Where Einzelpreis>20 ORDER BY Einzelpreis ASC", _
      dbOpenSnapshot)

   With rs
      Do Until .EOF
         Debug.Print "Artikelname: " & .Fields("Artikelname").Value & vbLf & _
            "Einzelpreis: " & .Fields("Einzelpreis").Value & vbLf & _
            "Lagerbestand: " & .Fields("Lagerbestand").Value & vbLf
         .MoveNext
      Loop

      .Close
   End With

   Set DBS = Nothing
   Set rs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 256: Datensätze sortieren und ausgeben (DAO)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Anschließend verwenden Sie die Methode `OpenRecordSet`, um die Tabelle zu öffnen. Dabei erfassen Sie direkt schon beim Öffnen der Tabelle die SQL-Anweisung, die die Filterung und Sortierung herstellt. Als Ergebnis stehen nun alle Datensätze der Tabelle im `RecordSet`-Objekt `rs` und können dort abgegriffen werden.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze des `RecordSet` so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz des `RecordSet`-Objekts ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

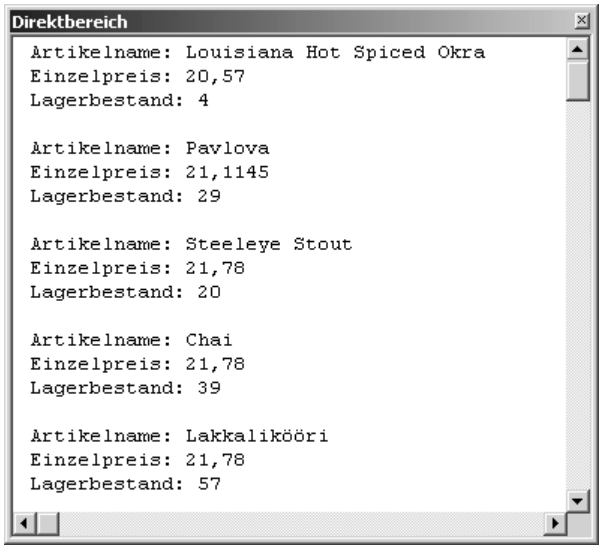


Abbildung 135: Der billigste Artikel steht ganz oben.

222 Tabellen zusammenfassen (ADO)

Mit der SQL-Anweisung `UNION` können Sie mehrere Tabellen zusammenfassen.

Das nachfolgende Makro aus Listing 257 liest aus den beiden Tabellen `KUNDEN` und `LIEFERANTEN` die Informationen `ORT`, `FIRMA` und `KONTAKTPERSON` heraus.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MehrereTabellenAbfragen()
   Dim DBS As New ADODB.Recordset

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler
   With DBS
     .Open "SELECT Ort, Firma, Kontaktperson " & _
        "FROM Kunden UNION SELECT Ort, Firma, " & _
        "Kontaktperson FROM " & _
        "Lieferanten ORDER BY Ort, Firma", _
        CurrentProject.Connection

     Do Until .EOF
        Debug.Print "Ort: " & .Fields("Ort").Value & _
           "  Firma: " & .Fields("Firma").Value & _
```

Listing 257: Mehrere Tabellen abfragen

```
            " Kontaktperson: " & .Fields("Kontaktperson")
          DBS.MoveNext
       Loop
       .Close
    End With
    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 257: Mehrere Tabellen abfragen (Forts.)

Über das Schlüsselwort `UNION` verbinden Sie die zwei Tabellen `Kunden` und `Lieferanten`.

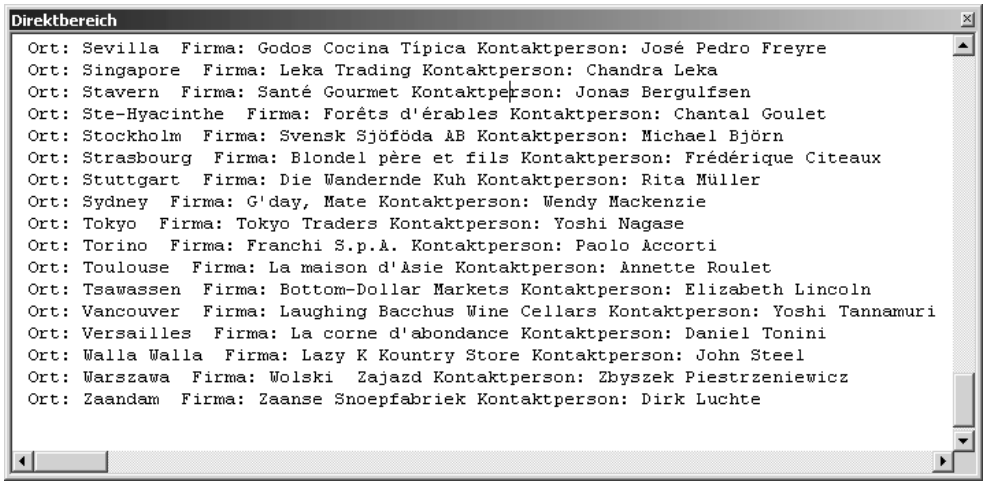


Abbildung 136: Das Ergebnis enthält Daten aus zwei Tabellen.

223 Verknüpfte Tabellen abfragen (ADO)

Wenn Sie beispielsweise im Menü EXTRAS den Befehl BEZIEHUNGEN auswählen, dann werden Sie feststellen, dass einige Tabellen wie die Tabelle `Artikel` sowie die `Lieferanten` über die `Lieferanten_Nr` verknüpft sind.

Wenn Sie nun beispielsweise die Tabelle `Artikel` auslesen, dann bekommen Sie als Lieferant nicht den Namen des Lieferanten, sondern eben eine Lieferantennummer. Mit dieser Nummer kann dann in der Tabelle `Lieferanten` der Name des Lieferanten ermittelt werden.

Sollen daher Informationen aus beiden Tabellen in eine Auswertung einfließen, dann müssen beide Tabellen in die `SELECT`-Anweisung integriert werden. Sehen Sie sich das Makro aus Listing 258 einmal etwas näher an.

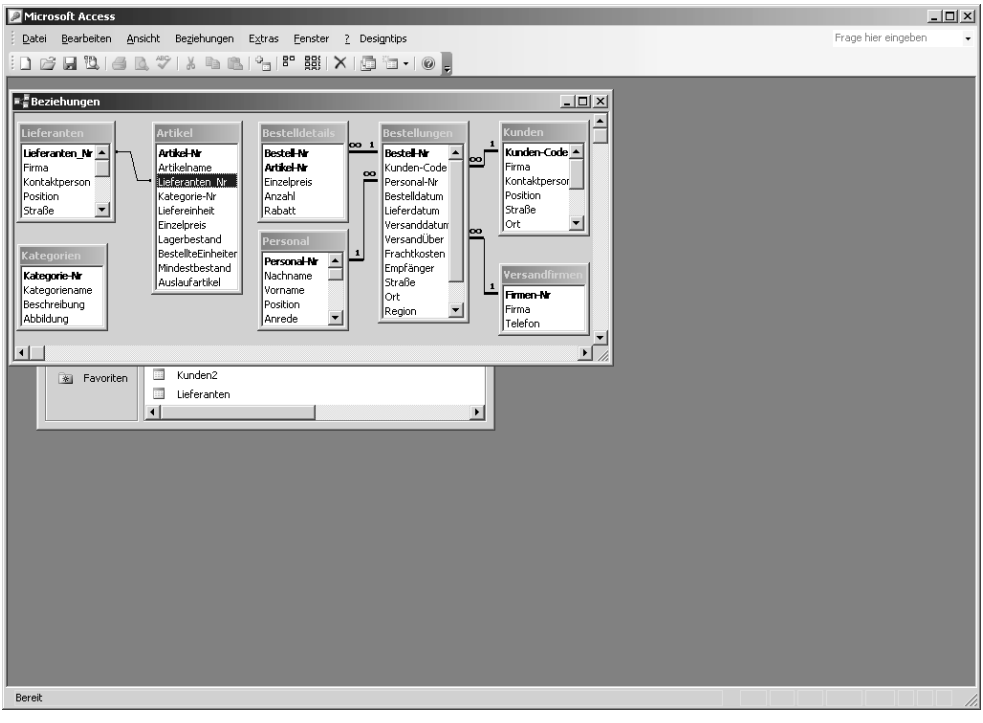


Abbildung 137: Die Tabelle Artikel ist mit der Artikel Lieferanten verknüpft.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MehrereVerknüpfteTabellenAbfragen()
   Dim dbs As New ADODB.Recordset

   Set dbs = New ADODB.Recordset
   On Error GoTo fehler

   With dbs
     .Open "SELECT Artikel.Artikelname, Lieferanten.Firma, " & _
        "Lieferanten.Kontaktperson " & _
        "FROM Artikel, Lieferanten " & _
        "WHERE Artikel.Lieferanten_Nr=Lieferanten.Lieferanten_Nr", _
        CurrentProject.Connection

     Do Until .EOF
        Debug.Print "Ort: " & .Fields("Artikelname").Value
        Debug.Print "Firma: " & .Fields("Firma").Value
```

Listing 258: Verknüpfte Tabellen abfragen (ADO)

```
         Debug.Print "Kontaktperson: " & .Fields("Kontaktperson") & vbLf
         dbs.MoveNext
      Loop

      .Close
   End With

   Set dbs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 258: Verknüpfte Tabellen abfragen (ADO) (Forts.)

Geben Sie bei der `SELECT`-Anweisung vor jedem Feld auch den Namen der Tabelle an, aus der das Feld stammt. Nach der `FROM`-Klausel geben Sie die Namen beider Tabellen an, aus denen Sie Daten ziehen möchten. In der `WHERE`-Klausel geben Sie den Namen des verknüpften Felds an, über das die beiden Tabellen miteinander verbunden sind.

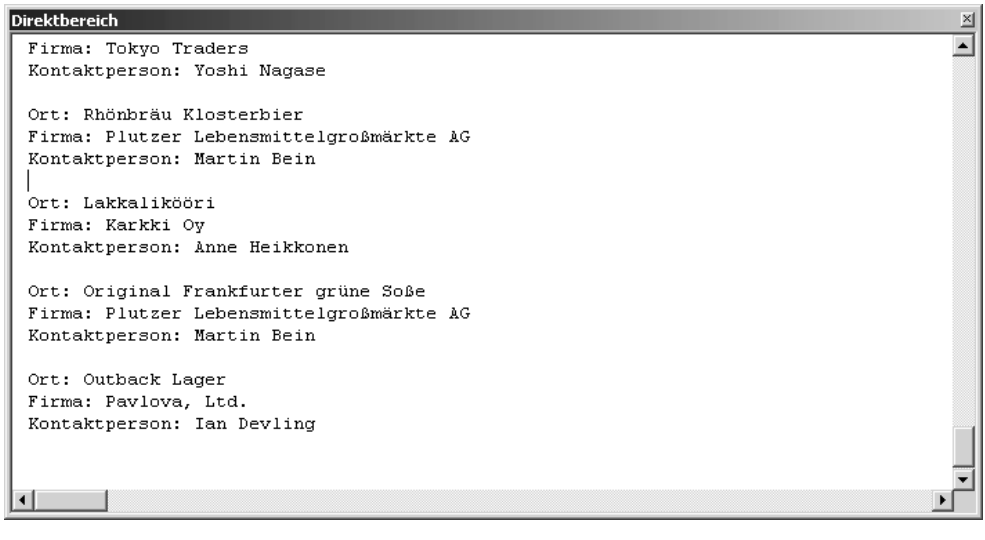


Abbildung 138: Der Lieferantenname wird korrekt angezeigt.

224 Verknüpfte Tabellen abfragen (DAO)

Die gleiche Aufgabe aus Listing 258 wird jetzt in DAO über den Einsatz des Makros aus Listing 259 gelöst.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub MehrereVerknüpfteTabellenAbfragenDAO()
   Dim dbs As DAO.Database
   Dim rs As DAO.Recordset

   Set dbs = Application.CurrentDb
   On Error GoTo fehler

   Set rs = dbs.OpenRecordset _
      ("SELECT Artikel.Artikelname, Lieferanten.Firma, " & _
      "Lieferanten.Kontaktperson FROM Artikel, Lieferanten " & _
      "WHERE Artikel.Lieferanten_Nr=Lieferanten.Lieferanten_Nr", dbOpenSnapshot)

   With rs
      Do Until .EOF
         Debug.Print "Ort: " & .Fields("Artikelname").Value
         Debug.Print "Firma: " & .Fields("Firma").Value
         Debug.Print "Kontaktperson: " & .Fields("Kontaktperson") & vbLf
         .MoveNext
      Loop

      .Close
   End With

   Set dbs = Nothing
   Set rs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 259: Verknüpfte Tabellen abfragen (DAO)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Dann führen Sie die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen. Geben Sie beim Öffnen der Tabelle in der `SELECT`-Anweisung vor jedem Feld auch den Namen der Tabelle an, aus der das Feld stammt. Nach der `FROM`-Klausel geben Sie die Namen beider Tabellen an, aus denen Sie Daten ziehen möchten. In der `WHERE`-Klausel geben Sie den Namen des verknüpften Felds an, über das die beiden Tabellen miteinander verbunden sind.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze der Tabelle so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung

`Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

225 Minimal- und Maximalwerte ermitteln (ADO)

Mit der Anweisung `TOP` können Sie die wertmäßig größten/kleinsten Feldinhalte einer Tabelle ermitteln.

Je nachdem, wie die Tabelle sortiert ist bzw. sortiert wird, werden bei der SQL-Anweisung `TOP` entweder die größten oder die kleinsten Werte angezeigt.

Negativ Top 5 nach Preis

Im folgenden Beispiel aus Listing 260 werden die fünf billigsten Artikel aus dem Sortiment der Tabelle `Artikel` ermittelt und angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWerteFinden()
   Dim DBS As New ADODB.Recordset
   Dim strSQL As String

   strSQL = "SELECT TOP 5 Einzelpreis, Artikelname, " & _
      "Lagerbestand FROM Artikel ORDER BY Einzelpreis"

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler

   With DBS
      .Open strSQL, CurrentProject.Connection

      Do Until DBS.EOF
         Debug.Print "Artikelname:  " & _
            .Fields("Artikelname").Value & vbLf & _
            "Lagerbestand: "; .Fields("Lagerbestand").Value & _
            vbLf & "Einzelpreis:  " & _
            .Fields("Einzelpreis").Value & vbLf
         DBS.MoveNext
      Loop

      .Close
   End With

   Set DBS = Nothing
```

Listing 260: Minimalwerte aus einer Tabelle ermitteln

```
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 260: Minimalwerte aus einer Tabelle ermitteln (Forts.)

Über die SQL-Anweisung `SELECT TOP 5` erhalten Sie die fünf billigsten Artikel im Sortiment, wenn die Sortierreihenfolge aufsteigend eingestellt ist.

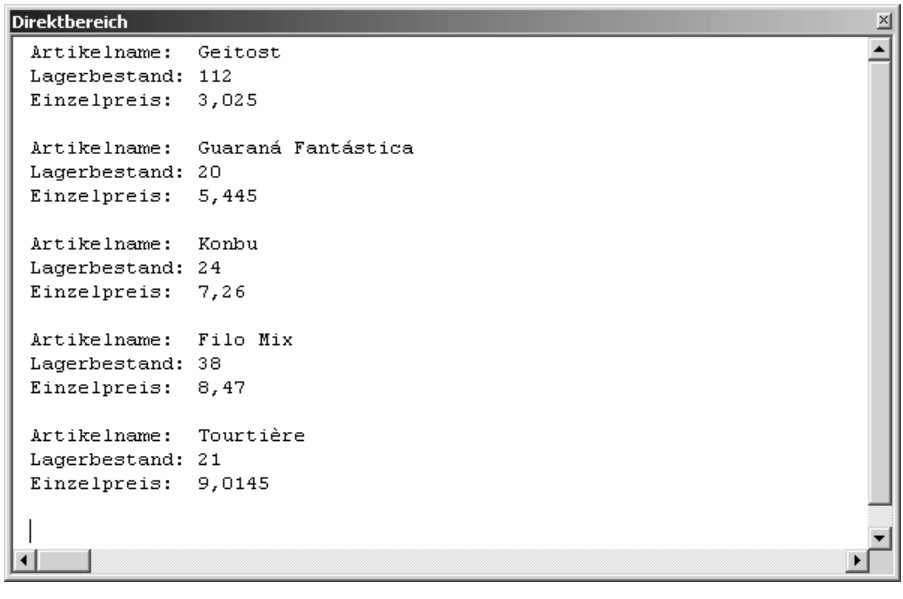


Abbildung 139: Die fünf billigsten Artikel im Sortiment

Top 3 nach Lagerbestand

Sollten hingegen die drei Artikel mit dem größten Lagerbestand ermittelt werden, dann starten Sie das Makro aus Listing 261.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MaxWerteFinden()
   Dim DBS As New ADODB.Recordset
   Dim strSQL As String
```

Listing 261: Die Maximalwerte aus einer Tabelle ermitteln

```
    strSQL = "SELECT TOP 3 Einzelpreis, Artikelname, " & _
       "Lagerbestand FROM Artikel ORDER BY Lagerbestand DESC"

    Set DBS = New ADODB.Recordset
    On Error GoTo fehler

    With DBS
       .Open strSQL, CurrentProject.Connection

       Do Until DBS.EOF
          Debug.Print "Artikelname:  " & _
             .Fields("Artikelname").Value & vbLf & _
             "Lagerbestand: "; .Fields("Lagerbestand").Value & _
             vbLf & "Einzelpreis:  " & _
             .Fields("Einzelpreis").Value & vbLf
          DBS.MoveNext
       Loop

       .Close
    End With

    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 261: Die Maximalwerte aus einer Tabelle ermitteln (Forts.)

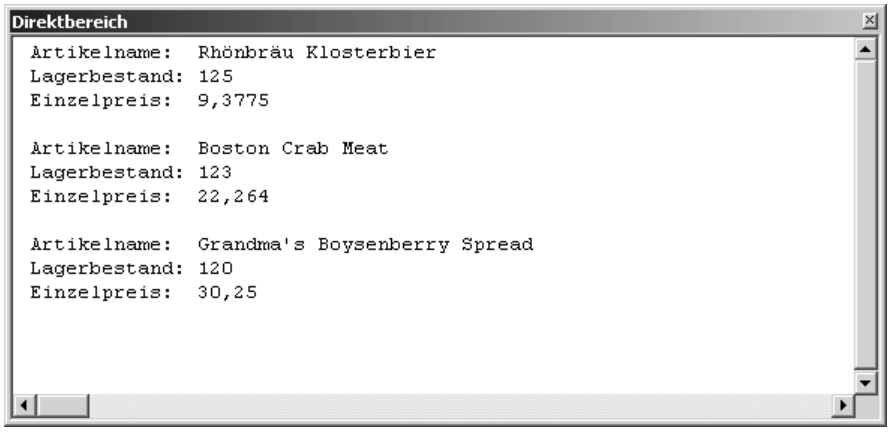


Abbildung 140: Die Ladenhüter im Sortiment

Eine weitere Möglichkeit, um bestimmte Datensätze in einer Tabelle aufzuspüren, ist der Einsatz von Funktionen wie `Max`, `Min` oder `Avg` (Mittelwert), die Sie auch in SQL-Anweisungen einsetzen können.

Der billigste Artikel

Im folgenden Makro aus Listing 262 wird der billigste Artikel der Tabelle `Artikel` wertmäßig abgefragt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWertFinden()
   Dim DBS As New ADODB.Recordset
   Dim strSQL As String

   strSQL = _
      "SELECT Min(Einzelpreis) FROM Artikel"

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler

   With DBS
      .Open strSQL, CurrentProject.Connection
      .MoveFirst
      MsgBox "Der niedigste Preis ist " & .Fields(0).Value
      .Close
   End With

   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 262: Der niedrigste Preis wird ermittelt.

Fügen Sie die Funktion `Min` gleich nach der Anweisung `SELECT` ein und setzen Sie das Feld in Klammern, aus welchem Sie den niedrigsten Wert ermitteln möchten. Nach der Suche wenden Sie die Methode `MoveFirst` an, um diesen Wert anzusteuern. Über die `Fields`-Auflistung können Sie diesen Wert ausgeben.

Ladenhüter ermitteln

Um den größten Ladenhüter im Sortiment zu ermitteln, starten Sie das Makro aus Listing 263.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=================================================

Sub MaxWertFinden()
   Dim DBS As New ADODB.Recordset
   Dim strSQL As String

   strSQL = "SELECT Max(Lagerbestand) FROM Artikel"

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler

   With DBS
      .Open strSQL, CurrentProject.Connection
      .MoveFirst
      MsgBox "Die höchste Lagermenge ist " & .Fields(0).Value
      .Close
   End With

   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 263: Den größten Ladenhüter ermitteln

Neben den Aggregatfunktionen `Min` und `Max` gibt es in Tabelle 93 weitere Funktionen, die Sie einsetzen können.

Funktion	Kurzbeschreibung
`Avg`	Mittelwertsberechnung
`Count`	Datensatzanzahl wird ermittelt
`First`	Meldet den ersten gefundenen Datensatz
`Last`	Meldet den letzten gefundenen Datensatz
`Max`	Ermittelt den größten Datensatz
`Min`	Ermittelt den kleinsten Datensatz
`StDev`	Errechnet die Standardabweichung
`Sum`	Bildet die Summe

Tabelle 93: Die verfügbaren Aggregatfunktionen

Lagerbestandsdurchschnitt errechnen

Im folgenden Beispiel aus Listing 264 werden der durchschnittliche Lagerbestand und der durchschnittliche Preis aller Artikel errechnet und im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub AVGWerteFinden()
   Dim DBS As New ADODB.Recordset
   Dim strSQL As String

   strSQL = _
      "SELECT AVG(Lagerbestand), AVG(Einzelpreis) FROM Artikel"

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler

   With DBS
      .Open strSQL, CurrentProject.Connection
      Debug.Print .GetString
      .Close
   End With

   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 264: Durchschnittsberechnungen durchführen (ADO)

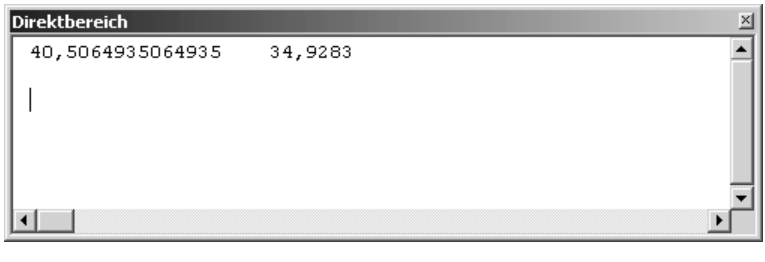


Abbildung 141: Der durchschnittliche Lagerbestand liegt so um die 40.

Eine weitere Möglichkeit zur Ermittlung von Extremwerten, bietet der Einsatz der »D-Funktionen«, die Sie in Tabelle 94 einsehen können.

Funktion	Kurzbeschreibung
DAvg	Mittelwertsberechnung.
DCount	Datensatzanzahl wird ermittelt.
DLookUp	Zeigt einen Wert eines Felds an, das sich nicht in der Datenquelle Ihres Formulars oder Berichts befindet.
DFirst	Meldet den ersten gefundenen Datensatz.
DLast	Meldet den letzten gefundenen Datensatz.
DMin	Ermittelt den kleinsten Datensatz.
DMAx	Ermittelt den größten Datensatz.
DstDev	Errechnet die Standardabweichung (Stichprobe).
DStDevP	Errechnet die Standardabweichung (Population).
DSum	Bildet die Summe.
DVar	Errechnet die Varianz (Stichprobe).
DVarP	Errechnet die Varianz (Population).

Tabelle 94: Die D-Funktionen im Überblick

Den ältesten Mitarbeiter ermitteln

Im folgenden Beispiel aus Listing 265 wird das Geburtsdatum des ältesten Mitarbeiters in der Tabelle `Personal` ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWertFindenVar2()
   MsgBox "Der älteste Mitarbeiter ist am : " & _
   DMin("Geburtsdatum", "Personal") & _
      " geboren!", vbInformation
End Sub
```

Listing 265: Das am weitesten zurückliegende Datum wird über die Funktion Min ermittelt.

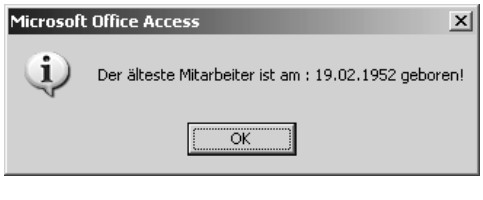


Abbildung 142: Auch Altersberechnungen sind möglich.

Soll das Alter in Jahren berechnet werden, dann erweitern Sie das Makro aus Listing 265 wie in Listing 236 angegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWertFindenVar2Erweitert()
   MsgBox DateDiff("YYYY", DMin("Geburtsdatum", "Personal"), Date)
End Sub
```

Listing 266: Rechnen ist auch mit DMin möglich.

226 Minimal- und Maximalwerte ermitteln (DAO)

Sie können auch beim Einsatz von DAO beispielsweise über das SQL-Schlüsselwort `TOP` die höchsten bzw. niedrigsten Werte aus einer Tabelle ziehen.

Beim folgenden Beispiel aus Listing 267 werden die fünf billigsten Artikel aus der Tabelle `Artikel` ermittelt und im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub MinWerteFindenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset _
      ("SELECT TOP 5 Einzelpreis, Artikelname, " & _
       "Lagerbestand FROM Artikel ORDER BY Einzelpreis", dbOpenSnapshot)

      With rs
         Do Until .EOF
            Debug.Print "Artikelname:  " & _
               .Fields("Artikelname").Value & vbLf & _
               "Lagerbestand: "; .Fields("Lagerbestand").Value & _
               vbLf & "Einzelpreis:  " & .Fields("Einzelpreis").Value & vbLf
            .MoveNext
         Loop

     .Close
```

Listing 267: Die fünf billigsten Artikel ermitteln (DAO)

```
    End With

    Set DBS = Nothing
    Set rs = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 267: Die fünf billigsten Artikel ermitteln (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Dann führen Sie die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen. Dabei wenden Sie eine SQL-Anweisung schon beim Öffnen der Tabelle an, indem Sie die SQL-Anweisung `("SELECT TOP 5 Einzelpreis, Artikelname, Lagerbestand FROM Artikel ORDER BY Einzelpreis")` ausführen. Danach stehen die fünf billigsten Artikel im `RecordSet`-Objekt `rs`. Dieses Objekt lesen Sie jetzt mithilfe einer `Do until`-Schleife aus und übertragen die einzelnen Datenfelder über die Auflistung `Fields` und die Anweisung `Debug.Print` in das Direktfenster der Entwicklungsumgebung. Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

227 Daten gruppieren (ADO)

Um komplexe Tabellen auszuwerten und dabei nach bestimmten Kriterien die Daten zu gruppieren, setzen Sie die Klausel `GROUP BY` ein.

Die folgenden beiden Beispiele gehen von der Tabelle `Gruppen` aus, in der Mitarbeiter, Gehälter und Arbeitsgruppenzugehörigkeit festgehalten werden. Den Aufbau dieser Tabelle entnehmen Sie Abbildung 143.

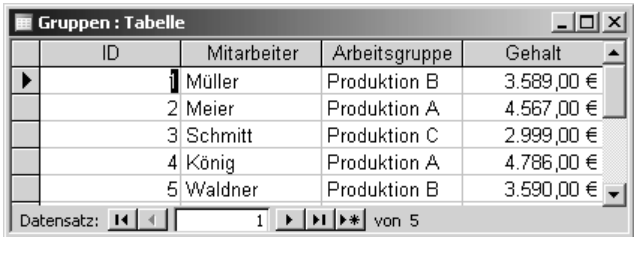


Abbildung 143: Die Ausgangstabelle fürs Gruppieren

Möchten Sie jetzt eine Auswertung haben, welche die Summen der Gehälter einer jeden Arbeitsgruppe ermittelt, dann starten Sie das Makro aus Listing 268.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeGruppieren()
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set DBS = CurrentProject.Connection.Execute _
      ("SELECT Arbeitsgruppe, sum(Gehalt) FROM Gruppen GROUP BY Arbeitsgruppe" _
      , , adCmdText)

   Debug.Print DBS.GetString
   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 268: Die Gehälter nach Arbeitsgruppe summieren und gruppieren

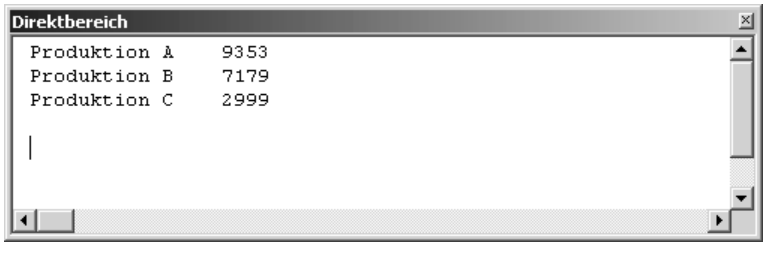


Abbildung 144: Die Gesamtgehälter pro Arbeitsgruppe

Soll das Ergebnis noch weiter eingeschränkt werden, beispielsweise, dass nur Arbeitsgruppen aufgelistet werden, die eine Gesamtsumme des Gehalts > 3000 haben, dann integrieren Sie die SQL-Klausel `HAVING`. Die Umsetzung dieser Klausel können Sie in Listing 269 sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================
```

Listing 269: Die Gruppierung einschränken

```
Sub DatensätzeGruppieren2()
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set DBS = _
      CurrentProject.Connection.Execute _
      ("SELECT Arbeitsgruppe, sum(Gehalt) FROM Gruppen " & _
      " GROUP BY Arbeitsgruppe HAVING SUM(Gehalt)> 3000" _
      , , adCmdText)

   Debug.Print DBS.GetString

   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 269: Die Gruppierung einschränken (Forts.)

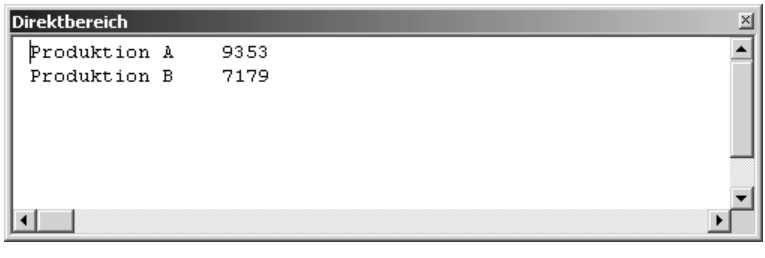


Abbildung 145: Es werden nur noch bestimmte Arbeitsgruppen angezeigt.

228 Datensätze in einer Tabelle finden (ADO)

Die Methode `Find` sucht in der Tabelle nach dem Datensatz, der den angegebenen Kriterien entspricht. Ist das Kriterium erfüllt, wird der gefundene Datensatz zum aktuellen Datensatz der Tabelle. Andernfalls wird der Zeiger auf das Ende des Tabelle festgelegt. Die Syntax dieser Methode lautet:

```
Find (criteria, SkipRows, searchDirection, start)
```

Argument	Beschreibung
`Criteria`	In diesem Argument müssen Sie angeben, was Sie konkret in der Tabelle suchen möchten.
`SkipRows`	Über dieses Argument können Sie den Abstand vom aktuellen Datensatz oder vom Argument `start` angeben.

Tabelle 95: Die Argumente der Methode Find

Argument	Beschreibung
SearchDirection	Dieses Argument gibt an, wie in der Tabelle gesucht werden soll. Dazu stehen Ihnen folgende Varianten zur Verfügung: adSearchForward: führt die Suche vorwärts durch. Die Suche ist am Ende der Tabelle beendet, sofern kein entsprechender Datensatz gefunden wurde. adSearchBackward: führt die Suche rückwärts durch. Die Suche hört am Anfang der Tabelle auf, sofern kein entsprechender Datensatz gefunden wurde.
Start	Bei diesem Argument können Sie einen numerischen Wert nennen, der angibt, bei welchem Datensatz die Suche in der Tabelle beginnen soll.

Tabelle 95: Die Argumente der Methode Find (Forts.)

Im folgenden Beispiel aus Listing 270 wird eine Artikelbezeichnung über eine Inputbox erfasst. Danach öffnen Sie die Tabelle ARTIKEL und durchsuchen Sie diese Tabelle. Wird der gesuchte Artikel gefunden, dann wird der entsprechende Artikel im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensatzFinden()
   Dim DBS As ADODB.Recordset
   Dim strEingabe As String

   strEingabe = _
   InputBox("Geben Sie den Artikel ein!")
   If strEingabe = "" Then Exit Sub
   strEingabe = "Artikelname='" & strEingabe & "'"

   On Error GoTo fehler
   Set DBS = New ADODB.Recordset

   With DBS
      .Open Source:="Artikel", _
         ActiveConnection:=CurrentProject.Connection, _
         CursorType:=adOpenKeyset, LockType:=adLockOptimistic

      .Find Criteria:=strEingabe, SearchDirection:=adSearchForward

      If Not .EOF Then
         Do While Not .EOF
```

Listing 270: Datensatz finden

```
            Debug.Print "Artikelname:" & _
               .Fields("Artikelname").Value
            Debug.Print "Einzelpreis :" & _
               .Fields("Einzelpreis").Value
            Debug.Print "Lagerbestand: " & _
               .Fields("Lagerbestand").Value
            .Find Criteria:=strEingabe, SkipRecords:=1
         Loop
      Else
         MsgBox "Datensatz nicht gefunden"
      End If
      .Close
   End With

   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 270: Datensatz finden (Forts.)

Im ersten Schritt fragen Sie über eine Eingabemaske die Artikelbezeichnung ab, nach der in der Tabelle `Artikel` gesucht werden soll. Bilden Sie den Suchbegriff im Anschluss daran. Achten Sie darauf, dass der Suchbegriff sowohl aus dem eigentlichen Suchbegriff als auch dem Feldnamen der Tabelle gebildet werden muss.

Öffnen Sie die Access-Datenbank mithilfe des Objekts `Connection`. Zum Öffnen einer Access-Datenbank brauchen Sie die Jet-Datenbank-Engine. Geben Sie daher mit der Eigenschaft `Provider` die Engine Microsoft Jet 4.0 OLE DB-Provider an.

Im Argument `ConnectionString` geben Sie den Namen der Datenbank an, die Sie öffnen möchten.

Definieren Sie daraufhin ein `RecordSet`-Objekt, das später den gesuchten Datensatz enthalten soll. Öffnen Sie jetzt über die Methode `Open` die Tabelle `Artikel`. Diese Methode hat folgende Syntax:

```
recordset.Open Source, ActiveConnection, CursorType, LockType, Options
```

Im Argument `Source` geben Sie den Namen der Tabelle an, die Sie öffnen möchten. Im Argument `ActiveConnection` verweisen Sie auf das `Connection`-Objekt `Conn`, welches Sie vorher angegeben haben.

Über das Argument `CursorType` bestimmen Sie die Art des Cursors. Unter anderem wird dadurch der Zugriff auf Ihre Daten festgelegt. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- `adOpenForwardOnly` (Voreinstellung): Öffnet einen Vorwärtscursor. Mithilfe dieses Cursors können Sie nur nach vorne blättern.

- `adOpenKeyset`: Öffnet einen Cursor vom Typ »Schlüsselgruppen«. Dieser Cursor ist vergleichbar mit dem dynamischen Cursor. Jedoch werden bei diesem Cursor Änderungen in der Tabelle, die von anderen Anwendern durchgeführt werden, nicht angezeigt.
- `adOpenDynamic`: Öffnet einen dynamischen Cursor. Damit haben Sie die Möglichkeit, Tabelleneinträge zu anzuzeigen, zu ändern und zu löschen. Alle Änderungen werden regelmäßig aktualisiert und angezeigt.
- `adOpenStatic`: Öffnet einen statischen Cursor. Bei diesem Cursor können die Daten nur angezeigt, jedoch nicht geändert werden. Die Datenansicht ist als Momentaufnahme des Zustands zu verstehen, der zum Zeitpunkt des Öffnens der Tabelle vorgelegen hat.

Über das Argument `LockType` bestimmen Sie, welches Sperrverfahren der Provider beim Öffnen der Tabelle einsetzen soll. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- `adLockReadOnly`: Bei dieser Standardeinstellung können Sie die Daten in der Tabelle nicht ändern.
- `adLockPessimistic`: Der Datensatz wird vollständig gesperrt. Dabei wird das erfolgreiche Bearbeiten der Datensätze sichergestellt, indem der Provider Datensätze in der Datenquelle sofort beim Bearbeiten sperrt.
- `adLockOptimistic`: Diese Einstellung sorgt dafür, dass die Tabelle teilweise gesperrt wird, d.h., ein Datensatz wird nur dann gesperrt, wenn Sie die `Update`-Methode aufrufen.
- `adLockBatchOptimistic`: Diese Einstellung lässt eine teilweise Stapelaktualisierung zu.

Im letzten Argument `Options` können Sie festlegen, wie der Provider die Daten auswerten soll.

Setzen Sie jetzt die Methode `Find` ein. Diese Methode sucht in der Tabelle nach dem Datensatz, der den angegebenen Kriterien entspricht. Ist das Kriterium erfüllt, wird der gefundene Datensatz zum aktuellen Datensatz der Tabelle. Andernfalls wird der Zeiger auf das Ende des Tabelle festgelegt.

In der ersten Verzweigung fragen Sie direkt nach, ob der gesuchte Satz gefunden wurde. Ist dies nicht der Fall, wird die Bedingung `EOF` (End Of File) erreicht. Wurde der gesuchte Datensatz gefunden, stehen alle Informationen hierüber in der Objektvariablen `DBS`.

Da es möglich ist, dass der gesuchte Artikel mehrfach in der Tabelle vorkommt, setzen Sie eine Schleife auf, die so lange durchlaufen wird, bis das Ende der Tabelle erreicht wird. Danach können Sie die einzelnen Werte über das Auflistungsobjekt `Fields` abrufen. In diesem Objekt stehen alle Feldinhalte, die Sie mithilfe der Eigenschaft `Value` ausgeben können. Achten Sie bei der Ausgabe darauf, dass Sie den Feldnamen, den Sie im Entwurfsmodus der Tabelle ansehen können, mit angeben.

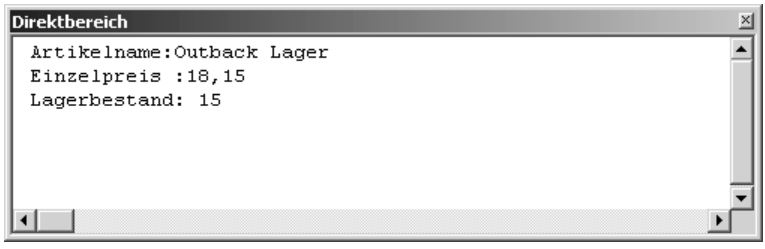


Abbildung 146: Der Datensatz Outback Lager wurde gefunden.

229 Datensätze in einer Tabelle finden (DAO)

Das Suchen von Daten unter DAO wird über die Methoden `FindFirst`, `FindNext`, `FindLast` und `FindPrevious` durchgeführt. Die Unterschiede der einzelnen Methoden sind in der Tabelle 96 dargestellt. Alle `Find`-Methoden können Sie für die `RecordSet`-Typen `SnapShot` und `DynaSet` verwenden.

Methode	Suche beginnt am	Bis zum
`FindFirst`	Beginn der Datensatzgruppe	Ende der Datensatzgruppe
`FindLast`	Ende der Datensatzgruppe	Beginn der Datensatzgruppe
`FindNext`	Aktuellen Datensatz	Ende der Datensatzgruppe
`FindPrevious`	Aktuellen Datensatz	Beginn der Datensatzgruppe

Tabelle 96: Die Unterschiede der Find-Methoden

Ein Suchtreffer

Beim Beispiel aus Listing 271 wird die Tabelle `Kunden` nach einer bestimmten Kontaktperson durchsucht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeFindenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Kunden", dbOpenDynaset)

   rs.FindFirst "Kontaktperson =  'Victoria Ashworth'"

   If rs.NoMatch = True Then
```

Listing 271: Datensatz finden über DAO

```
        MsgBox "Kontaktperson konnte nicht gefunden werden!"
    Else
        Debug.Print rs.Fields("Firma").Value
        Debug.Print rs.Fields("Kontaktperson").Value
        Debug.Print rs.Fields("Straße").Value
        Debug.Print rs.Fields("Ort").Value
        Debug.Print rs.Fields("Land").Value & vbLf
    End If

    rs.Close
    Set rs = Nothing
    Set DBS = Nothing
    Exit Sub


    fehler:
        MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 271: Datensatz finden über DAO (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte: eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Danach führen Sie die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen.

Wenden Sie die Methode `FindFirst` an, um den gewünschten Datensatz in der Tabelle zu finden. Übergeben Sie dazu den Namen der Kontaktperson. Über die Eigenschaft `NoMatch` kann festgestellt werden, ob überhaupt ein passender Datensatz gefunden werden konnte. Bei einer erfolgreichen Suche greifen Sie auf die einzelnen Felder über die Auflistung `Fields` zu und geben Sie über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

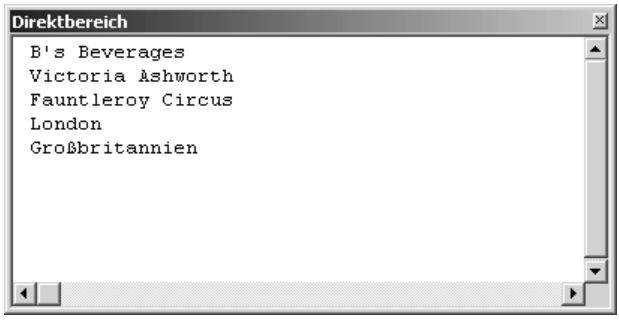


Abbildung 147: Die Kontaktperson konnte gefunden werden.

Mehrere Suchtreffer

Selbstverständlich gibt es auch Suchdurchläufe, bei denen mehrere Datensätze gefunden werden können. So wird beim folgenden Beispiel aus Listing 272 eine Suche in der Tabelle `Kunden` durchgeführt, bei der alle `Kunden` aus Deutschland gefunden und im Direktfenster der Entwicklungsumgebung ausgegeben werden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=================================================

Sub MehrereDatensätzeFindenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Kunden", dbOpenDynaset)

   rs.FindFirst "Land =  'Deutschland'"

   Do Until rs.NoMatch = True
      Debug.Print rs.Fields("Firma").Value
      Debug.Print rs.Fields("Kontaktperson").Value
      Debug.Print rs.Fields("Straße").Value
      Debug.Print rs.Fields("Ort").Value
      Debug.Print rs.Fields("Land").Value & vbLf
      rs.FindNext "Land = 'Deutschland'"
   Loop

   rs.Close
   Set rs = Nothing
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 272: Mehrere Datensätze suchen über DAO

Bei der Suche nach mehreren Datensätzen finden Sie den ersten Satz über die Methode `FindFirst`. Dabei übergeben Sie im Suchbegriff sowohl den Feldnamen als auch den gesuchten Feldinhalt. Danach setzen Sie eine Schleife auf, die so lange durchlaufen wird, bis kein Datensatz mehr gefunden werden kann. In diesem Fall meldet die Eigenschaft `NoMatch` den Wert `True`. Innerhalb der Schleife übertragen Sie die einzelnen Feldinhalte über die Auflistung `Fields` und geben diese im Direktfenster aus. Mit der Methode `FindNext` finden Sie jeweils den nächsten Datensatz, der den Suchkriterien entspricht.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

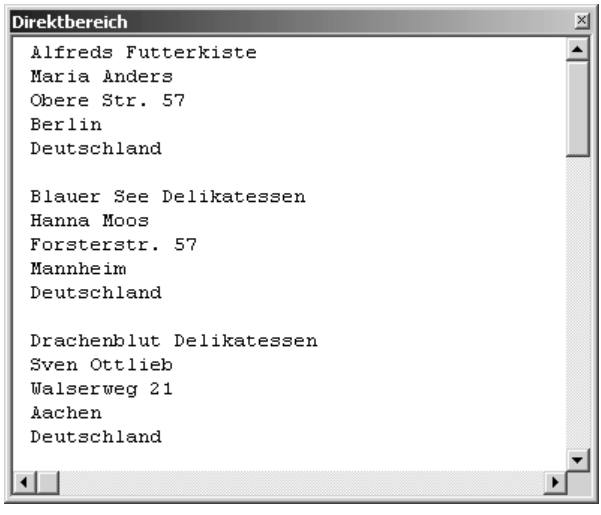


Abbildung 148: Alle Kunden aus Deutschland wurden gefunden.

Mehrere Bedingungen bei der Suche berücksichtigen

Wenn Sie nach mehreren Bedingungen suchen möchten, die zutreffen sollen, dann arbeiten Sie mit den Operatoren `AND` bzw. `OR`.

Im folgenden Beispiel aus Listing 273 werden in der Tabelle `Kunden` alle Kunden aus `Deutschland` gesucht, die in `Stuttgart` ansässig sind.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub MehrereDatensätzeFindenDAO2()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Kunden", dbOpenDynaset)

   rs.FindFirst "Land =  'Deutschland' AND Ort = 'Stuttgart'"

   Do Until rs.NoMatch = True
      Debug.Print rs.Fields("Firma").Value
      Debug.Print rs.Fields("Kontaktperson").Value
      Debug.Print rs.Fields("Straße").Value
```

Listing 273: Mehrere Suchkriterien berücksichtigen (DAO)

```
      Debug.Print rs.Fields("Ort").Value
      Debug.Print rs.Fields("Land").Value & vbLf
      rs.FindNext "Land =  'Deutschland' AND Ort = 'Stuttgart'"
   Loop

   rs.Close
   Set rs = Nothing
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 273: Mehrere Suchkriterien berücksichtigen (DAO) (Forts.)

Über die Methode `FindFirst` finden Sie den ersten Datensatz, der dem angegebenen Kriterium entspricht. Danach setzen Sie eine `Do Until`-Schleife auf, in der Sie die Suche so lange wiederholen, bis kein Datensatz mehr gefunden werden kann, der dem Suchkriterium aus der Methode `FindNext` entspricht. In diesem Fall meldet die Eigenschaft `NoMatch` den Wert `True`.

230 Datensätze verändern (ADO)

Mithilfe der Methode `Update` können Sie eine Änderung an einem Datensatz speichern.

Im folgenden Beispiel aus Listing 274 werden in der Tabelle `Gruppen` die Gehälter um 2,5% erhöht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub GehälterAnpassungVornehmen()
   Dim DBS As ADODB.Recordset

   Set DBS = New ADODB.Recordset
   On Error GoTo fehler
   DBS.Open "Gruppen", CurrentProject.Connection, _
      adOpenKeyset, adLockOptimistic

   Do Until DBS.EOF
      DBS("Gehalt") = DBS("Gehalt") * 1.025
      DBS.Update
      DBS.MoveNext
```

Listing 274: Gehaltsanpassung per Makro vornehmen

```
    Loop

    DBS.Close
    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 274: Gehaltsanpassung per Makro vornehmen (Forts.)

Da die Methoden zum Öffnen einer Tabelle bereits beschrieben wurden, gehen wir gleich mitten in die Schleife. Die Gehaltserhöhung führen Sie durch, indem Sie das Feld `Gehalt` mit dem Faktor 1.025 multiplizieren. Gleich danach wenden Sie die Methode `Update` an, um diese Änderung wirksam werden zu lassen. Mit der Methode `MoveNext` gehen Sie zum nächsten Datensatz der Tabelle. Diese Vorgehensweise führen Sie so lange durch, bis Sie am letzten Satz der Tabelle angelangt sind. Dann tritt die Schleifenbedingung `EOF` ein und die Schleife wird verlassen. Vergessen Sie dann nicht, die Tabelle über die Methode `Close` zu schließen und die Objektverweise wieder aufzuheben.

Eine zweite Variante, diese Aufgabe zu lösen, führt über die Methode `Execute`. Dabei wird die SQL-Anweisung dieser Methode übergeben. Sehen Sie sich dazu das Makro aus Listing 275 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub GehaltsanpassungExtDB()
   Dim Conn As ADODB.Connection
   Dim intZ As Long
   Dim strSQL As String

   strSQL = "UPDATE Gruppen SET Gehalt = Gehalt * 1.025"
   On Error GoTo fehler
   Set Conn = New ADODB.Connection

   With Conn
      .Provider = "Microsoft.Jet.OLEDB.4.0"
      .ConnectionString = _
      "C:\Eigene Dateien\Nordwind.mdb"
      .Open
   End With
```

Listing 275: Gehaltsanpassung in einer anderen Datenbank vornehmen (ADO)

```
    Conn.Execute CommandText:=strSQL, RecordsAffected:=intZ
    Debug.Print "Ersetzte Datensätze:  " & intZ

    Conn.Close
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 275: Gehaltsanpassung in einer anderen Datenbank vornehmen (ADO) (Forts.)

Formulieren Sie in der Variablen `strSQL` Ihre SQL-Anweisung. Nach dem Öffnen der Datenbank übergeben Sie den Inhalt der Variablen `strSQL` der Methode `Execute`. Die Konstante `RecordsAffected` gibt Ihnen die Anzahl der geänderten Sätze bekannt.

231 Datensätze verändern (DAO)

Sollen Datensätze über DAO verändert werden, dann sehen Sie sich das Beispiel aus Listing 276 an. In diesem Beispiel werden alle Artikel der Tabelle ARTIKEL3 im Feld EINZELPREIS um 10% erhöht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeÄndernDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Artikel3", dbOpenDynaset)

   With rs
      .MoveFirst

      Do Until rs.EOF
         rs.Edit
         rs.Fields("Einzelpreis").Value = rs.Fields("Einzelpreis").Value * 1.1
         rs.Update
         .MoveNext
      Loop
```

Listing 276: Datenfelder verändern mit DAO

```
        .Close
    End With

    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 276: Datenfelder verändern mit DAO (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Öffnen Sie im Anschluss mithilfe der Methode `OpenRecordSet` die Tabelle. Wenden Sie die Methode `MoveFirst` an, um den ersten Satz in der Tabelle zu aktivieren.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze der Tabelle so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife wenden Sie die Methode `Edit` an, um den Datensatz zu ändern. Über die Auflistung `Fields` greifen Sie auf das Feld `Einzelpreis` zu und weisen diesem Feld einen neuen Wert zu. Danach setzen Sie die Methode `Update` ein, um den Datensatz zu speichern. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

232 Währungsumrechnungen vornehmen (ADO)

Für Währungsumrechnungen stellt Ihnen Access eine eigene Funktion zur Verfügung. Mithilfe der Funktion `EuroConvert` können Sie viele gängige Währungen in den Euro umrechnen.

Die Funktion `EuroConvert` hat folgende Syntax:

```
EuroConvert(Zahl, Quellwährung, Zielwährung, [VolleGenauigkeit,
Trianguliergenauigkeit])
```

Das Argument `Zahl` enthält den Wert, den Sie umrechnen möchten.

Im Argument `Quellwährung` geben Sie die Währung an, in der Ihr Ursprungswert vorliegt. Hier müssen Sie den ISO-Code angeben, den Sie in der folgenden Tabelle sehen können.

Währung	ISO-Code	Berechnungsgenauigkeit	Anzeige
Belgische Franc	BEF	0	0
Luxemburgische Franc	LUF	0	0

Tabelle 97: Die Tabelle mit den ISO-Codes (Quelle: Online-Hilfe)

Währung	ISO-Code	Berechnungsgenauigkeit	Anzeige
Deutsche Mark	DEM	2	2
Spanische Peseten	ESP	0	0
Französische Franc	FRF	2	2
Irisches Pfund	IEP	2	2
Italienische Lire	ITL	0	0
Niederländische Gulden	NLG	2	2
Österreichische Schilling	ATS	2	2
Portugiesische Escudos	PTE	1	2
Finnische Mark	FIM	2	2
Euro	EUR	2	2

Tabelle 97: Die Tabelle mit den ISO-Codes (Quelle: Online-Hilfe) (Forts.)

Im Argument `Zielwährung` geben Sie den ISO-Code der gewünschten Endwährung an.

Beim Argument `VolleGenauigkeit` handelt es sich um ein optionales Argument. Setzen Sie dieses Argument auf den Wert `True`, wenn die währungsspezifischen Rundungsvorschriften ignoriert und der Umwandlungsfaktor mit sechs signifikanten Ziffern ohne anschließende Rundung verwendet werden sollen. Setzen Sie dieses Argument auf den Wert `False`, um die währungsspezifischen Rundungsvorschriften anzuwenden. Wird der Parameter nicht angegeben, ist der Standardwert `False`.

Im letzten Argument `Trianguliergenauigkeit` können Sie einen Wert größer als oder gleich 3 angeben. Damit wird die Anzahl der signifikanten Ziffern in der Berechnungsgenauigkeit bestimmt, die für den Euro-Zwischenwert verwendet wird, wenn zwischen zwei nationalen Währungen umgewandelt wird.

Um diese Funktion in einem praktischen Beispiel zu üben, wird die Tabelle `BestellungenAlt`, in der die Währungsangaben noch in DM erfasst sind, in die neue Währung Euro umgewandelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub EuroUmrechnen()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset
```

Listing 277: Währungsumrechnungen durchführen (ADO)

```
    On eror GoTo fehler
    Set Conn = CurrentProject.Connection
    Set DBS = New ADODB.Recordset

    DBS.Open "BestellungenAlt", Conn, adOpenKeyset, adLockOptimistic

    Do While Not DBS.EOF
       DBS![Einzelpreis] = EuroConvert(DBS![Einzelpreis], "DEM", "EUR")
       DBS.MoveNext
    Loop

    DBS.Close
    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 277: Währungsumrechnungen durchführen (ADO) (Forts.)

Öffnen Sie die Tabelle `BestellungenAlt` über die Methode `Open`. Danach durchlaufen Sie in einer Schleife alle darin enthaltenen Datensätze. Innerhalb der Schleife wenden Sie die Funktion `EuroConvert` an, um die Währungsumrechnung durchzuführen. Übergeben Sie dabei dieser Funktion das Datenfeld, welches umgerechnet werden soll, sowie die momentane und die gewünschte Währung.

233 Währungsumrechnung vornehmen (DAO)

Die Funktion `EuroConvert` kann auch mit DAO eingesetzt werden.

Im folgenden Makro aus Listing 278 werden alle Einzelpreise der Tabelle ARTIKEL1 von DM auf Euro umgestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub EuroUmrechnenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset
```

Listing 278: Währungsumstellung vornehmen (DAO)

```
    On Error GoTo fehler
    Set DBS = Application.CurrentDb
    Set rs = DBS.OpenRecordset("Artikel1", dbOpenTable, dbOpenDynaset)

    With rs
       .MoveFirst

       Do While Not .EOF
          .Edit
          .Fields("Einzelpreis").Value = _
          EuroConvert(.Fields("Einzelpreis").Value, "DEM", "EUR")
          .Update
          .MoveNext
       Loop

       .Close
    End With

    Set DBS = Nothing
    Set rs = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 278: Währungsumstellung vornehmen (DAO) (Forts.)

Der Änderungsprozess geht so vor sich, dass zuerst der entsprechende Satz über die Methode `Edit` in den Editiermodus versetzt wird. Danach wenden Sie die Funktion `EuroConvert` an, um die Umrechnung im Feld EINZELPREIS durchzuführen. Anschließend speichern Sie die Änderung, indem Sie die Methode `Update` einsetzen. In einer `Do until`-Schleife arbeiten Sie so alle Datensätze der Tabelle ab, bis die Eigenschaft `EOF` erfüllt ist.

234 Datensätze filtern (ADO)

Um Daten aus einer Tabelle zu filtern, setzen Sie die Eigenschaft `Filter` ein. Als Argumente können Sie dabei problemlos auch mehrere Bedingungen angeben, die Sie dann über die Operatoren `And` oder `Or` miteinander verbinden

Filtern nach bestimmten Artikeln

Im nächsten Beispiel aus Listing 269 werden in der Tabelle `Artikel` alle Artikel im Direktbereich ausgegeben, deren Lagerbestand > 20 und deren Einzelpreis >= 30 ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeFiltern()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open "Artikel", Conn, adOpenKeyset, adLockOptimistic
      .Filter = "Lagerbestand > 20 AND Einzelpreis >= 30"

      Do While Not .EOF
         Debug.Print .Fields("Artikelname").Value
         .MoveNext
      Loop

      .Close
   End With

   Conn.Close
   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 279: Datensätze über den Filter näher einschränken

Übergeben Sie der Eigenschaft `Filter` das Filterkriterium. Dabei arbeiten Sie bei numerischen Feldern mit den Zeichen =, < und >. In einer `Do While`-Schleife geben Sie danach die gefundenen Artikel im Direktfenster der Entwicklungsumgebung aus.

Filtern nach bestimmten Kunden

Im Makro aus Listing 280 werden aus der Tabelle `Kunden` alle Kunden gefiltert, die aus den `USA` oder `Kanada` kommen. Beachten Sie dabei die Schreibweise des Filterkriteriums.

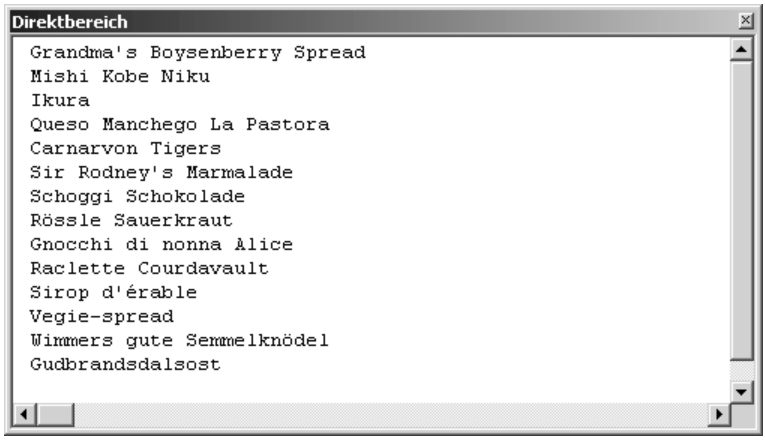


Abbildung 149: Alle Artikel, die dem Filterkriterium entsprechen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeFiltern2()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open "Kunden", Conn, adOpenKeyset, adLockOptimistic
      .Filter = "Land = 'USA' OR Land = 'Kanada'"

      Do While Not .EOF
         Debug.Print .Fields("Kontaktperson").Value
         Debug.Print .Fields("Ort").Value & vbLf
         .MoveNext
      Loop

      .Close
   End With

   Conn.Close
   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub
```

Listing 280: Das Filtern von alphanumerischen Feldern (ADO)

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 280: Das Filtern von alphanumerischen Feldern (ADO) (Forts.)

Bei alphanumerischen Feldern müssen Sie darauf achten, dass Sie die Filterkriterien in Apostrophe setzen.

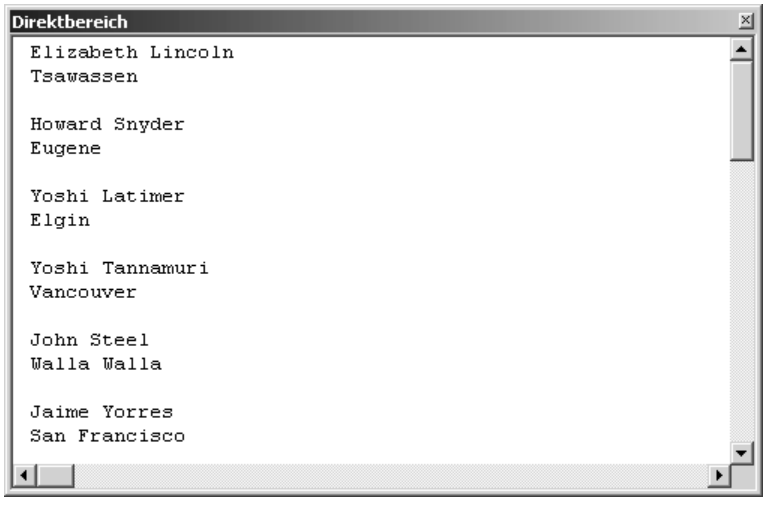


Abbildung 150: Kunden aus den USA oder Kanada

235 Datensätze filtern (DAO)

Soll eine Filterung von Daten aus einer Tabelle über den Einsatz von DAO durchgeführt werden, dann können Sie auch hier auf SQL-Anweisungen zugreifen.

Im Beispiel aus Listing 281 werden in der Tabelle `Artikel` alle Datensätze gefiltert, die einen Einzelpreis von über 30 Euro und einen Lagerbestand von über 10 Stück aufweisen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensätzeFilternDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
```

Listing 281: Datensätze filtern (DAO)

```
    Set DBS = Application.CurrentDb
    Set rs = DBS.OpenRecordset("SELECT * " & _
      "FROM Artikel WHERE Einzelpreis>30 and Lagerbestand > 10", dbOpenSnapshot)

    With rs
       Do While Not .EOF
          Debug.Print "Artikel-Nr: " & .Fields("Artikel-Nr").Value & vbLf & _
             "Artikelname: " & .Fields("Artikelname").Value & vbLf & _
             "Einzelpreis: " & .Fields("Einzelpreis").Value & vbLf & _
             "Lagerbestand: " & .Fields("Lagerbestand").Value & vbLf
          .MoveNext
       Loop

       .Close
    End With

    Set DBS = Nothing
    Set rs = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 281: Datensätze filtern (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Anschließend öffnen Sie mithilfe der Methode `OpenRecordSet` die Tabelle. Dabei erfassen Sie bereits direkt beim Öffnen der Tabelle die SQL-Anweisung. Als Ergebnis stehen nun alle Datensätze der Tabelle im `RecordSet`-Objekt `rs` und können dort abgegriffen werden.

In einer `Do While`-Schleife durchlaufen Sie alle Datensätze des `RecordSet` so lange, wie die Eigenschaft `EOF` (End of File) noch nicht erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz des `RecordSet`-Objekts ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

236 Datensätze zählen (ADO)

Möchten Sie ermitteln, wie viele Sätze sich in Ihrer Tabelle befinden, dann setzen Sie die Eigenschaft `RecordCount` ein. Diese Eigenschaft gibt die Anzahl der Datensätze einer angegebenen Tabelle aus.

Zählen über die Eigenschaft RecordCount

Im folgenden Beispiel aus Listing 282 wird die Anzahl der Datensätze der Tabelle `Kunden` ermittelt.

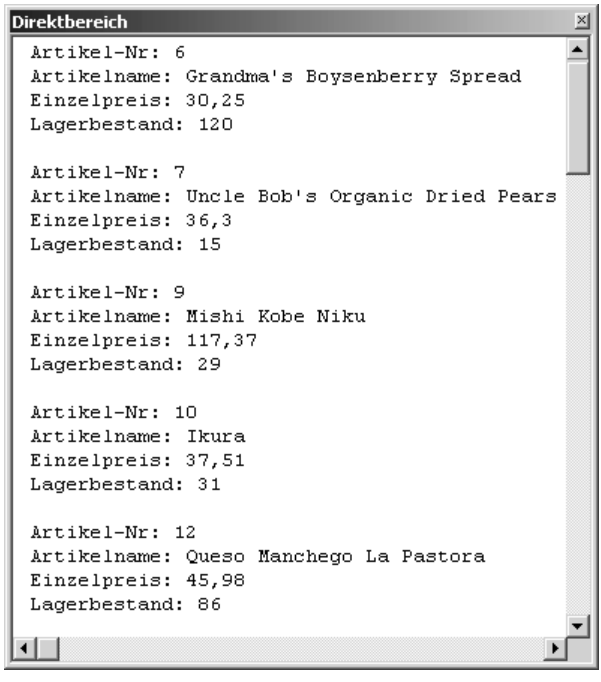


Abbildung 151: Die Daten wurden gefiltert und ausgegeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub DatensätzeZählen()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open "Kunden", Conn, adOpenKeyset, adLockOptimistic
      MsgBox "In der Tabelle befinden sich " & DBS.RecordCount & " Datensätze"
      .Close
   End With

   Conn.Close
   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub
```

Listing 282: Datensätze zählen – Variante 1 (ADO)

```
   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 282: Datensätze zählen – Variante 1 (ADO) (Forts.)

Zählen über die Funktion DCount

Eine etwas kürzere Variante bietet die Funktion `DCount` aus Listing 283.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeZählenVar2()
   MsgBox "Es befinden sich : " & _
      DCount("Firma", "Kunden") & _
      " Datensätze in der Tabelle!", vbInformation
End Sub
```

Listing 283: Datensätze zählen – Variante 2 (ADO)

Übergeben Sie der Funktion `DCount` als erstes Argument das Feld, das Sie zählen möchten. Im zweiten Argument geben Sie den Namen der Tabelle an, auf die Sie zugreifen möchten.

Gefilterte Datensätze zählen

Bei den letzten beiden Makros wurden jeweils alle Datensätze der Tabelle `Kunden` gemeldet. Die Eigenschaft `RecordCount` können Sie beispielsweise auch auf eine begrenzte Gruppe von Datensätzen anwenden. Stellen Sie sich einmal vor, Sie möchten die Anzahl der Kunden bestimmen, die aus `Großbritannien` oder `Frankreich` kommen. Sehen Sie sich die Lösung dieser Aufgabe in Listing 284 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeFilternUndZählen()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset
```

Listing 284: Datensätze filtern und zählen (ADO)

```
    With DBS
       .Open "Kunden", Conn, adOpenKeyset, adLockOptimistic
       .Filter = "Land = 'Großbritannien' or Land='Frankreich'"

       Do While Not .EOF
          Debug.Print .Fields("Kontaktperson").Value
          Debug.Print .Fields("Land").Value
          Debug.Print .Fields("Ort").Value & vbLf
          .MoveNext
       Loop
       MsgBox "In der Tabelle befinden sich " & _
          DBS.RecordCount & " Datensätze"
       .Close
    End With

    Conn.Close
    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 284: Datensätze filtern und zählen (ADO) (Forts.)

Nachdem Sie das Filterkriterium eingestellt haben, durchlaufen Sie in einer `Do While`-Schleife alle gefundenen Datensätze und geben Sie über die Auflistung `Fields` und die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus. Am Ende ermitteln Sie die Gesamtzahl der gefundenen Datensätze über die Eigenschaft `RecordCount`.

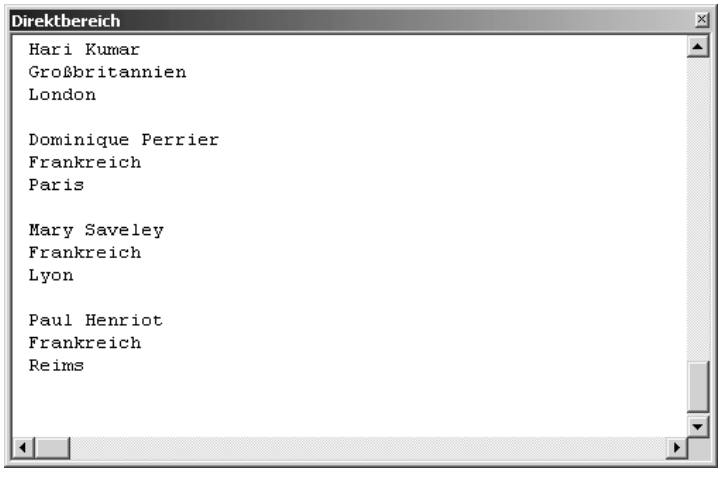


Abbildung 152: Kunden aus Großbritannien und Frankreich

237 Datensätze zählen (DAO)

Auch die Zählung von Datensätzen einer Tabelle funktioniert bei DAO über die Eigenschaft `RecordCount`.

Einfache Zählung vornehmen

Im folgenden Beispiel aus Listing 285 werden die Datensätze der Tabelle ARTIKEL gezählt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeZählenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Artikel", dbOpenTable, dbReadOnly)

   With rs
      MsgBox "In der Tabelle befinden sich " & _
         .RecordCount & " Datensätze"
      .Close
   End With

   Set DBS = Nothing
   Set rs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 285: Datensätze zählen (DAO)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Anschließend öffnen Sie mithilfe der Methode `OpenRecordSet` die Tabelle. Danach können Sie über die Eigenschaft `RecordCount` die Anzahl der Datensätze in der Tabelle zählen. Am Ende des Makros schließen Sie die Tabelle und heben die Objektverweise wieder auf.

Zählung nach Filterung durchführen

Beim folgenden Beispiel aus Listing 286 werden in der Tabelle `Kunden` vorher alle Kunden aus den `USA` und `Kanada` gefiltert und danach im Direktfenster ausgegeben und gezählt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeFilternUndZählenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler

   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("SELECT * " & _
      "FROM Kunden WHERE Land = 'Kanada' OR Land= 'USA'", dbOpenSnapshot)

   With rs
      Do While Not .EOF
         Debug.Print .Fields("Kontaktperson").Value
         Debug.Print .Fields("Land").Value
         Debug.Print .Fields("Ort").Value & vbLf
         .MoveNext
      Loop
      MsgBox "In der Tabelle befinden sich " & .RecordCount & " Datensätze"
      .Close
   End With

   Set DBS = Nothing
   Set rs = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 286: Datensätze filtern und zählen (DAO)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Öffnen Sie die Tabelle anschließend mithilfe der Methode `OpenRecordSet`. Dabei wenden Sie eine SQL-Anweisung schon beim Öffnen der Tabelle an, indem Sie die Kunden aus den `USA` und `Kanada` herausfiltern.

Jetzt können Sie über die Eigenschaft `RecordCount` die Anzahl der Datensätze in der Tabelle feststellen.

238 Datensätze summieren

Zum Summieren von Datensätzen steht Ihnen die Funktion `Dsum` zur Verfügung. Diese Funktion kann sowohl für ADO als auch für DAO eingesetzt werden.

Im Beispiel aus Listing 287 wird die Tabelle `Bestelldetails` ausgewertet. Dabei soll der Gesamtwert aller Bestellungen ermittelt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub GesamtsummeErmitteln()
   On Error GoTo fehler
   MsgBox "Die Gesamtsumme aller Bestellungen beträgt: " & _
      DSum("[Bestelldetails]![Einzelpreis]", "[Bestelldetails]")
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 287: Die Gesamtsumme aller Bestellungen ermitteln

Mit der Funktion `DSum` können Sie die Summe einer Gruppe von Werten in einer bestimmten Datensatzgruppe (Domäne) berechnen. Übergeben Sie der Funktion den Namen der Tabelle sowie des Datenfelds, dessen Summe Sie berechnen möchten. Im Anhang können Sie eine Auflistung weiterer D-Funktionen einsehen.

239 Datensätze klonen (ADO)

Die Methode `Clone` erstellt eine Kopie eines `Recordset`-Objekts, die auf das Original des `Recordset`-Objekts verweist. Die Syntax dieser Methode lautet:

```
Set Kopie = Original.Clone
```

Argument	Beschreibung
`Kopie`	Erforderlich. Eine Objektvariable, die die Kopie des zu erstellenden `Recordset`-Objekts kennzeichnet.
`Original`	Erforderlich. Eine Objektvariable, die das zu kopierende `Recordset`-Objekt kennzeichnet.

Tabelle 98: Die Argumente der Methode Clone

Im folgenden Beispiel aus Listing 288 wird ein bestimmter Datensatz in der Tabelle `Artikel` gesucht, dieser danach geklont und am Ende der Tabelle unter Vergabe einer neuen `Artikel-Nr` eingefügt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'====================================================

Sub DatensatzClonen()
   Dim rst1 As Recordset
   Dim rst2 As Recordset
   Dim strSql As String
   Dim lngNr As Long
   Dim intZ As Integer

   On Error GoTo fehler
   strSql = "Select * from Artikel where Artikelname='Outback Lager'"
   Set rst1 = CurrentDb.OpenRecordset(strSql)
   Set rst2 = rst1.Clone

   With rst2
      .AddNew
      'Übertragen der Felder
      For intZ = 1 To .Fields.Count - 1
         rst2(intZ) = rst1(intZ)
      Next intZ
      'Ermitteln der letzten Artikel-Nr
      lngNr = DMax("[Artikel-Nr]", "[Artikel]") + 1
      rst2.Fields(0).Value = lngNr
      .Update
      .Close
   End With

   Set rst1 = Nothing
   Set rst2 = Nothing
   Exit Sub


   fehler:
      Msgbox Err.Number & " " & Err.Description
End Sub
```

Listing 288: Datensatz klonen und am Ende der Tabelle anhängen (ADO)

Im ersten Schritt geben Sie die SQL-Anweisung in einer String-Variablen bekannt. Danach wenden Sie die Methode `OpenRecordSet` an, um ein neues `RecordSet`-Objekt zu erstellen. Der gefundene Satz steht anschließend in der Objektvariablen `rst1` für eine weitere Verarbeitung bereit. Über die Methode `Clone` wird dieser Satz dupliziert und in der Objektvariablen `rst2` zur Verfügung gestellt. Mithilfe der Methode `AddNew` wird ein neuer Datensatz der Tabelle

angefügt. Im Anschluss daran werden die Feldinhalte von `rst1` nach `rst2` übertragen. Mithilfe der Funktion `DMax` wird die letzte vergebene Artikelnummer in der Tabelle gesucht und um den Wert 1 erhöht. Die Methode `Update` sorgt dafür, dass dieser neue Datensatz gespeichert wird.

Artikel : Tabelle

Artikel-	Artikelname	Lieferant	Kategorie	Liefereinheit
67	Laughing Lumberjack Lager	Bigfoot Breweries	Getränke	24 x 12-oz-Flaschen
68	Scottish Longbreads	Specialty Biscuits, Ltd.	Süßwaren	10 Kartons x 8 Stück
69	Gudbrandsdalsost	Norske Meierier	Milchprodukte	10-kg-Paket
70	Outback Lager	Pavlova, Ltd.	Getränke	24 x 355-ml-Flaschen
71	Fløtemysost	Norske Meierier	Milchprodukte	10 x 500-g-Packungen
72	Mozzarella di Giovanni	Formaggi Fortini s.r.l.	Milchprodukte	24 x 200 g-Packungen
73	Röd Kaviar	Svensk Sjöföda AB	Meeresfrüchte	24 x 150-g-Gläser
74	Longlife Tofu	Tokyo Traders	Naturprodukte	5-kg-Paket
75	Rhönbräu Klosterbier	Plutzer Lebensmittelgroßmärkte A	Getränke	24 x 0,5-l-Flaschen
76	Lakkalikööri	Karkki Oy	Getränke	500-ml-Flasche
77	Original Frankfurter grüne Soße	Plutzer Lebensmittelgroßmärkte A	Gewürze	12 Kartons
78	Outback Lager	Pavlova, Ltd.	Getränke	24 x 355-ml-Flaschen
oWert)				

Datensatz: 78 von 78

Abbildung 153: Der geklonte Datensatz wurde am Ende der Tabelle eingefügt.

240 Datensätze klonen (DAO)

Das Beispiel aus Listing 288 wird im Makro aus Listing 289 über den Einsatz von DAO gelöst.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'====================================================

Sub DatensatzClonenDAO()
   Dim DBS As DAO.Database
   Dim rst1 As DAO.Recordset
   Dim rst2 As DAO.Recordset
   Dim lngNr As Long
   Dim intZ As Integer

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rst1 = DBS.OpenRecordset _
      ("Select * from ARTIKEL where ARTIKELNAME='Outback Lager'", dbOpenDynaset)

   Set rst2 = rst1.Clone

   With rst2
      .AddNew
      'Übertragen der Felder
```

Listing 289: Datensatz klonen und am Ende der Tabelle anhängen (DAO)

```
      For intZ = 1 To .Fields.Count - 1
         rst2(intZ) = rst1(intZ)
      Next intZ
      'Ermitteln der letzten Artikel-Nr
      lngNr = DMax("[Artikel-Nr]", "[Artikel]") + 1
      rst2.Fields(0).Value = lngNr
      .Update
      .Close
   End With

   Set rst1 = Nothing
   Set rst2 = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 289: Datensatz klonen und am Ende der Tabelle anhängen (DAO) (Forts.)

241 Lesezeichen setzen (ADO)

Wenn Sie durch Ihre Tabelle navigieren, verändern Sie laufend die aktuelle Position des Datenzeigers. Mit der Eigenschaft `Bookmark` können Sie die Position des aktuellen Datensatzes speichern und zu einem beliebigen Zeitpunkt zu diesem Datensatz zurückgehen. Dabei speichern Sie diese Position in einer Variablen vom Typ `Variant`. Wie das genau funktioniert, sehen Sie in Listing 290. Dort lesen Sie alle Datensätze aus der Tabelle `Personal` der Datenbank *Tabellen.mdb*. Dann setzen Sie ein Lesezeichen auf den ersten und den letzten Datensatz.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub LesezeichenSetzen()
   Dim Conn As ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim varLZ1 As Variant
   Dim varLZ2 As Variant

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open Source:="SELECT Nachname,Vorname FROM Personal " & _
      "ORDER BY Nachname", ActiveConnection:=CurrentProject.Connection, _
```

Listing 290: Lesezeichen setzen

```
      CursorType:=adOpenStatic, Options:=adCmdText

      .MoveLast
      varLZ1 = .Bookmark
      Debug.Print "Letzter gefundener Satz: " & varLZ1
      Debug.Print .Fields("Nachname") & Chr(13)

      .MoveFirst
      varLZ2 = .Bookmark
      Debug.Print "Erster gefundener Satz:" & varLZ2
      Debug.Print .Fields("Nachname") & Chr(13)

      DBS.Bookmark = varLZ1
      Debug.Print "Wieder zurück zu:" & varLZ1
      Debug.Print .Fields("Nachname") & Chr(13)
      .Close
   End With

   Conn.Close
   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 290: Lesezeichen setzen (Forts.)

Öffnen Sie im ersten Schritt die Tabelle `Personal` und verwenden Sie dabei eine `SELECT`-Anweisung. Wählen Sie alle Datensätze der Tabelle aus und sortieren Sie diese mithilfe des Arguments `ORDER BY`. Danach springen Sie über die Methode `MoveLast` zum letzten Datensatz der Tabelle. Dort angekommen speichern Sie die augenblickliche Position in der Variablen `VarLZ1` und geben diese Position und den dazugehörigen Nachnamen des Mitarbeiters im Direktbereich aus. Wenden Sie jetzt die Methode `MoveFirst` an, um zum ersten Datensatz in der Tabelle zu gelangen. Speichern Sie auch hier die aktuelle Position in einer Variablen mit dem Namen `varLZ2`. Geben Sie diese Position und den dazugehörigen Namen wiederum zur Kontrolle im Direktbereich aus.

Mit der Anweisung `DBS.Bookmark = varLZ1` springen Sie direkt wieder an Ihr vorher gesetztes Lesezeichen und geben die Position und den Nachnamen im Direktfenster aus.

242 Datensätze löschen (ADO)

Über den Einsatz der Methode `Delete` können Sie Datensätze aus einer Tabelle löschen. Die Syntax dieser Methode lautet:

```
recordset.Delete AffectRecords
```

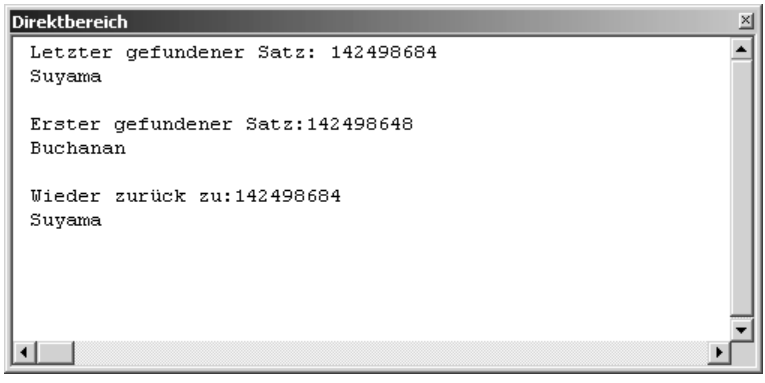


Abbildung 154: Lesezeichen setzen und aktivieren

Argument	Beschreibung
recordset	Gibt ein Recordset-Objekt zurück.
AffectRecords	Optional. Hier können die zu löschenden Sätze gezählt werden.

Tabelle 99: Die Argumente der Methode Delete

Im folgenden Beispiel aus Listing 291 werden alle Datensätze aus der Tabelle Artikel entfernt, die einen Lagerbestand von 0 aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeLöschen()
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   DBS.Open _
      "Select * FROM Artikel2 where (Lagerbestand = 0)", _
      Conn, adOpenKeyset, adLockOptimistic

   Do Until DBS.EOF
      DBS.Delete
      DBS.MoveNext
   Loop
```

Listing 291: Datensätze löschen

```
    DBS.Close
    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 291: Datensätze löschen (Forts.)

Beim Löschen von Datensätzen aus einer Tabelle müssen Sie darauf achten, dass Sie mithilfe der Methode `MoveNext` jeweils zum nächsten Datensatz springen.

243 Datensätze löschen (DAO)

Die Löschung von Datensätzen erfolgt bei DAO über die Methode `Delete`.

Bestimmte Kunden löschen

Im Beispiel aus Listing 292 werden aus der Tabelle `Kunden2` alle Kunden aus Deutschland gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeLöschenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Kunden2", dbOpenDynaset)

   With rs
      .FindFirst "Land =  'Deutschland'"

      Do Until .NoMatch = True
      .Delete
      .FindNext "Land = 'Deutschland'"
   Loop

   .Close
   End With

   Set DBS = Nothing
```

Listing 292: Datensätze löschen (DAO)

```
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 292: Datensätze löschen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Öffnen Sie anschließend die Tabelle mithilfe der Methode `OpenRecordSet`. Wenden Sie die Methode `FindFirst` an, um den ersten Satz in der Tabelle zu finden, der dem Suchkriterium entspricht.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze der Tabelle so lange, bis kein Datensatz mehr gefunden werden kann, der dem Suchkriterium entspricht. In diesem Fall meldet die Eigenschaft `NoMatch` den Wert `True`. Innerhalb der Schleife wenden Sie die Methode `Delete` an, um alle die Datensätze zu löschen, die dem Suchkriterium entsprechen. Über die Methode `FindNext` finden Sie jeweils den nächsten Datensatz, der den Suchkriterien entspricht.

Schließen Sie am Ende des Makros die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

Lager bereinigen

Im folgenden Beispiel aus Listing 293 werden in der Tabelle `Artikel3` alle Datensätze gelöscht, die einen Lagerbestand von 0 haben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeLöschenDAO2()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Artikel3", dbOpenDynaset)

   With rs
      .MoveFirst
```

Listing 293: Datensätze mit Nullbestand löschen (DAO)

```
      Do Until rs.EOF
         If rs.Fields("Lagerbestand").Value = 0 Then rs.Delete
         .MoveNext
      Loop

      .Close
   End With

   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 293: Datensätze mit Nullbestand löschen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Objekte, eines vom Typ `Database` und eines vom Typ `Recordset`. Danach weisen Sie die aktuelle Datenbank der Objektvariablen `DBS` über die Anweisung `Set` zu. Anschließend führen Sie die Methode `OpenRecordSet` durch, um die Tabelle zu öffnen. Wenden Sie die Methode `MoveFirst` an, um den ersten Satz in der Tabelle zu aktivieren.

In einer `Do Until`-Schleife durchlaufen Sie alle Datensätze der Tabelle so lange, bis die Eigenschaft `EOF` (End of File) erfüllt ist. Innerhalb der Schleife wenden Sie die Methode `Delete` an, um die Datensätze zu löschen, die einen Lagerbestand von 0 haben. Über die Methode `MoveNext` stellen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz ein.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

244 Datensätze hinzufügen (ADO)

Durch den Einsatz der Methode `AddNew` können Sie einer Tabelle einen neuen Datensatz hinzufügen. Alles, was Sie dazu wissen müssen, sind die genauen Feldbezeichnungen, die Sie in der Entwurfsansicht der Tabelle einsehen können.

Im folgenden Beispiel aus Listing 294 wird ein neuer Satz in die Tabelle `Personal` eingefügt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=================================================
```

Listing 294: Neuen Datensatz hinzufügen und füllen (ADO)

```
Sub DatensatzHinzufügen()
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open "Personal", Conn, adOpenKeyset, adLockOptimistic
      .AddNew
      .Fields("Nachname") = "Held"
      .Fields("Vorname") = "Bernd"
      .Fields("Position") = "Externer Berater"
      .Fields("Anrede") = "Herr"
      .Fields("Geburtsdatum") = "02.04.1969"
      .Fields("Einstellung") = "17.01.2002"
      .Fields("Straße") = "Maybachstr. 35"
      .Fields("Ort") = "Stuttgart"
      .Fields("Region") = "BW"
      .Fields("PLZ") = "70469"
      .Fields("Land") = "Deutschland"
      .Update
      .Close
   End With

   Set DBS = Nothing
   Set Conn = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

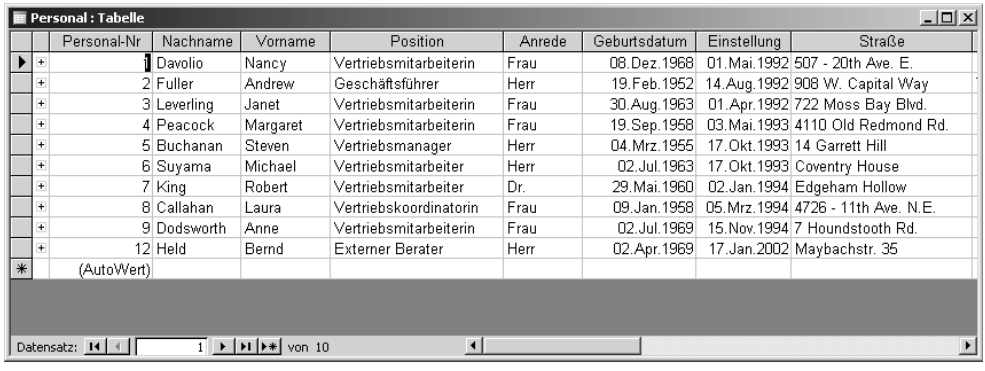
Listing 294: Neuen Datensatz hinzufügen und füllen (ADO) (Forts.)

Das erste Feld, die `Personal-Nr` brauchen Sie übrigens nicht zu füllen. Dieses Feld weist den Felddatentyp `AUTOWERT` auf, was bedeutet, dass Access selbst einen gültigen Wert vergibt. Über die Methode `AddNew` wird ein neuer, noch leerer Datensatz in die Tabelle geschrieben. Danach werden die Felder über die Auflistung `Fields` gefüllt. Über den Einsatz der Methode `Update` wird der Datensatz gesichert. Die Methode `Close` schließt die Tabelle wieder.

Neue Daten aus Excel beziehen

Beim folgenden Beispiel aus Listing 295 wird die vorherige Aufgabe aus Listing 294 erweitert. Dabei liegen die neuen Mitarbeiterdaten in einer Excel-Tabelle vor, die jetzt Zeile für Zeile in die Tabelle `Personal` eingelesen werden soll.

Die Excel-Tabelle mit dem Namen `Mitarbeiter` sieht dabei wie in Abbildung 156 gezeigt aus.

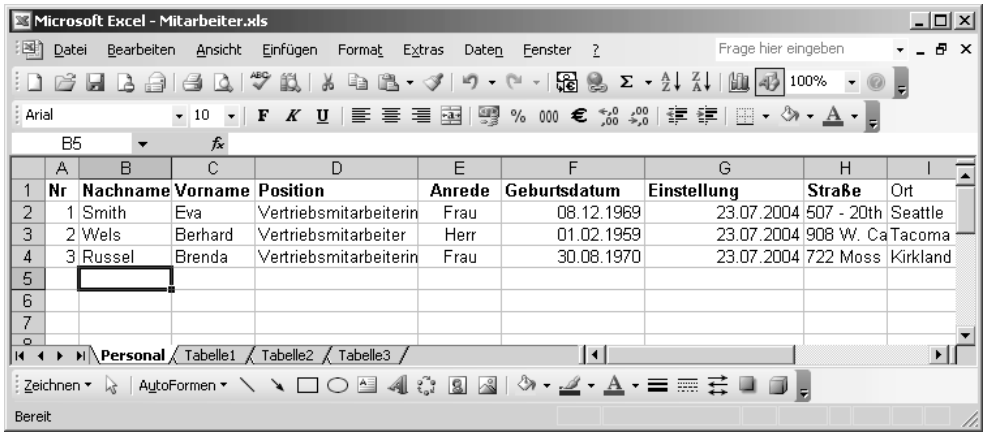


Personal : Tabelle

Personal-Nr	Nachname	Vorname	Position	Anrede	Geburtsdatum	Einstellung	Straße
1	Davolio	Nancy	Vertriebsmitarbeiterin	Frau	08.Dez.1968	01.Mai.1992	507 - 20th Ave. E.
2	Fuller	Andrew	Geschäftsführer	Herr	19.Feb.1952	14.Aug.1992	908 W. Capital Way
3	Leverling	Janet	Vertriebsmitarbeiterin	Frau	30.Aug.1963	01.Apr.1992	722 Moss Bay Blvd.
4	Peacock	Margaret	Vertriebsmitarbeiterin	Frau	19.Sep.1958	03.Mai.1993	4110 Old Redmond Rd.
5	Buchanan	Steven	Vertriebsmanager	Herr	04.Mrz.1955	17.Okt.1993	14 Garrett Hill
6	Suyama	Michael	Vertriebsmitarbeiter	Herr	02.Jul.1963	17.Okt.1993	Coventry House
7	King	Robert	Vertriebsmitarbeiter	Dr.	29.Mai.1960	02.Jan.1994	Edgeham Hollow
8	Callahan	Laura	Vertriebskoordinatorin	Frau	09.Jan.1958	05.Mrz.1994	4726 - 11th Ave. N.E.
9	Dodsworth	Anne	Vertriebsmitarbeiterin	Frau	02.Jul.1969	15.Nov.1994	7 Houndstooth Rd.
12	Held	Bernd	Externer Berater	Herr	02.Apr.1969	17.Jan.2002	Maybachstr. 35
(AutoWert)							

Datensatz: 1 von 10

Abbildung 155: Der neue Mitarbeiter wurde am Ende der Tabelle eingefügt.



Microsoft Excel - Mitarbeiter.xls

Datei Bearbeiten Ansicht Einfügen Format Extras Daten Fenster ?

Frage hier eingeben

Arial 10

B5

	A	B	C	D	E	F	G	H	I
1	**Nr**	**Nachname**	**Vorname**	**Position**	**Anrede**	**Geburtsdatum**	**Einstellung**	**Straße**	Ort
2	1	Smith	Eva	Vertriebsmitarbeiterin	Frau	08.12.1969	23.07.2004	507 - 20th	Seattle
3	2	Wels	Berhard	Vertriebsmitarbeiter	Herr	01.02.1959	23.07.2004	908 W. Ca	Tacoma
4	3	Russel	Brenda	Vertriebsmitarbeiterin	Frau	30.08.1970	23.07.2004	722 Moss	Kirkland
5									
6									
7									

Personal | Tabelle1 | Tabelle2 | Tabelle3

Zeichnen AutoFormen

Bereit

Abbildung 156: Die neuen Mitarbeiter liegen in einer Excel-Tabelle vor.

```
'=====================================================
' Auf CD      Buchdaten\Beispiele\Kap04
' Dateiname   Tabellen.mdb
' Modul       mdlADO
'=====================================================

Sub DatenAusExcelTabelleHinzufügen()
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim xlApp As Object
   Dim xlMappe As Object
   Dim intz As Integer

   On Error GoTo fehler
   Set xlApp = CreateObject("Excel.Application")
```

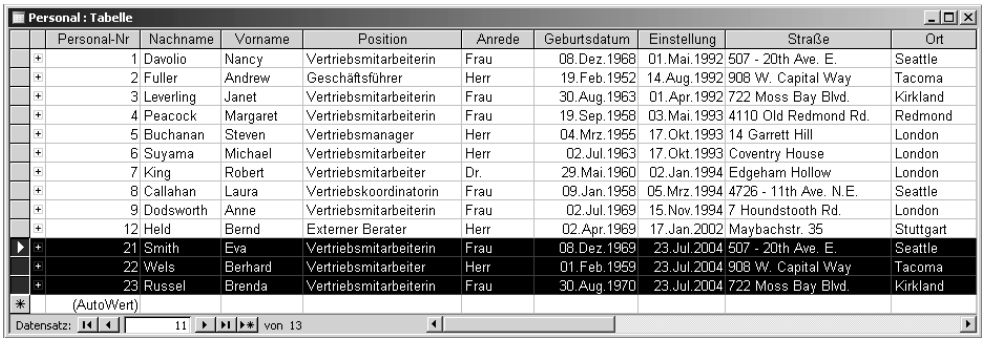
Listing 295: Personaldaten aus einer Excel-Tabelle importieren

```
    Set xlMappe = _
       xlApp.workbooks.Open(Application.CurrentProject.Path & "\Mitarbeiter.xls")

    Set Conn = CurrentProject.Connection
    Set DBS = New ADODB.Recordset
    DBS.Open "Personal", Conn, adOpenKeyset, adLockOptimistic

    For intz = 2 To xlMappe.sheets("Personal").usedrange.rows.Count
       With DBS
          .AddNew
          .Fields("Nachname") = xlMappe.sheets(1).cells(intz, 2).Value
          .Fields("Vorname") = xlMappe.sheets(1).cells(intz, 3).Value
          .Fields("Position") = xlMappe.sheets(1).cells(intz, 4).Value
          .Fields("Anrede") = xlMappe.sheets(1).cells(intz, 5).Value
          .Fields("Geburtsdatum") = xlMappe.sheets(1).cells(intz, 6).Value
          .Fields("Einstellung") = xlMappe.sheets(1).cells(intz, 7).Value
          .Fields("Straße") = xlMappe.sheets(1).cells(intz, 8).Value
          .Fields("Ort") = xlMappe.sheets(1).cells(intz, 9).Value
          .Fields("Region") = xlMappe.sheets(1).cells(intz, 10).Value
          .Fields("PLZ") = xlMappe.sheets(1).cells(intz, 11).Value
          .Fields("Land") = xlMappe.sheets(1).cells(intz, 12).Value
          .Update
       End With
    Next intz

    DBS.Close
    xlMappe.Close
    xlApp.Quit
    Set DBS = Nothing
    Set Conn = Nothing
    Set xlMappe = Nothing
    Set xlApp = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 295: Personaldaten aus einer Excel-Tabelle importieren (Forts.)

Über die Funktion `CreateObject` erstellen Sie einen Verweis auf die Excel-Bibliothek. So erhalten Sie Zugriff auf alle Methoden und Eigenschaften, die für Excel-Objekte zur Verfügung stehen. Bei diesem Vorgang spricht man übrigens vom Late-Binding. Näheres dazu erfahren Sie in Kapitel 10 dieses Buchs.

Öffnen Sie die Excel-Arbeitsmappe über die Methode `Open`. Arbeiten Sie in einer Schleife Satz für Satz ab und übertragen Sie die Datensätze in die Access-Tabelle.



Personal : Tabelle

Personal-Nr	Nachname	Vorname	Position	Anrede	Geburtsdatum	Einstellung	Straße	Ort
1	Davolio	Nancy	Vertriebsmitarbeiterin	Frau	08.Dez.1968	01.Mai.1992	507 - 20th Ave. E.	Seattle
2	Fuller	Andrew	Geschäftsführer	Herr	19.Feb.1952	14.Aug.1992	908 W. Capital Way	Tacoma
3	Leverling	Janet	Vertriebsmitarbeiterin	Frau	30.Aug.1963	01.Apr.1992	722 Moss Bay Blvd.	Kirkland
4	Peacock	Margaret	Vertriebsmitarbeiterin	Frau	19.Sep.1958	03.Mai.1993	4110 Old Redmond Rd.	Redmond
5	Buchanan	Steven	Vertriebsmanager	Herr	04.Mrz.1955	17.Okt.1993	14 Garrett Hill	London
6	Suyama	Michael	Vertriebsmitarbeiter	Herr	02.Jul.1963	17.Okt.1993	Coventry House	London
7	King	Robert	Vertriebsmitarbeiter	Dr.	29.Mai.1960	02.Jan.1994	Edgeham Hollow	London
8	Callahan	Laura	Vertriebskoordinatorin	Frau	09.Jan.1958	05.Mrz.1994	4726 - 11th Ave. N.E.	Seattle
9	Dodsworth	Anne	Vertriebsmitarbeiterin	Frau	02.Jul.1969	15.Nov.1994	7 Houndstooth Rd.	London
12	Held	Bernd	Externer Berater	Herr	02.Apr.1969	17.Jan.2002	Maybachstr. 35	Stuttgart
21	Smith	Eva	Vertriebsmitarbeiterin	Frau	08.Dez.1969	23.Jul.2004	507 - 20th Ave. E.	Seattle
22	Wels	Berhard	Vertriebsmitarbeiter	Herr	01.Feb.1959	23.Jul.2004	908 W. Capital Way	Tacoma
23	Russel	Brenda	Vertriebsmitarbeiterin	Frau	30.Aug.1970	23.Jul.2004	722 Moss Bay Blvd.	Kirkland
	(AutoWert)							

Datensatz: 11 von 13

Abbildung 157: Die neuen Daten wurden der Access-Tabelle hinzugefügt.

245 Datensätze hinzufügen (DAO)

Die Neuanlage von Datensätzen funktioniert in DAO ganz ähnlich wie bei ADO. Im folgenden Beispiel aus Listing 296 wird ebenso ein neuer Satz der Tabelle `Personal` hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensatzHinzufügenDAO()
   Dim DBS As DAO.Database
   Dim rs As DAO.Recordset

   On Error GoTo fehler
   Set DBS = Application.CurrentDb
   Set rs = DBS.OpenRecordset("Personal", dbOpenDynaset)

   With rs
      .AddNew
      .Fields("Nachname") = "Held"
      .Fields("Vorname") = "Bernd"
      .Fields("Position") = "Externer Berater"
      .Fields("Anrede") = "Herr"
      .Fields("Geburtsdatum") = "02.04.1969"
      .Fields("Einstellung") = "17.01.2002"
      .Fields("Straße") = "Maybachstr. 35"
      .Fields("Ort") = "Stuttgart"
      .Fields("Region") = "BW"
      .Fields("PLZ") = "70469"
      .Fields("Land") = "Deutschland"
      .Update
      .Close
   End With
```

Listing 296: Neuen Datensatz anlegen (DAO)

```
    Set rs = Nothing
    Set DBS = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 296: Neuen Datensatz anlegen (DAO) (Forts.)

Über die Methode `AddNew` wird ein neuer, noch leerer Datensatz in die Tabelle geschrieben. Danach werden die Felder über die Auflistung `Fields` gefüllt. Über den Einsatz der Methode `Update` wird der Datensatz gesichert. Die Methode `Close` schließt die Tabelle wieder.

246 Der Einsatz von ADOX und DAO

Möchten Sie noch tiefer in die Materie einsteigen und an Tabellenstrukturen herumbasteln, dann benötigen Sie die Bibliothek ADO EXTENSIONS FOR DDL AND SECURITY (ADOX). Dieses Objektmodell enthält Objekte, Eigenschaften und Methoden zum Erstellen, Bearbeiten und Anzeigen des Aufbaus von Tabellen. Binden Sie diese Bibliothek in der Entwicklungsumgebung ein, indem Sie aus dem Menü EXTRAS den Befehl VERWEISE auswählen, die Bibliothek aktivieren und mit OK bestätigen.

247 Tabellennamen ermitteln

Über die Eigenschaft `Type` können Sie ermitteln, ob es sich bei einer Tabelle um eine Systemtabelle, eine normale Tabelle oder eine Ansicht handelt. Für den Fall, dass es sich um eine Tabelle handelt, gibt diese Eigenschaft den Text `Table` zurück.

Im folgenden Beispiel aus Listing 297 werden die unterschiedlichen Tabellentypen im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabellenAusgeben()
   Dim Katalog As ADOX.Catalog
   Dim TabInfo As ADOX.Table

   On Error GoTo fehler
   Set Katalog = New ADOX.Catalog
```

Listing 297: Alle Tabellentypen ermitteln

```
    Katalog.ActiveConnection = _
       "Provider=Microsoft.Jet.OLEDB.4.0;" & _
       "Data Source= C:\Eigene Dateien\Nordwind.mdb"

    For Each TabInfo In Katalog.Tables
       With TabInfo
          Debug.Print "Name: " & vbTab & .Name & vbLf & _
             "Typ:" & vbTab & .Type & vbLf
       End With
    Next TabInfo

    Set TabInfo = Nothing
    Set Katalog = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 297: Alle Tabellentypen ermitteln (Forts.)

Deklarieren Sie im ersten Schritt zwei ADOX-Objektvariablen. Im `Catalog`-Objekt sind unter anderem alle Tabellen der Datenbank verzeichnet. Arbeiten Sie diese Tabellen mithilfe einer `For each Next`-Schleife ab und ermitteln Sie den Namen sowie den Tabellentyp über die Eigenschaften `Name` und `Type`.

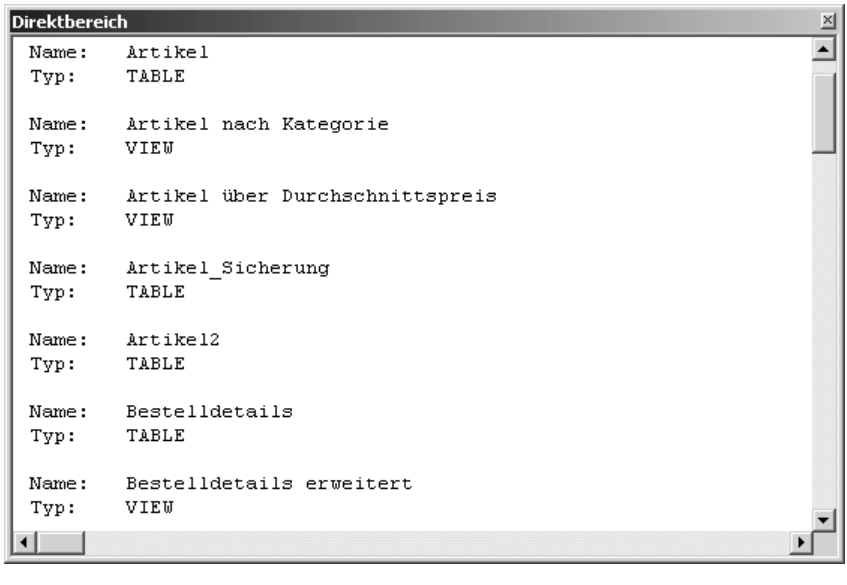


Abbildung 158: Tabelle oder Abfrage

248 Tabelleneigenschaften auslesen

Über die ADOX-Bibliothek haben Sie Zugriff auf Tabelleneigenschaften wie beispielsweise das Erstellungs- sowie das letzte Änderungsdatum der Tabelle.

Im folgenden Beispiel aus Listing 298 wird das Erstellungsdatum sowie das letzte Änderungsdatum der Tabelle `Artikel` aus der Datenbank *Nordwind.mdb* ermittelt und am Bildschirm angezeigt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=================================================

Sub TabellenInfosAusgeben()
   Dim Katalog As ADOX.Catalog
   Dim TabInfo As ADOX.Table

   On Error GoTo fehler
   Set Katalog = New ADOX.Catalog
   Katalog.ActiveConnection = _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source= C:\Eigene Dateien\Nordwind.mdb"

   Set TabInfo = Katalog.Tables("Artikel")
   With TabInfo
      MsgBox "Name:" & .Name & vbLf & _
         "Erstellungsdatum: " & .DateCreated & vbLf & _
         "Änderungsdatum:   " & .DateModified
   End With

   Set TabInfo = Nothing
   Set Katalog = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 298: Tabelleninfos ausgeben

Definieren Sie im ersten Schritt ein Objekt vom Typ `Catalog`. Über dieses Objekt können Sie unter anderem auf alle Tabellen Ihrer Datenbank zugreifen. Danach geben Sie die Datenquelle über die Eigenschaft `ActiveConnection` bekannt. Setzen Sie eine Schleife auf und durchlaufen Sie alle Tabellen, die sich in Ihrer Datenbank befinden. Innerhalb der Schleife ermitteln Sie den Namen, das Erstellungsdatum sowie das letzte Änderungsdatum jeder Tabelle und geben diese im Direktfenster aus. Heben Sie die Objektverknüpfung am Ende des Makros mithilfe der Anweisung `Set Katalog = Nothing` wieder auf.

Abbildung 159: Zugriff auf Tabelleneigenschaften über ADOX

249 Tabellenstrukturen auslesen

Um die Struktur der einzelnen Felder einer Tabelle auszulesen, gibt es einige Eigenschaften, die Sie über das ADOX-Objekt `Column` ansprechen können.

Im folgenden Beispiel aus Listing 299 werden die Felddatentypen der Tabelle `Artikel` ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabellenStrukturenAuslesen()
   Dim ADOKAT As New ADOX.Catalog
   Dim TabDef As ADOX.Table
   Dim strText As String
   Dim intz As Integer

   On Error GoTo fehler
   ADOKAT.ActiveConnection = CurrentProject.Connection
   Set TabDef = ADOKAT.Tables("Artikel")

   With TabDef
      For intz = 0 To .Columns.Count - 1
         Debug.Print .Columns(intz).Name
         Select Case .Columns(intz).Type
            Case 202
               strText = "Text"
            Case 2
               strText = "Zahl - Integer"
            Case 3
               strText = "Zahl - Long Integer"
            Case 6
               strText = "Währung"
            Case 7
               strText = "Datum/Uhrzeit"
            Case 11
               strText = "Ja/Nein"
```

Listing 299: Datenfeldtypen ermitteln

```
            Case Else
               strText = "noch nicht erfasst"
         End Select

         Debug.Print "Felddatentyp: " & _
         .Columns(intz).Type & " ---> " & strText & vbLf
      Next intz
   End With

   Set ADOKAT = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 299: Datenfeldtypen ermitteln (Forts.)

Mithilfe einer `Select Case`-Anweisung können Sie den numerischen Wert, den Ihnen die Eigenschaft `Type` meldet, in einen Text umwandeln.

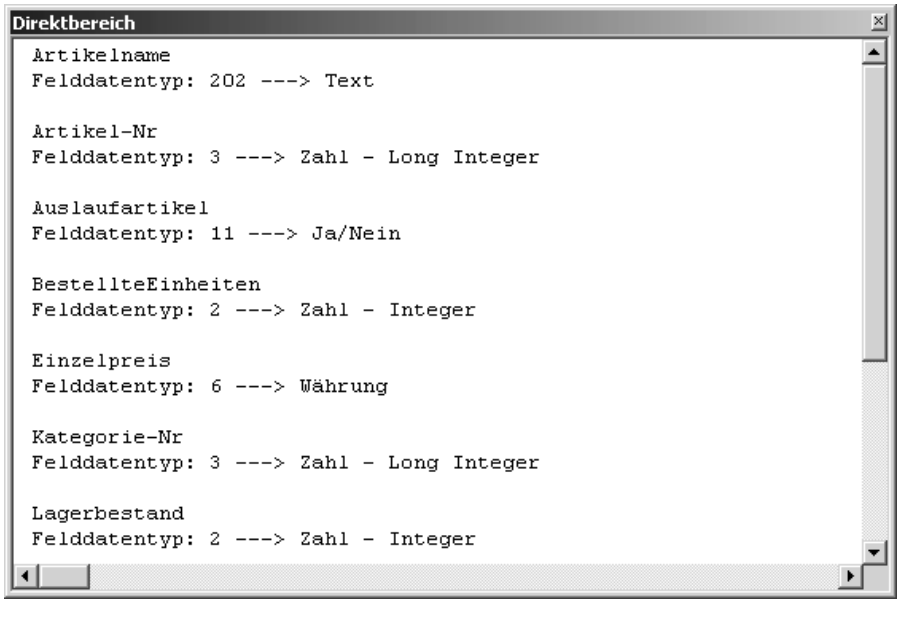


Abbildung 160: Die Feldtypen wurden ausgelesen.

250 Tabelleneigenschaften abfragen

Um die Struktur der einzelnen Felder einer Tabelle auszulesen, gibt es einige Eigenschaften, die Sie über das Objekt `Column` ansprechen können.

Im folgenden Beispiel aus Listing 300 lesen Sie alle Datenfelddefinitionen der Tabelle `Artikel` aus der Datenbank *Tabellen.mdb* aus.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=================================================

Sub FeldInfosAuslesen()
   Dim ADOKAT As New ADOX.Catalog
   Dim TabDef As ADOX.Table
   Dim intz As Integer

   On Error GoTo fehler
   ADOKAT.ActiveConnection = CurrentProject.Connection
   Set TabDef = ADOKAT.Tables("Artikel")

   With TabDef
      For intz = 0 To .Columns.Count - 1
         Debug.Print .Columns(intz).Name
         Debug.Print .Columns(intz).Properties("Description")
         Debug.Print .Columns(intz).DefinedSize
         Debug.Print .Columns(intz).Type
         Debug.Print .Columns(intz).NumericScale
         Debug.Print .Columns(intz).Precision
         Debug.Print .Columns(intz).Attributes
      Next intz
   End With

   Set ADOKAT = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 300: Tabelleneigenschaften auslesen

Über die Eigenschaft `Name` können Sie den Namen des Felds ermitteln. Den Datentyp des Datenfelds finden Sie über die Eigenschaft `Type` heraus. Mit der Eigenschaft `Attributes` können Sie bestimmen, ob die Spalte eine feste Länge hat bzw. ob sie Nullwerte enthalten darf. Die maximale Größe der Spalte legen Sie mit der Eigenschaft `DefinedSize` fest. Bei numerischen Werten existiert die Skalierung über die Eigenschaft `NumericScale`, die Sie abfragen

können. Für numerische Datenwerte können Sie die maximale Präzision mithilfe der Eigenschaft `Precision` angeben.

Mit der Eigenschaft `ParentCatalog` geben Sie das `Catalog`-Objekt an, das Eigentümer der Spalte ist. Die Eigenschaft `RelatedColumn` liefert für Schlüsselspalten den Namen der verbundenen Spalte in der verbundenen Tabelle. Für Indexspalten können Sie über die Eigenschaft `SortOrder` ermitteln, ob die Sortierreihenfolge aufsteigend oder absteigend ist. Auf providerspezifische Eigenschaften – beispielsweise die Feldbeschreibung des Datenfelds – können Sie mit der `Properties`-Auflistung zugreifen.

Geben Sie im Auflistungsobjekt `Tables` an, welche Tabelle Sie auslesen möchten. Dazu verwenden Sie die Anweisung `Set`. Danach wenden Sie das Auflistungsobjekt `Columns` an, das alle Datenfelder der Tabelle enthält. Diese Datenfelder durchlaufen Sie mit einer Schleife, die den Startwert `0` hat. Über die Eigenschaft `Count` ermitteln Sie die Anzahl der definierten Datenfelder in der Tabelle. Von diesem ermittelten Wert müssen Sie den Wert `1` subtrahieren, damit es zu keinem Makroabsturz kommt. Innerhalb der Schleife fragen Sie die einzelnen Felder über die vorher beschriebenen Eigenschaften ab und geben diese im Direktfenster aus. Heben Sie am Ende des Makros die Objektverweise auf den Katalog über das Schlüsselwort `Nothing` wieder auf, um den Arbeitsspeicher wieder freizugeben.

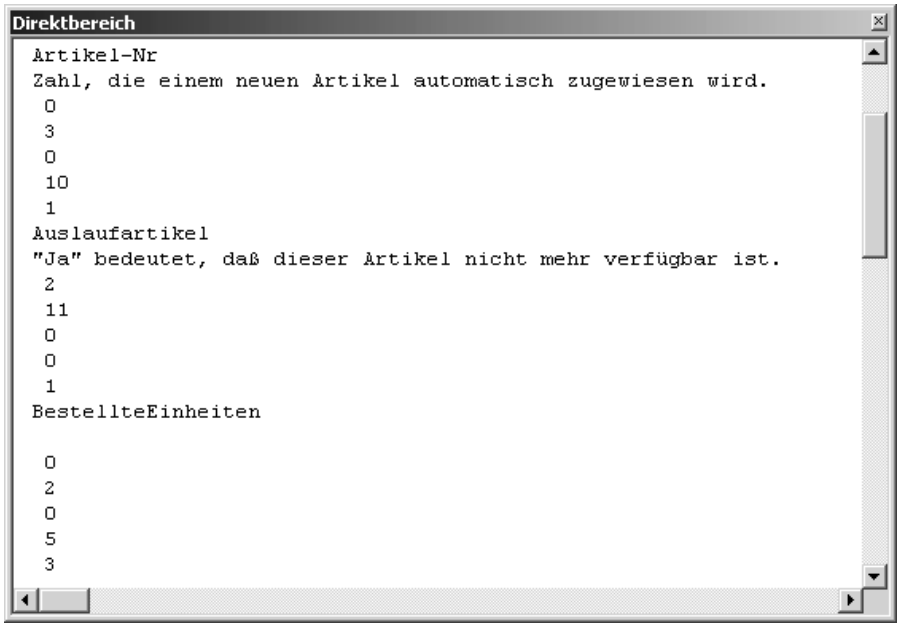


Abbildung 161: Die Tabelleneigenschaften wurden dokumentiert.

251 Tabellen erstellen

Mithilfe der Methode `Append` können Sie für eine Tabelle einzelne Datenfelder definieren. Die Syntax dieser Methode lautet:

```
Columns.Append Column [, Type] [, DefinedSize]
```

Mit dem Argument `Column` geben Sie den Namen der Spalte an, die erstellt und angehängt werden soll.

Über das Argument `Type` geben Sie den Datentyp der Spalte über eine Konstante oder einen numerischen Wert an. Entnehmen Sie die wichtigsten Konstanten aus der folgenden Tabelle.

Konstante	Beschreibung
`adSingle`	Gleitkommazahl mit einzelner Genauigkeit
`adDouble`	Gleitkommazahl mit doppelter Genauigkeit
`adCurrency`	Währungstyp
`adDecimal`	Dezimaler Varianttyp
`adInteger`	Integer Typ
`adBoolean`	Boolescher Varianttyp
`adChar`	Zeichenfolge mit fester Länge
`adVarChar`	Zeichenfolge mit variabler Länge
`adVarWChar`	Zeichenfolge mit Wide-variabler Länge
`adLongVarChar`	Zeichenfolge mit Long-variabler Länge

Tabelle 100: Die wichtigsten Datentypen der Methode Append

Neue Tabelle in aktueller Datenbank anlegen

Im folgenden Beispiel aus Listing 301 wird eine neue Tabelle mit dem Namen `Waren` erstellt. Dazu werden einige Felder definiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabelleErstellen()
   Dim ADOKAT As ADOX.Catalog
   Dim tabDef   As ADOX.Table

   On Error GoTo fehler
   Set ADOKAT = New ADOX.Catalog
   ADOKAT.ActiveConnection = CurrentProject.Connection
```

Listing 301: Neue Tabelle mit Feldern erstellen

```
    Set tabDef = New ADOX.Table

    With tabDef
       .Name = "Waren"
       Set .ParentCatalog = ADOKAT
       With .Columns
          .Append "WarenNr", adGUID
          .Item("WarenNr").Properties("Jet OLEDB:AutoGenerate") = True
          .Append "Bezeichnung", adVarWChar, 255
          .Append "Menge", adInteger
          .Append "Preis", adCurrency
       End With
    End With
    ADOKAT.Tables.Append tabDef

    ADOKAT.ActiveConnection = Nothing
    Set ADOKAT = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 301: Neue Tabelle mit Feldern erstellen (Forts.)

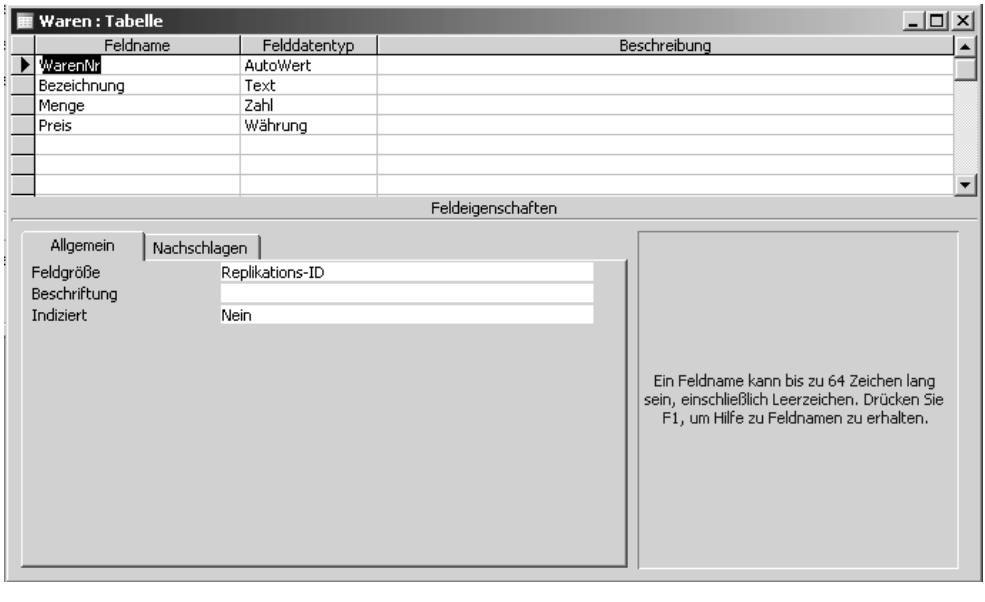


Abbildung 162: Die neue Tabelle wurde angelegt.

Eine Alternative zur Methode `Append` bietet die Methode `CreateField`.

Neue Tabelle in neuer Datenbank anlegen

Im folgenden Beispiel aus Listing 302 werden eine neue Datenbank sowie eine Tabelle angelegt, die folgende Felder aufweist: `Firma`, `Nachname`, `Vorname`, `E-Mail-Adresse` und `Telefonnummer`.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=================================================

Sub NeueDBAnlegenUndTabelleErstellen()
   Dim AccessObj As Object
   Dim DBS As Object
   Dim TabDef As Object
   Dim varTabfeld As Variant

   Set AccessObj = _
      CreateObject("Access.Application.11")
   AccessObj.NewCurrentDatabase "C:\Eigene Dateien\Adressen.mdb"

   Set DBS = AccessObj.CurrentDb

   Set TabDef = DBS.CreateTableDef("Kontakte")
   Set varTabfeld = TabDef.CreateField("Firma", 10, 20)
   TabDef.Fields.Append varTabfeld

    Set varTabfeld = TabDef.CreateField("Nachname", 10, 30)
    TabDef.Fields.Append varTabfeld

    Set varTabfeld = TabDef.CreateField("Vorname", 10, 20)
    TabDef.Fields.Append varTabfeld

    Set varTabfeld = TabDef.CreateField("E-Mail", 10, 30)
    TabDef.Fields.Append varTabfeld

    Set varTabfeld = TabDef.CreateField("Telefon", 10, 25)
    TabDef.Fields.Append varTabfeld
    DBS.TableDefs.Append TabDef

    Set AccessObj = Nothing
End Sub
```

Listing 302: Neue Datenbank und neue Tabelle anlegen

Erstellen Sie im ersten Schritt ein Access-Objekt. `Access.Application.11` bedeutet hier, dass die Access-2003-Objektbibliothek angesprochen wird. Über die Versionsnummer können Sie festlegen, welche Access-Bibliothek verwendet werden soll. So wird Access 2002 beispielsweise durch `Access.Application.10` und Access 2003 durch `Access.Application.11` identifiziert.

Mit der Methode `NewCurrentDatabase` erstellen Sie eine neue Access-Datenbank. Dabei müssen Sie den Pfad und den Dateinamen angeben. Im nächsten Schritt speichern Sie den Namen dieser neuen Datenbank in der Objektvariablen `DBS`. Setzen Sie für diesen Zweck die Methode `CurrentDb` ein. Erstellen Sie danach im Objekt `TabDef` eine neue Tabelle über die Methode `CreateTableDef`. Geben Sie dabei den Namen der Tabelle an.

Legen Sie die einzelnen Datenfelder für die Tabelle an und verwenden Sie für diese Aufgabe die Methode `CreateField`. Geben Sie dieser Methode bekannt, wie das Feld heißen soll, welchen Datentyp es bekommen soll und wie groß das Datenfeld werden darf. Über die Methode `Append` fügen Sie die so angelegten Tabellenfelder in Ihre Tabelle ein. Am Ende des Makros bauen Sie die Tabelle in Ihre Datenbank ein. Damit ist die Tabelle fest verankert. Heben Sie zuletzt die Objektvariablen über die Anweisung `Set AccessObj = Nothing` auf.

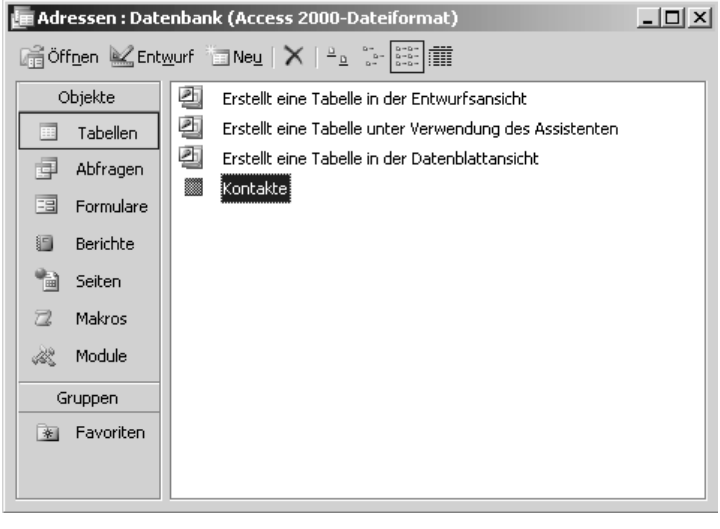


Abbildung 163: Die neue Tabelle wurde erfolgreich angelegt.

252 Tabellen verknüpfen

Über die `Properties`-Auflistung können Sie auf Tabelleneigenschaften zugreifen. Unter anderem können Sie diese Auflistung auch nützen, um externe Tabellen in Ihre Datenbank zu verlinken. Im folgenden Beispiel aus Listing 303 wird die externe Tabelle `Artikel` in die aktuell geöffnete Datenbank unter dem Namen `Kunden_Verknüpft` verknüpft.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================
```

Listing 303: Eine Tabelle verknüpfen

```
Sub VerknüpfteTabelleÖffnen()
   Dim ADOKAT As New ADOX.Catalog
   Dim TabDef As New ADOX.Table

   On Error GoTo fehler
   ADOKAT.ActiveConnection = CurrentProject.Connection
   TabDef.Name = "Kunden_Verknüpft"
   Set TabDef.ParentCatalog = ADOKAT
   TabDef.Properties("Jet OLEDB:Link Datasource") = _
      "C:\Eigene Dateien\Nordwind.mdb"
   TabDef.Properties("Jet OLEDB:Remote Table Name") = "Kunden"
   TabDef.Properties("Jet OLEDB:Create Link") = True
   ADOKAT.Tables.Append TabDef

   Set TabDef = Nothing
   Set ADOKAT = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 303: Eine Tabelle verknüpfen (Forts.)

Geben Sie im Argument `Link Datasource` den Namen sowie den Pfad der Datenbank an, in der sich die Tabelle befindet, die Sie verlinken möchten. Danach geben Sie den eigentlichen Namen der Tabelle im Argument `Remote Table Name` an. Im letzten Argument `Create Link` teilen Sie Ihre Absicht mit, die Tabelle zu verknüpfen. Über den Einsatz der Methode `Append` hängen Sie die Tabelle als Verknüpfung in die Datenbank.

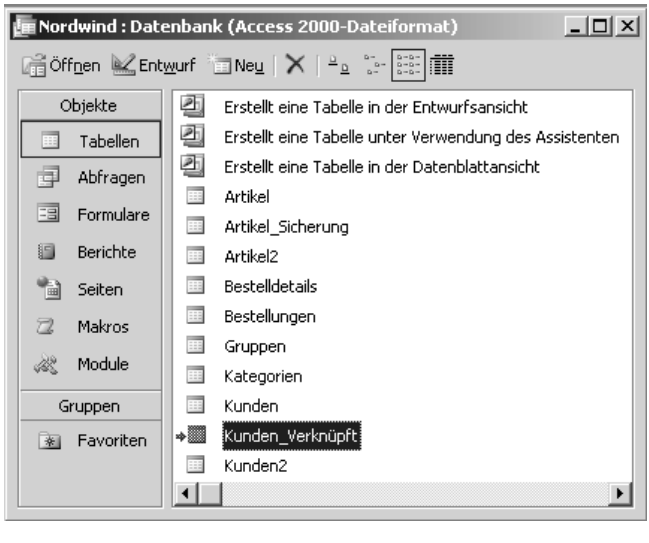


Abbildung 164: Die Tabelle Kunden wurde verknüpft und umbenannt.

253 Tabellen löschen

Über den Einsatz der Methode `Delete` können Sie eine Tabelle löschen.

Im folgenden Beispiel aus Listing 304 wird die Tabelle `Waren` gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabelleEntfernen()
   Dim ADOKAT As ADOX.Catalog
   Dim TabDef As ADOX.Table

   Set ADOKAT = New ADOX.Catalog
   ADOKAT.ActiveConnection = CurrentProject.Connection

   On Error GoTo fehler
   ADOKAT.Tables.Delete "Waren"
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 304: Eine Tabelle entfernen

254 Tabellenfeld anhängen (DAO)

Sie haben jederzeit die Möglichkeit, über ein Makro einer bestehenden Tabelle ein weiteres Feld hinzuzufügen. Im folgenden Beispiel aus Listing 305 wird der Tabelle `Personal` das Feld `Alter` hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub FeldAnhängen()
   Dim DBS As Database
   Dim FeldDef As TableDef
   Dim Feld As Field

   On Error GoTo fehler
   Set DBS = CurrentDb
```

Listing 305: Ein zusätzliches Feld anhängen

```
    Set FeldDef = DBS.TableDefs!Personal

    Set Feld = FeldDef.CreateField("Alter")
    Feld.Type = dbDouble
    FeldDef.Fields.Append Feld
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 305: Ein zusätzliches Feld anhängen (Forts.)

Mithilfe der Methode `CurrentDb` können Sie den Namen der aktiven Datenbank ermitteln. Greifen Sie danach auf das Objekt `TabDef` der Tabelle `Personal` zu und erzeugen Sie über die Methode `CreateField` ein neues Tabellenfeld. Geben Sie dieser Methode bekannt, wie das Feld heißen und welchen Datentyp es bekommen soll. Über die Methode `Append` fügen Sie die so angelegten Tabellenfelder der Tabelle hinzu.

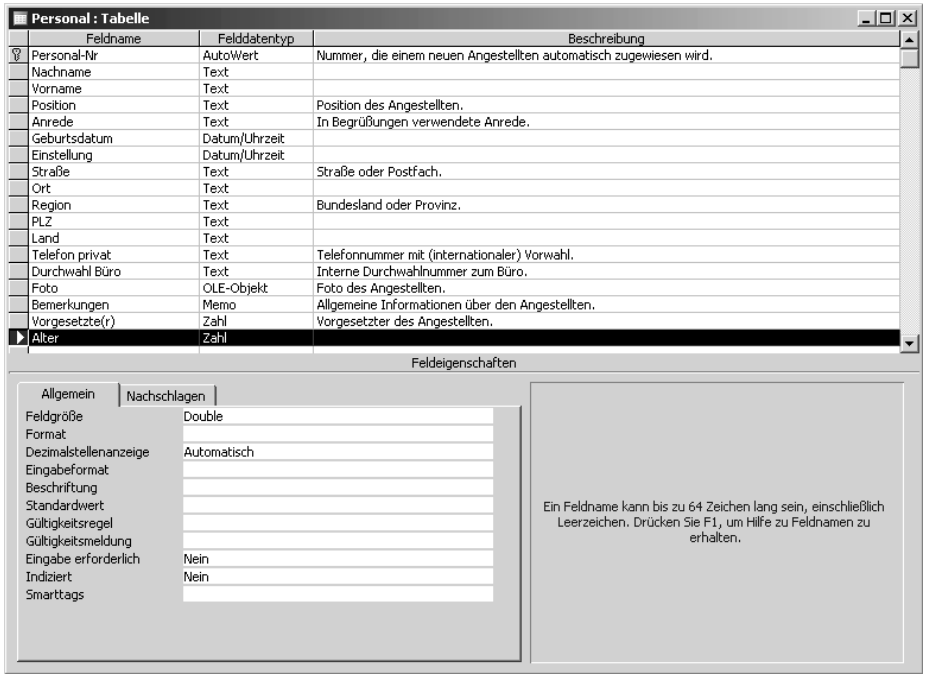


Abbildung 165: Das Feld wurde hinzugefügt.

255 Verlinken und Verknüpfen (DAO)

Welche Tabellen in eine Access-Datenbank verlinkt eingebunden sind, können Sie ganz schnell erkennen, wenn Sie im Register TABELLEN eine Tabelle mit einem Blockpfeil sehen (siehe Abbildung 166). Möchten Sie aber über ein Makro anzeigen lassen, welche Tabellen in Ihrer Datenbank eigentlich aus einer anderen Datenbank stammen und nur in Ihre eigene Datenbank verknüpft eingefügt wurden, dann starten Sie das Makro aus Listing 306.

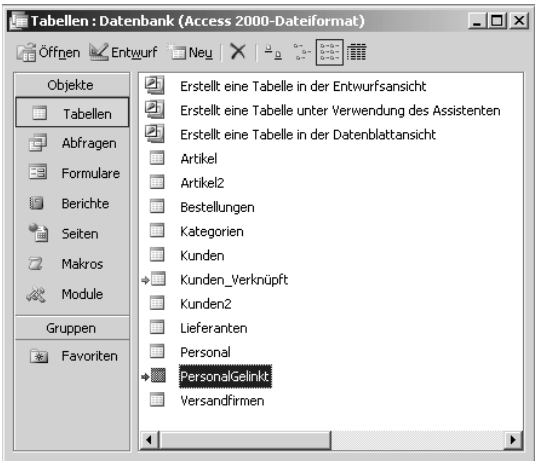


Abbildung 166: An den Blockpfeilen können verlinkte Tabellen erkannt werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub VerlinkteTabellenAnzeigen()
   Dim DBS As Database
   Dim intz As Integer
   Dim tdf As TableDef
   On Error GoTo fehler
   Set DBS = CurrentDb

   For intz = 0 To DBS.TableDefs.Count - 1
      Set tdf = DBS.TableDefs(intz)
      If Len(tdf.Connect) > 0 Then
         Debug.Print tdf.Name
      End If
   Next intz

   Set DBS = Nothing
   Exit Sub
```

Listing 306: Ermitteln, welche Tabellen extern verlinkt sind

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 306: Ermitteln, welche Tabellen extern verlinkt sind (Forts.)

Mithilfe des Objekts `TableDefs` haben Sie die Möglichkeit, alle in der Datenbank befindlichen Tabellen zu ermitteln. Über die Eigenschaft `Connect` können Sie herausbekommen, ob diese Tabellen verknüpft sind. Ist dies der Fall, dann erhalten Sie einen Wert > 0 zurück. In diesem Fall schreiben Sie diese Tabellen über den Befehl `Debug.Print` in den Direktbereich.

Abbildung 167: Die verlinkten Dateien wurden ausgegeben.

256 Verknüpfte Tabellen aktualisieren

Möchten Sie alle verlinkten Tabellen der aktuellen Datenbank aktualisieren, können Sie diese Aufgabe über das Makro aus Listing 307 lösen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=================================================

Sub VerlinkteTabellenAnktualisieren()
   Dim DBS As Database
   Dim intz As Integer
   Dim tdf As TableDef
   On Error GoTo fehler
   Set DBS = CurrentDb

   For intz = 0 To DBS.TableDefs.Count - 1
      Set tdf = DBS.TableDefs(intz)
      If Len(tdf.Connect) > 0 Then
         tdf.RefreshLink
      End If
   Next intz

   Set DBS = Nothing
   Exit Sub
```

Listing 307: Verlinkte Tabellen aktualisieren

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 307: Verlinkte Tabellen aktualisieren (Forts.)

Wenden Sie die Methode `RefreshLink` an, um die verknüpften Tabellen zu aktualisieren.

257 Verknüpfungsadresse anpassen

Ändert sich der Pfad bzw. das Laufwerk einer Datenbank, dann können Sie über folgendes Makro aus Listing 308 diese Pfade leicht anpassen. Hier soll das alte Verzeichnis *C:\Eigene Dateien* in *C:\Daten* geändert werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap04
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub PfadVerlinkterTabellenÄndern()
   Dim DBS As Database
   Dim tdf As TableDef

   On Error GoTo fehler
   Set DBS = CurrentDb

   For Each tdf In DBS.TableDefs
      If Len(tdf.Connect) > 0 Then
         tdf.Connect = ";DATABASE=" & "C:\Daten\Nordwind.mdb"
         tdf.RefreshLink
      End If
   Next tdf

   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 308: Pfade von verknüpften Tabellen anpassen

Mithilfe der Methode `RefreshLink` können Sie die Pfade der verlinkten Tabellen anpassen.

Abfragen programmieren

Im vorherigen Kapitel haben Sie komplexere Auswertungen von Tabellen vorgenommen und die Ergebnisse im Direktfenster der Entwicklungsumgebung ausgegeben. Wenn Sie Abfragen programmieren oder auch manuell erstellen, dann können Sie die Ergebnisse in so genannten »Views« oder Abfragetabellen speichern. Wie Sie Abfragen per VBA erstellen, das erfahren Sie in diesem Kapitel.

Bei der Programmierung von Abfragen steht Ihnen das Objekt `DoCmd` mit den Methoden `OpenQuery` und `RunSQL` zur Verfügung oder die Methode `Execute` des ADO-Objekts `Command`. Möchten Sie Abfragen auch als Abfragen speichern, dann können Sie auf das ADOX-Objekt `Catalog` zurückgreifen. Auch in diesem Kapitel kommen Abfragen auf Basis von DAO nicht zu kurz.

Die Abfragetypen von Access

Access bietet eine ganze Reihe von verschiedenen Abfragen an, die Sie manuell erstellen oder auch programmieren können. Sie verwenden Abfragen, um Daten auf mehrere Arten anzuzeigen, zu ändern und zu analysieren. Sie können sie auch als Datenquellen für Formulare, Berichte und Datenzugriffsseiten verwenden.

In Access unterscheidet man zwischen folgenden Abfragetypen:

1. AUSWAHLABFRAGEN: Mithilfe einer Auswahlabfrage rufen Sie Daten aus einer oder mehreren Tabellen ab und zeigen die Ergebnisse in einem Datenblatt an. Dort können Sie die Datensätze aktualisieren. Mit einer Auswahlabfrage können Sie auch Datensätze gruppieren und Summen, Anzahl, Durchschnittswerte und andere Werte berechnen.

2. PARAMETERABFRAGEN: Eine Parameterabfrage ist eine Abfrage, die beim Ausführen ein Dialogfeld zur Eingabe von Informationen anzeigt. Dies können Kriterien zum Abrufen von Datensätzen oder auch Werte sein, die in ein Feld eingefügt werden sollen. Sie können die Abfrage auch so entwerfen, dass nach mehreren Informationen gefragt wird.

3. KREUZTABELLENABFRAGEN: Sie verwenden Kreuztabellenabfragen, um Daten zur Vereinfachung von Analysen zu berechnen und neu zu strukturieren. Kreuztabellenabfragen berechnen eine Summe, einen Durchschnitt, eine Anzahl oder eine andere Funktion für Daten, die nach zwei Informationstypen gruppiert sind: entlang der linken Seite des Datenblatts und entlang der oberen Seite.

4. AKTIONSABFRAGEN: Diese Abfragen führen in nur einer Operation Änderungen an einer Vielzahl von Datensätzen durch bzw. verschieben diese. Unter den Aktionsabfragen unterscheidet man zwischen folgenden Typen:

    1. LÖSCHABFRAGEN: Mithilfe einer Löschabfrage löschen Sie eine Gruppe von Datensätzen aus einer oder mehreren Tabellen. Löschabfragen löschen immer vollständige Datensätze, nicht die in Datensätzen markierten Felder.

2. AKTUALISIERUNGSABFRAGEN: Eine Aktualisierungsabfrage führt globale Änderungen an Gruppen von Datensätzen in einer oder mehreren Tabellen durch. Mit einer Aktualisierungsabfrage können Sie Daten in vorhandenen Tabellen ändern.
3. ANFÜGEABFRAGEN: Eine Anfügeabfrage fügt eine Gruppe von Datensätzen aus einer oder mehreren Tabellen am Ende einer anderen Tabelle oder mehrerer Tabellen an.
4. TABELLENERSTELLUNGSABFRAGEN: Eine Tabellenerstellungsabfrage erstellt eine Tabelle aus allen oder einem Teil der Daten in einer oder mehreren Tabellen.
5. SQL-ABFRAGEN: Eine SQL-Abfrage ist eine Abfrage, die unter Verwendung einer SQL-Anweisung erstellt wird. Mit der Structured Query Language (SQL) können Sie relationale Datenbanken, wie z. B. Access, abfragen, aktualisieren und verwalten.

258 Abfragen starten mit OpenQuery

Über die Methode `OpenQuery` können Sie eine Auswahl- oder Kreuztabellenabfrage in der Datenblattansicht, Entwurfsansicht oder Seitenansicht öffnen. Diese muss natürlich schon in der Datenbank angelegt sein. Die Syntax dieser Funktion lautet:

```
OpenQuery(Abfragename, Ansicht, Datenmodus)
```

Argument	Beschreibung
`Abfragename`	Erforderlich. Legt den Namen der Abfrage fest, die Sie durchführen möchten.
`Ansicht`	Optional. Legt die Ansicht der Abfrage fest. Es stehen Ihnen dabei folgende Konstanten zur Verfügung: `acViewDesign`: Öffnet die Abfrage in der Entwurfsansicht. `acViewNormal`: Öffnet die Abfrage in gewohnter Weise in der Tabellenansicht (Standardeinstellung). `acViewPivotChart`: Stellt die Abfrage für ein Pivot-Diagramm zur Verfügung. `acViewPivotTable`: Hiermit können Sie die Felder der Abfrage für eine Pivot-Tabelle verwenden. `acViewPreview`: Zeigt die Abfrage in der Seitenansicht an.
`Datenmodus`	Optional. Legt fest, ob Änderungen an der Abfrage durchgeführt werden dürfen oder nicht. Dabei können Sie folgende Konstanten festlegen: `acAdd`: Der Anwender kann neue Datensätze hinzufügen, jedoch keine bestehenden Datensätze bearbeiten. `acEdit`: Der Anwender kann bestehende Datensätze bearbeiten und neue Datensätze hinzufügen (Standardeinstellung). `acReadOnly`: Der Anwender kann die Datensätze nur ansehen.

Tabelle 101: Die Argumente der Methode OpenQuery

Im folgenden Beispiel aus Listing 309 wird die Abfrage mit dem Namen `Artikel nach Kategorie` ausgeführt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub AbfrageStarten()
   On Error GoTo fehler
   DoCmd.OpenQuery "Artikel nach Kategorie", acViewNormal, acReadOnly
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 309: Abfrage starten

Die Methode `OpenQuery` greift auf die hinterlegten Tabellen `Artikel` und `Kategorie` zurück und liefert das Ergebnis der Abfrage in einer Tabelle, die unter der Rubrik ABFRAGEN hinterlegt ist.

Artikel nach Kategorie : Auswahlabfrage

Kategoriename	Artikelname	Liefereinheit	Lagerbestand	Auslaufartikel
Fleischprodukte	Pâté chinois	24 Kartons x 2 Pasteten	115	☐
Fleischprodukte	Tourtière	16 Pasteten	21	☐
Getränke	Chai	10 Kartons x 20 Beutel	39	☐
Getränke	Chang	24 x 12-oz-Flaschen	17	☐
Getränke	Chartreuse verte	750-ml-Flasche	69	☐
Getränke	Côte de Blaye	12 x 75-cl-Flaschen	17	☐
Getränke	Ipoh Coffee	16 x 500-g-Dosen	17	☐
Getränke	Lakkalikööri	500-ml-Flasche	57	☐
Getränke	Laughing Lumberjack Lager	24 x 12-oz-Flaschen	52	☐
Getränke	Outback Lager	24 x 355-ml-Flaschen	15	☐
Getränke	Outback Lager	24 x 355-ml-Flaschen	15	☐
Getränke	Rhönbräu Klosterbier	24 x 0,5-l-Flaschen	125	☐
Getränke	Sasquatch Ale	24 x 12-oz-Flaschen	111	☐
Getränke	Steeleye Stout	24 x 12-oz-Flaschen	20	☐

Datensatz: 1 von 70

Abbildung 168: Die Abfrage wurde ausgeführt und geöffnet.

259 Abfragen erstellen mit RunSQL

Eine weitere Möglichkeit, eine Abfrage in VBA zu erstellen und zu starten, ist die Methode `RunSQL`. Mit dieser Methode können Sie eine Microsoft Access-Aktionsabfrage ausführen, indem Sie die entsprechende SQL-Anweisung verwenden. Die Syntax dieser Methode lautet:

```
RunSQL(SQLAnweisung, TransaktionVerwenden)
```

Argument	Beschreibung
`SQLAnweisung`	Erforderlich. Legt die SQL-Anweisung fest. Speichern Sie diese SQL-Anweisung vorher am besten in einer String-Variablen und übergeben Sie diese dann der Methode `RunSQL`.
`TransaktionVerwenden`	Optional. Soll die Abfrage in eine Transaktion aufgenommen werden? Unter einer Transaktion versteht man eine Reihe von Änderungen, die an den Daten und am Schema einer Datenbank vorgenommen werden. Wenn ja, setzen Sie dieses Argument auf den Wert `True`.

Tabelle 102: Die Argumente der Methode RunSQL

In Verbindung mit der Methode `RunSQL` kommen einige SQL-Anweisungen zum Einsatz, die im Folgenden anhand von praktischen Beispielen beschrieben werden.

260 UPDATE (Aktualisierungsabfrage)

Mit der Anweisung `UPDATE` geben Sie bekannt, dass Sie eine Tabelle aktualisieren möchten. Die Syntax dieser Anweisung lautet:

```
UPDATE Tabellenname " & "SET FeldName = 'NeuerFeldInhalt' " & "WHERE FeldName =
'AlterFeldInhalt'"
Geben Sie über SET an, wie der neue Inhalt des Datenfelds lauten soll. Über WHERE
bestimmen Sie, nach welchen Sätzen in der Tabelle gesucht werden soll.
```

Im Beispiel aus Listing 310 werden in der Tabelle `Artikel2` alle Bestände auf den Wert 0 gesetzt. Dabei wird die Meldung, wie viele Datensätze aktualisiert werden sollen deaktiviert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
```

Listing 310: Mit UPDATE Bestände initialisieren

```
Sub BestandInitialisieren()
   On Error GoTo fehler
   DoCmd.SetWarnings False
   DoCmd.RunSQL "UPDATE Artikel2 SET Artikel2.Lagerbestand = 0;"
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 310: Mit UPDATE Bestände initialisieren (Forts.)

Über die Methode `SetWarnings`, die auf den Wert `False` gesetzt wird, können Standardmeldungen, wie z.B. die Aktualisierungsmeldung, unterdrückt werden. Übergeben Sie danach die SQL-Anweisung der Methode `RunSql`.

Artikel2 : Tabelle

Artikel-Nr	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbestand
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	18,62 €	0
6	Grandma's Boysenberry S	Grandma Kelly's Hor	Gewürze	12 x 8-oz-Gläser	25,86 €	0
7	Uncle Bob's Organic Drie	Grandma Kelly's Hor	Naturprodukte	12 x 1-lb-Packungen	31,04 €	0
8	Northwoods Cranberry Sa	Grandma Kelly's Hor	Gewürze	12 x 12-oz-Gläser	41,38 €	0
10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200-ml-Gläser	32,07 €	0
11	Queso Cabrales	Cooperativa de Ques	Milchprodukte	1-kg-Paket	21,73 €	0
13	Konbu	Mayumi's	Meeresfrüchte	2-kg-Karton	6,21 €	0
14	Tofu	Mayumi's	Naturprodukte	40 x 100-g-Packungen	24,05 €	0
16	Pavlova	Pavlova, Ltd.	Süßwaren	32 x 500-g-Kartons	18,05 €	0
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte	16-kg-Paket	64,66 €	0
19	Teatime Chocolate Biscu	Specialty Biscuits, L	Süßwaren	10 Kartons x 12 Stück	9,52 €	0

Datensatz: 1 von 62

Abbildung 169: Der Bestand wurde für alle Artikel auf 0 gesetzt.

Im nächsten Beispiel aus Listing 311 wird ein Lieferantenwechsel vorgenommen. Dabei soll der Lieferant `New Orleans Cajun Delights` (`Lieferanten-Nr` = 2) in Zukunft häufiger eingesetzt werden und die Lieferanten `Grandma Kelly's Homestead` (`Lieferanten-Nr` = 3) und `Tokyo Traders` (`Lieferanten-Nr` = 4) ablösen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
Sub LieferantenWechsel()
   Dim strSQL As String

   strSQL = "UPDATE Artikel2 " & _
      "SET Artikel2.Lieferanten_Nr = 2 " & _
      "WHERE Artikel2.Lieferanten_Nr = 3 OR Artikel2.Lieferanten_Nr = 4"
```

Listing 311: Lieferantenwechsel durchführen

```
    On Error GoTo fehler
    DoCmd.SetWarnings False
    DoCmd.RunSQL strSQL
    DoCmd.SetWarnings True
    Exit Sub

    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 311: Lieferantenwechsel durchführen (Forts.)

261 INSERT INTO (Anfügeabfrage)

Über die SQL-Anweisung `INSERT INTO` geben Sie bekannt, dass Sie in einer Tabelle einen neuen Datensatz anfügen möchten. Die Syntax lautet:

```
INSERT INTO ZielTabelle IN 'NameZielDatenbank' SELECT Tabellenfeld(er) FROM
QuellTabelle ;"
```

Über `IN` geben Sie den Namen der Zieldatenbank an. Bestimmen Sie danach über die Anweisung `SELECT` die Datenfelder der Tabelle, die Sie anfügen möchten. Achten Sie dabei darauf, dass die Feldnamen der Tabelle mit Ihren Angaben übereinstimmen. Geben Sie nach `FROM` den Namen der Tabelle an, aus der die Daten herausgeholt werden sollen.

```
INSERT INTO ZielTabelle IN 'NameZielDatenbank' SELECT Tabellenfeld(er) VALUES
,'Inhalte der Felder'"
```

Über die SQL-Anweisung `INSERT INTO` geben Sie bekannt, dass Sie in einer Tabelle einen Satz anfügen möchten. Geben Sie danach die Datenfelder der Tabelle über die Anweisung `SELECT` an, die Sie anfügen möchten. Achten Sie dabei, darauf, dass die Feldnamen der Tabelle mit Ihren Angaben übereinstimmen. Erfassen Sie nach `VALUES` die tatsächlichen Werte in Textform. Betten Sie jede Information in einfache Anführungszeichen ein. Trennen Sie jede Information durch Kommas voneinander ab.

Im Beispiel aus Listing 312 wird der Tabelle `Personal` ein neuer Mitarbeiter hinzugefügt.

```
 '=======================================================
 ' Auf CD     Buchdaten\Beispiele\Kap05
 ' Dateiname  Abfragen.mdb
 ' Modul      mdlDoCmd
 '=======================================================
 Sub MitarbeiterHinzufügen()
    Dim strSQL As String

    On Error GoTo fehler
```

Listing 312: Neuen Datensatz mit INSERT INTO hinzufügen

```
    strSQL = "INSERT INTO Personal(Nachname, Vorname, " & _
       "Position, Anrede, Geburtsdatum) VALUES " & _
       "('Held', 'Bernd', 'Externer Berater', 'Herr', '02.04.1969')"

    DoCmd.SetWarnings False
    DoCmd.RunSQL strSQL
    DoCmd.SetWarnings True
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 312: Neuen Datensatz mit INSERT INTO hinzufügen (Forts.)

Personal : Tabelle

Personal-Nr	Nachname	Vorname	Position	Anrede	Geburtsdatum
1	Davolio	Nancy	Vertriebsmitarbeiterin	Frau	08.Dez.1968
2	Fuller	Andrew	Geschäftsführer	Herr	19.Feb.1952
3	Leverling	Janet	Vertriebsmitarbeiterin	Frau	30.Aug.1963
4	Peacock	Margaret	Vertriebsmitarbeiterin	Frau	19.Sep.1958
5	Buchanan	Steven	Vertriebsmanager	Herr	04.Mrz.1955
6	Suyama	Michael	Vertriebsmitarbeiter	Herr	02.Jul.1963
7	King	Robert	Vertriebsmitarbeiter	Dr.	29.Mai.1960
8	Callahan	Laura	Vertriebskoordinatorin	Frau	09.Jan.1958
9	Dodsworth	Anne	Vertriebsmitarbeiterin	Frau	02.Jul.1969
21	Smith	Eva	Vertriebsmitarbeiterin	Frau	08.Dez.1969
22	Wels	Berhard	Vertriebsmitarbeiter	Herr	01.Feb.1959
23	Russel	Brenda	Vertriebsmitarbeiterin	Frau	30.Aug.1970
25	Held	Bernd	Externer Berater	Herr	02.Apr.1969
(AutoWert)					

Datensatz: 10 von 13

Abbildung 170: Der neue Datensatz wurde am Ende der Tabelle eingefügt.

Im folgenden Beispiel aus Listing 313 wird die Tabelle `NeueKunden` aus der Datenbank *Abfragen.mdb* in die Datenbank *Nordwind.mdb* in die Tabelle `Kunden` geschrieben. Sehen Sie sich zunächst die Tabelle `NeueKunden` in Abbildung 171 an.

Abbildung 171: Diese Kunden sollen in eine bereits bestehende Tabelle überführt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub KundenÜbertragen()
   Dim strSQL As String

   On Error GoTo fehler
   DoCmd.SetWarnings False
   strSQL = "INSERT INTO Kunden IN 'C:\Eigene Dateien\Nordwind.mdb' " & _
      " SELECT NeueKunden.* FROM  NeueKunden ;"
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings False
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 313: Neue Kunden einlesen

262 DELETE (Löschabfrage)

Mithilfe der SQL-Anweisung `DELETE` können Sie Datensätze aus einer Tabelle entfernen. Die Syntax dieser Anweisung lautet:

```
DELETE * FROM Tabellename WHERE Datenfeld='xxx'
```

Setzen Sie die SQL-Anweisung `DELETE` ein, um eine Löschabfrage zu starten. Geben Sie dabei an, welche Felder Sie aus welcher Tabelle löschen möchten. Geben Sie unter WHERE das Löschkriterium bekannt.

Im folgenden Beispiel aus Listing 314 werden aus der Tabelle `Lieferanten` alle Lieferanten entfernt, die aus Japan kommen.

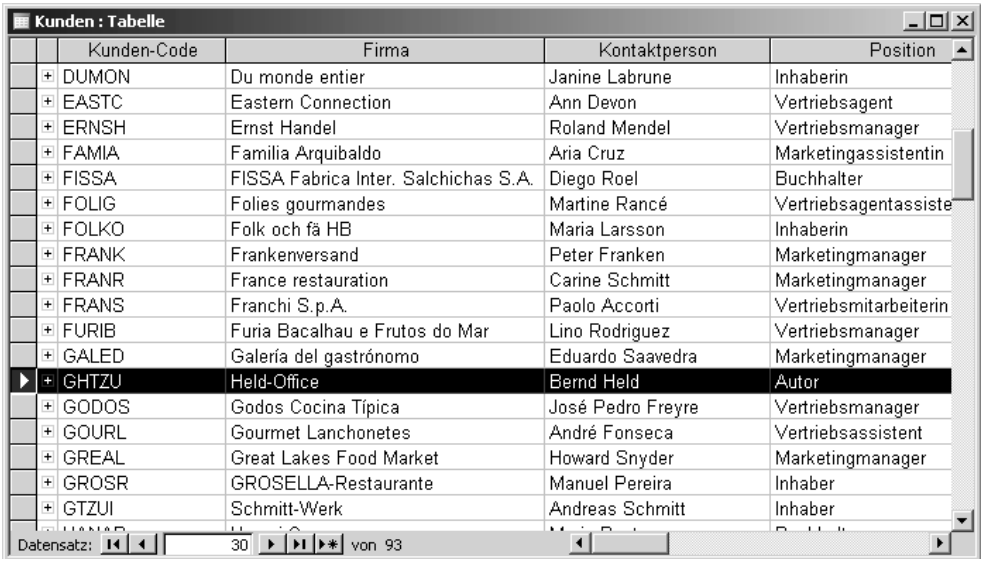


Abbildung 172: Die neuen Kunden wurden übertragen und einsortiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub DatensätzeLöschen2()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "DELETE * FROM Lieferanten WHERE Land='Japan'"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 314: Datensätze löschen mit DELETE

Geben Sie bei der `WHERE`-Klausel das Kriterium in Apostrophen an, sofern es sich um alphanumerische Vergleichsfelder handelt. Die Apostrophe sind nicht nötig, wenn es sich um numerische Vergleichsfelder wie beispielsweise `Lagerbestand`, `Preis` usw. handelt.

Über die LIKE-Klausel können Sie beispielsweise alle Datensätze löschen, die mit dem Buchstaben F beginnen, also Länder wie Frankreich und Finnland. Das Makro für diesen Zweck sehen Sie in Listing 315.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub DatensätzeLöschen3()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "DELETE * FROM Lieferanten3 WHERE Land LIKE 'F*'"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 315: Länder mit dem Anfangsbuchstaben »F« löschen

Vorsicht ist beim nächsten Beispiel aus Listing 316 geboten. Die Aufgabe besteht darin, alle Milchprodukte aus der Tabelle Artikel2 zu entfernen. Diese Produkte werden durch die Kategorienummer 4 repräsentiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub KategorieEntfernen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "DELETE * FROM Artikel2 WHERE [Kategorie-Nr] = 4;"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub
```

Listing 316: Datensätze löschen über eine Nummer

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 316: Datensätze löschen über eine Nummer (Forts.)

Enthält der Feldnamen Sonderzeichen wie beispielsweise das Minuszeichen, dann müssen Sie den Feldnamen in eckige Klammern setzen, damit das Makro nicht abstürzt.

263 SELECT INTO (Tabellenerstellungsabfrage)

Mithilfe der SQL-Anweisung `SELECT INTO` können Sie eine neue Tabelle erstellen und dabei bestimmte Daten aus einer anderen Tabelle einfügen. Die Syntax dieser Anweisung lautet:

```
SELECT Datenfelder INTO [NeueTabelle] FROM QuellTabelle
WHERE (Datenfeld = 'xxx')
```

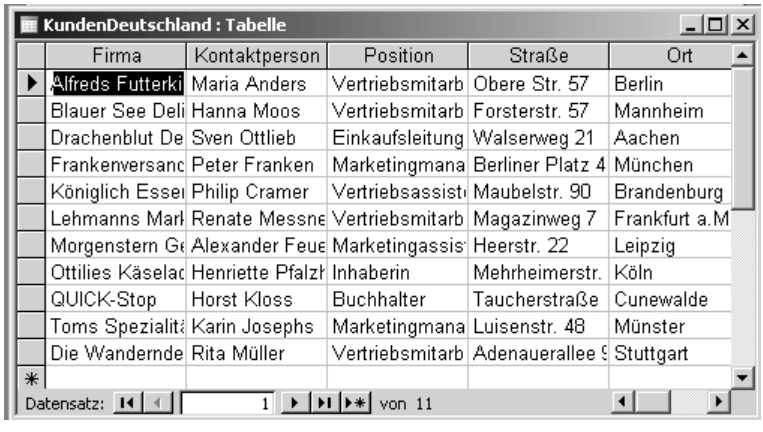
Über die SQL-Anweisung `SELECT` wählen Sie die Felder aus, die Sie in der neuen Tabelle speichern möchten. Bei der Vergabe der Feldnamen können Sie bis zu 64 Zeichen für einen Namen verwenden. Über `INTO` geben Sie den Namen der Tabelle an, die angelegt werden soll. Danach legen Sie mit der SQL-Anweisung `FROM` fest, woher die Daten stammen. Mit der `WHERE`-Anweisung können Sie die zu übertragenden Datensätze einschränken.

Im folgenden Beispiel aus Listing 317 werden aus der Tabelle `Kunden` alle Kunden ermittelt, die aus Deutschland stammen, und in die Tabelle `KundenDeutschland` geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub NeueTabelleErstellen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "SELECT Firma, Kontaktperson, Position, Straße, Ort " & _
      "INTO [KundenDeutschland] FROM Kunden WHERE (Land = 'Deutschland')"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 317: Neue Tabelle erstellen über SELECT INTO

Geben Sie gleich hinter der Anweisung `SELECT` die Felder an, die Sie in der neuen Tabelle haben möchten. Danach geben Sie hinter der Klausel `INTO` den Namen für die neue Tabelle an. Die Klausel `FROM` sagt aus, aus welcher Tabelle die Daten gezogen werden sollen. Über die `WHERE`-Klausel wird eine Einschränkung vorgenommen, so dass nur Kunden aus Deutschland übernommen werden sollen.



KundenDeutschland : Tabelle

Firma	Kontaktperson	Position	Straße	Ort
Alfreds Futterki	Maria Anders	Vertriebsmitarb	Obere Str. 57	Berlin
Blauer See Deli	Hanna Moos	Vertriebsmitarb	Forsterstr. 57	Mannheim
Drachenblut De	Sven Ottlieb	Einkaufsleitung	Walserweg 21	Aachen
Frankenversanc	Peter Franken	Marketingmana	Berliner Platz 4	München
Königlich Esser	Philip Cramer	Vertriebsassist	Maubelstr. 90	Brandenburg
Lehmanns Mark	Renate Messne	Vertriebsmitarb	Magazinweg 7	Frankfurt a.M
Morgenstern Ge	Alexander Feue	Marketingassis	Heerstr. 22	Leipzig
Ottilies Käselac	Henriette Pfalzh	Inhaberin	Mehrheimerstr.	Köln
QUICK-Stop	Horst Kloss	Buchhalter	Taucherstraße	Cunewalde
Toms Spezialit	Karin Josephs	Marketingmana	Luisenstr. 48	Münster
Die Wandernde	Rita Müller	Vertriebsmitarb	Adenauerallee 9	Stuttgart

Datensatz: 1 von 11

Abbildung 173: Alle Kunden aus Deutschland in einer separaten Tabelle

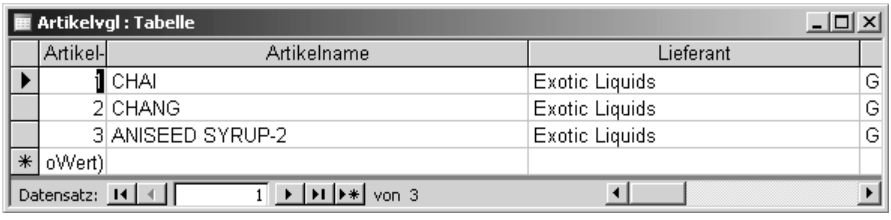
264 INNER JOIN (Vergleichsabfrage)

Über die SQL-Anweisung `INNER JOIN` können Sie Tabellen miteinander vergleichen und nur diejenigen Sätze weiterverarbeiten, die in den Tabellen gleich sind. Die Syntax dieser Anweisung lautet:

```
SELECT Datenbankfelder INTO NameNeuerTabelle FROM Tabelle1 INNER JOIN Tabelle2 ON
Tabelle1.Feld1 = Tabelle2.Feld1
```

Nach der Anweisung `SELECT` geben Sie die Datenfelder an, die in der neuen Tabelle ausgegeben werden sollen. Nach der Anweisung `INTO` folgt der Name der neuen Tabelle, die angelegt werden soll. Direkt hinter der Anweisung `FROM` geben Sie den Namen der ersten Tabelle an. Hinter der Anweisung `INNER JOIN` geben Sie den Namen der Vergleichstabelle an, die mit der ersten Tabelle abgeglichen werden soll. Nach der Anweisung `ON` erfassen Sie das Vergleichskriterium.

Im folgenden Beispiel aus Listing 318 werden die Felder `Artikelname` und `Einzelpreis` in die neue Tabelle `GleicheArtikel` übertragen. Es werden dabei nur Datensätze übertragen, die in beiden Tabellen `Artikel` sowie `Artikelvgl` vorhanden sind.

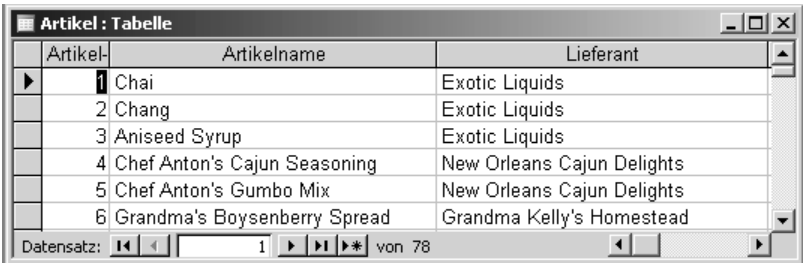


Artikelvgl : Tabelle

Artikel-	Artikelname	Lieferant	
1	CHAI	Exotic Liquids	G
2	CHANG	Exotic Liquids	G
3	ANISEED SYRUP-2	Exotic Liquids	G
oWert)			

Datensatz: 1 von 3

Abbildung 174: Vergleichsartikel in der Tabelle Artikelvgl



Artikel : Tabelle

Artikel-	Artikelname	Lieferant
1	Chai	Exotic Liquids
2	Chang	Exotic Liquids
3	Aniseed Syrup	Exotic Liquids
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead

Datensatz: 1 von 78

Abbildung 175: Die Ausgangstabelle Artikel

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabellenVergleichen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = _
      "SELECT Artikel.Artikelname, Artikel.Einzelpreis INTO GleicheArtikel " & _
      "FROM Artikel INNER JOIN ArtikelVgl ON Artikel.Artikelname =
      Artikelvgl.Artikelname"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 318: Tabellen vergleichen mit INNER JOIN

Geben Sie hinter der Anweisung `SELECT` die Felder an, die Sie in der neuen Tabelle haben möchten. Bei der `INTO`-Klausel geben Sie den Namen der neuen Tabelle an. Die `FROM`-Klausel enthält den Namen der ersten Tabelle, also der Tabelle `Artikel`. Hinter der Anweisung `INNER`

JOIN geben Sie den Namen der Vergleichstabelle an, die mit der ersten Tabelle abgeglichen werden soll. Nach der Anweisung ON erfassen Sie das Vergleichskriterium, hier der Artikelname der Tabelle Artikel sowie der Artikelname der Tabelle Artikelvgl.

Die SQL-Anweisung INNER JOIN unterscheidet nicht zwischen Groß- und Kleinschreibung.

Abbildung 176: Nur zwei Artikel sind identisch.

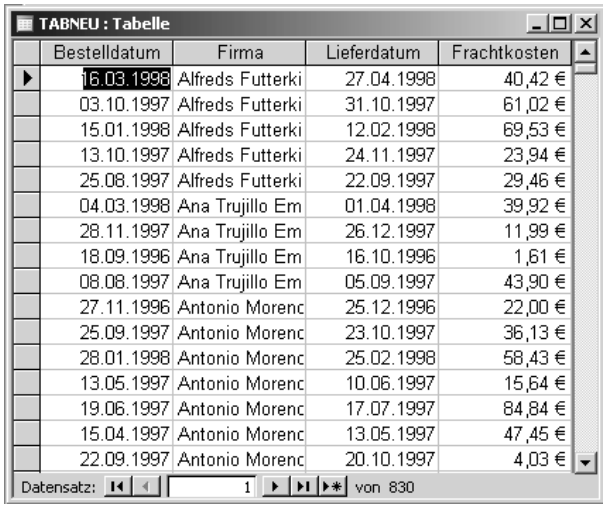
Im folgenden Beispiel aus Listing 319 werden die Tabellen Bestellungen und Kunden miteinander verarbeitet. Über den gemeinsamen eindeutigen Schlüssel Kunden-Code können Sie diese Aufgabe über den Einsatz der SQL-Anweisung INNER JOIN lösen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=================================================

Sub MehrereTabellenChecken()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "SELECT Bestellungen.Bestelldatum, " & _
      "Kunden.Firma, Bestellungen.Lieferdatum, " & _
      "Bestellungen.Frachtkosten INTO TABNEU FROM " & _
      "Bestellungen INNER JOIN Kunden " & _
      "ON (Bestellungen.[Kunden-Code] =" & _
      "Kunden.[Kunden-Code])"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 319: Daten abgleichen

Geben Sie hinter der Anweisung SELECT die Felder an, die Sie in der neuen Tabelle haben möchten. Achten Sie dabei darauf, dass Sie genau angeben, aus welcher Tabelle Access die

Daten ziehen soll. Bei der `INTO`-Klausel geben Sie den Namen der neuen Tabelle an. Die `FROM`-Klausel enthält den Namen der ersten Tabelle, also der Tabelle `Bestellungen`. Hinter der Anweisung `INNER JOIN` geben Sie den Namen der Vergleichstabelle an, die mit der ersten Tabelle abgeglichen werden soll. Nach der Anweisung `ON` erfassen Sie das Vergleichskriterium, hier der `Kunden-Code`.



TABNEU : Tabelle

Bestelldatum	Firma	Lieferdatum	Frachtkosten
16.03.1998	Alfreds Futterki	27.04.1998	40,42 €
03.10.1997	Alfreds Futterki	31.10.1997	61,02 €
15.01.1998	Alfreds Futterki	12.02.1998	69,53 €
13.10.1997	Alfreds Futterki	24.11.1997	23,94 €
25.08.1997	Alfreds Futterki	22.09.1997	29,46 €
04.03.1998	Ana Trujillo Em	01.04.1998	39,92 €
28.11.1997	Ana Trujillo Em	26.12.1997	11,99 €
18.09.1996	Ana Trujillo Em	16.10.1996	1,61 €
08.08.1997	Ana Trujillo Em	05.09.1997	43,90 €
27.11.1996	Antonio Morenc	25.12.1996	22,00 €
25.09.1997	Antonio Morenc	23.10.1997	36,13 €
28.01.1998	Antonio Morenc	25.02.1998	58,43 €
13.05.1997	Antonio Morenc	10.06.1997	15,64 €
19.06.1997	Antonio Morenc	17.07.1997	84,84 €
15.04.1997	Antonio Morenc	13.05.1997	47,45 €
22.09.1997	Antonio Morenc	20.10.1997	4,03 €

Datensatz: 1 von 830

Abbildung 177: Alle Bestellungen aus der Tabellen Bestellungen und Kunden zusammengestellt

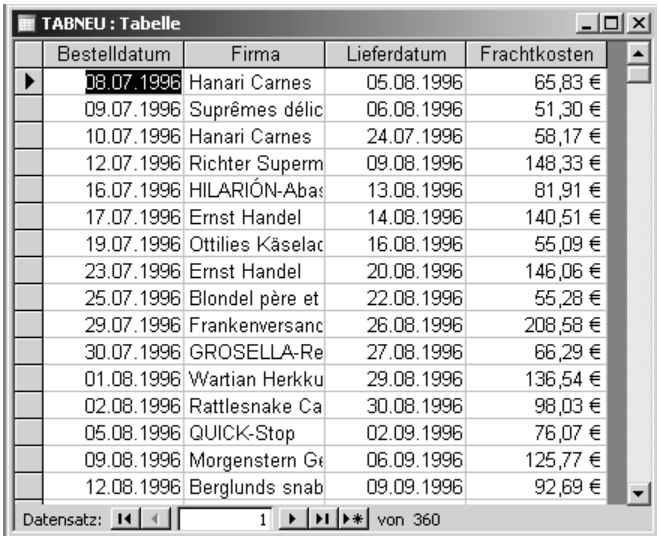
Soll zusätzlich noch eine weitere Abfrage integriert werden, beispielsweise alle Frachtkosten, die höher sind als 50 Euro, dann starten Sie das Makro aus Listing 320.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub FrachtkostenÜber50()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "SELECT Bestellungen.Bestelldatum, " & _
      "Kunden.Firma, Bestellungen.Lieferdatum, " & _
      "Bestellungen.Frachtkosten INTO TABNEU FROM " & _
      "Bestellungen INNER JOIN Kunden " & _
      "ON (Bestellungen.[Kunden-Code] =" & _
      "Kunden.[Kunden-Code]) WHERE " & _
      "Bestellungen.Frachtkosten > 50"

   DoCmd.SetWarnings False
```

Listing 320: Eine weitere Einschränkung wurde hinzugefügt.

```
    DoCmd.RunSQL strSQL
    DoCmd.SetWarnings True
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 320: Eine weitere Einschränkung wurde hinzugefügt. (Forts.)



TABNEU : Tabelle

Bestelldatum	Firma	Lieferdatum	Frachtkosten
08.07.1996	Hanari Carnes	05.08.1996	65,83 €
09.07.1996	Suprêmes délic	06.08.1996	51,30 €
10.07.1996	Hanari Carnes	24.07.1996	58,17 €
12.07.1996	Richter Superm	09.08.1996	148,33 €
16.07.1996	HILARIÓN-Abas	13.08.1996	81,91 €
17.07.1996	Ernst Handel	14.08.1996	140,51 €
19.07.1996	Ottilies Käselac	16.08.1996	55,09 €
23.07.1996	Ernst Handel	20.08.1996	146,06 €
25.07.1996	Blondel père et	22.08.1996	55,28 €
29.07.1996	Frankenversanc	26.08.1996	208,58 €
30.07.1996	GROSELLA-Re	27.08.1996	66,29 €
01.08.1996	Wartian Herkku	29.08.1996	136,54 €
02.08.1996	Rattlesnake Ca	30.08.1996	98,03 €
05.08.1996	QUICK-Stop	02.09.1996	76,07 €
09.08.1996	Morgenstern Ge	06.09.1996	125,77 €
12.08.1996	Berglunds snab	09.09.1996	92,69 €

Datensatz: 1 von 360

Abbildung 178: Nur Frachtkosten über 50 € werden berücksichtigt.

265 UNION (Auswahlabfrage)

Über den Einsatz der SQL-Anweisung `UNION` können Sie mehrere Tabellen zusammenführen, die denselben Datensatzaufbau haben. Die Syntax lautet:

```
SELECT * INTO Zieltabelle FROM (SELECT * FROM ARTIKEL UNION ALL SELECT * FROM
Artikelvgl
```

Mit der SQL-Anweisung `SELECT *` holen Sie alle Sätze aus den danach angegebenen Tabellen. Nach dem Argument `INTO` geben Sie den Namen der Zieltabelle an. Rechts und links neben der Anweisung `UNION` geben Sie die Namen der Tabellen an, die Sie vereinen möchten.

Im folgenden Beispiel aus Listing 321 werden die beiden Tabellen `Artikel` und `ArtikelNEU` zusammengefasst und nach dem Artikelnamen sortiert.

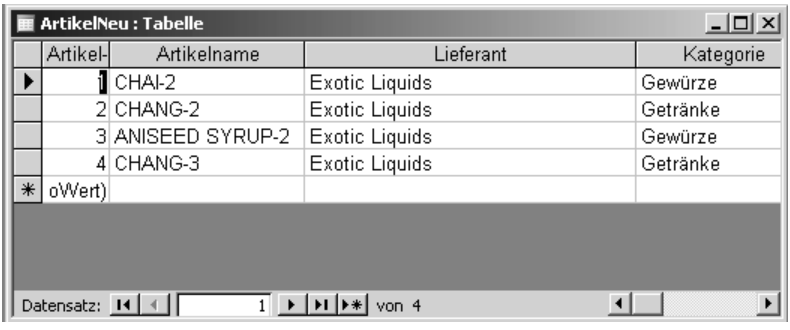


Abbildung 179: Die Tabelle mit den neuen Artikeln

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabellenZusammenwerfen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "SELECT * INTO ErgebnisTab " & _
      "FROM(SELECT * FROM ARTIKEL " & _
      "UNION ALL SELECT * FROM Artikelneu " & _
      "ORDER BY Artikelname)"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 321: Tabellen konsolidieren mit UNION

Im Makro aus Listing 321 wurden alle Datenfelder der beiden Tabellen in der Ergebnistabelle gespeichert. Sollen jedoch nur bestimmte Felder übertragen werden, dann starten Sie das Makro aus Listing 322. In diesem Makro werden nur die Felder `Artikelname`, `Einzelpreis` und `Lagerbestand` in die Tabelle `ErgebnisTab` übertragen.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'====================================================

Sub TabellenZusammenwerfenAuswahl()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "SELECT Artikelname, Einzelpreis, Lagerbestand INTO ErgebnisTab " & _
      "FROM(SELECT Artikelname, Einzelpreis, Lagerbestand FROM ARTIKEL " & _
      "UNION ALL SELECT Artikelname, Einzelpreis, Lagerbestand FROM Artikelneu " &
      "ORDER BY Artikelname);"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 322: Eine Auswahl einiger Felder beider Tabellen in die Zieltabelle übertragen

Nach der SQL-Anweisung `SELECT` geben Sie die Felder an, die Sie in der Ergebnistabelle haben möchten. Nach dem Argument `INTO` geben Sie den Namen der Zieltabelle an. Rechts und links neben der Anweisung `UNION` nennen Sie die Namen der Tabellen, die Sie vereinen möchten. Über die `ORDER`-Klausel sortieren Sie die Ergebnistabelle nach dem Artikelnamen.

266 Abfragen mit Berechnungen ausführen

Wie schon im vorherigen Kapitel angesprochen, können mathematische Funktionen wie Summe, Mittelwert und andere aus der Tabelle 103 in SQL-Anweisungen eingebaut werden.

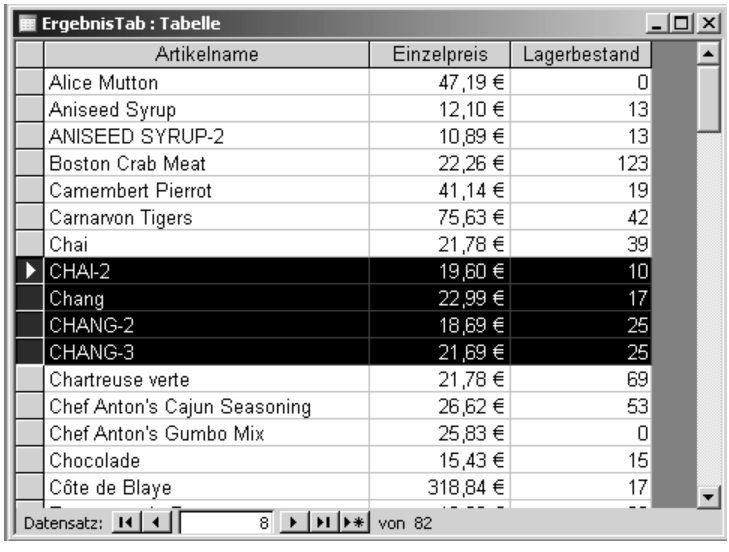


Abbildung 180: Es wurden nur die wichtigsten Felder übertragen.

Funktion	Beschreibung
SUM	Summe
AVG	Mittelwert
MIN	Kleinster Wert
MAX	Größter Wert
COUNT	Anzahl
STDDEV	Standardabweichung
VAR	Varianz
FIRST	Erster Satz
LAST	Letzter Satz

Tabelle 103: Mathematische Funktionen für SQL-Anweisungen

Im folgenden Beispiel aus Listing 323 sollen die Bestellungen pro Hersteller aus der Tabelle `Bestellungen` gezählt und in einer Ergebnistabelle ausgegeben werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
```

Listing 323: Datensätze zählen über die Funktion COUNT

```
Sub DatensätzeZählenAbfrage()
   Dim strSQL As String
   On Error GoTo fehler

   strSQL = _
      "SELECT COUNT(Frachtkosten) AS ERG, " & _
      "Bestellungen.[Kunden-Code] INTO Ergebnis " & _
      " FROM Bestellungen GROUP BY " & _
      " Bestellungen.[Kunden-Code]"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 323: Datensätze zählen über die Funktion COUNT (Forts.)

Über die Funktion `COUNT` werden alle Bestellungen pro Kunde ermittelt. Dabei werden die gezählten Bestellungen im Feld `ERG` der Tabelle `Ergebnis` ausgegeben.

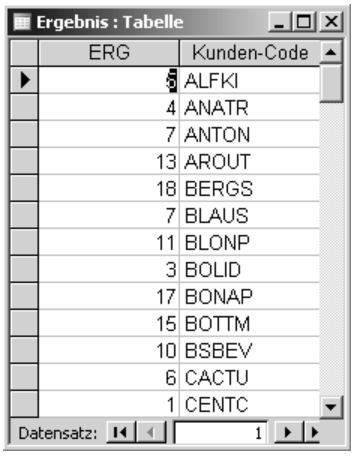


Abbildung 181: Die Anzahl der Bestellungen pro Kunde werden dargestellt.

Im nächsten Beispiel aus Listing 324 werden aus der Tabelle `Artikel` die Lagermengen pro Hersteller summiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub LagermengenSummieren()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = _
      "SELECT SUM(Artikel.Lagerbestand) " & _
      "AS SUMBestand, Artikel.[Lieferanten_Nr] INTO ErgebnisLager " & _
      " FROM Artikel GROUP BY Artikel.[Lieferanten_Nr]"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 324: Lagermengen pro Hersteller summieren

Über die Funktion `SUM` werden alle Lagerbestände pro Lieferant ermittelt. Dabei werden die summierten Lagerbestände im Feld `SUMBestand` der Tabelle `Ergebnislager` ausgegeben.

ErgebnisLager : Tabelle

SUMBestand	Lieferanten_Nr
69	1
133	2
141	3
64	4
108	5
98	6
125	7
74	8
165	9
20	10
140	11
205	12
10	13
23	14
164	15
183	16
224	17

Datensatz: 1

Abbildung 182: Die Lagerbestände der einzelnen Lieferanten

267 ALTER (Tabellenänderungsabfrage)

Um eine bereits vorhandene Tabelle zu ändern, können Sie die SQL-Anweisung `ALTER TABLE` einsetzen. Dabei müssen Sie genau angeben, was Sie konkret mit der Tabelle anstellen möchten. Dafür stehen Ihnen weitere SQL-Statements zur Verfügung:

- `ADD COLUMN`: Über diese Anweisung fügen Sie einer Tabelle ein neues Feld hinzu.
- `ADD CONSTRAINT`: Mithilfe dieser Anweisung fügen Sie Ihrer Tabelle einen Mehrfachindex hinzu.
- `DROP COLUMN`: Durch den Einsatz dieser SQL-Anweisung entfernen Sie ein Feld aus Ihrer Tabelle.
- `DROP CONSTRAINT`: Löscht einen Mehrfachindex aus Ihrer Tabelle.

Die Syntax der Anweisung `Alter` lautet:

```
Alter Table Tabellennamen SQLAnw Name Datentyp
```

Im Argument `Tabellennamen` geben Sie den Namen der Tabelle an, in der Sie eine der vier Operatoren durchführen möchten. Über die SQL-Anweisung `SQLAnw` legen Sie fest, welche Aktion Sie ausführen möchten. Im Argument `Name` geben Sie den Namen der Spalte/des Indexes an. Im letzten Argument `Datentyp` geben Sie einen Typ aus Tabelle 104 an. Bei Löschaktionen sowie bei der Anlage eines Indexes muss das letzte Argument nicht angegeben werden.

Datentyp	Felddatentyp
`BIT`	Ja/Nein
`DATETIME`	Datum/Uhrzeit
`DECIMAL`	Numerischer Datentyp
`IMAGE`	OLE-Objekt
`MONEY`	Währung
`CHARACTER`	Text
`TEXT`	Memo
`SMALLINT`	Integerzahl (–32.768 bis 32.767)
`TINYINT`	Integerzahl zwischen 0 und 255
`INTEGER`	Integerzahl (–2.147.483.648 bis 2.147.483.647)

Tabelle 104: Die zur Verfügung stehenden Datentypen

Im folgenden Beispiel aus Listing 325 wird der Tabelle `Artikel` ein zusätzliches Feld mit dem Namen `AlternativArtikel` hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub SpalteEinfügen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "ALTER TABLE Artikel ADD COLUMN AlternativArtikel Text"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 325: Eine Spalte hinzufügen über ADD COLUMN

Im nächsten Beispiel aus Listing 326 wird die Spalte `AlternativArtikel` aus der Tabelle `Artikel` wieder gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub SpalteEntfernen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "ALTER TABLE Artikel DROP COLUMN AlternativArtikel"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 326: Eine Spalte löschen über DROP COLUMN

268 CREATE TABLE (Tabelle erstellen)

Verwenden Sie die SQL-Anweisung `CREATE TABLE`, um eine neue Tabelle in Ihrer Datenbank anzulegen. Dabei bestimmen Sie den Namen, Feldtyp und die Größe des Felds. Die so erstellte Tabelle ist vorerst aber noch leer. Die Syntax dieser Anweisung lautet:

```
CREATE TABLE Tabellennamen (XY Text(30), YY Text(10))
```

Im Argument `Tabellennamen` geben Sie den Namen der Tabelle an, die Sie erstellen möchten. Danach folgen die einzelnen Felder, getrennt durch ein Komma, mit den gewünschten Felddefinitionen.

Im nachfolgenden Beispiel aus Listing 327 wird eine neue Tabelle mit dem Namen `KundenAkt` angelegt, die folgende Felder enthalten soll:

- `Firma` (Textfeld mit maximal 30 Zeichen)
- `Kontaktperson` (Textfeld mit maximal 30 Zeichen)
- `Ort` (Textfeld mit maximal 20 Zeichen)
- `Land` (Textfeld mit maximal 20 Zeichen)

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=================================================

Sub TabelleErstellenAbfrage()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "CREATE TABLE KundenAKT " & _
      "(Firma Text(30), Kontaktperson Text(30), " & _
      "Ort Text(20), Land Text (20))"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Application.RefreshDatabaseWindow
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 327: Eine Tabelle erstellen über CREATE TABLE

Denken Sie daran, am Ende des Makros mithilfe der Methode `RefreshDataBaseWindow` die Ansicht zu aktualisieren, da sonst die neue Tabelle nicht angezeigt wird. Diese Methode simuliert das Drücken der Taste F5.

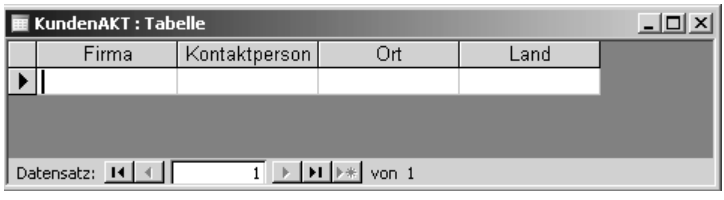


Abbildung 183: Die neue Tabelle wurde angelegt.

Im folgenden Beispiel aus Listing 328 wird eine neue Tabelle mit dem Namen `AlternativArtikel` angelegt und das Datenfeld `Lagerbestand` so definiert, dass darin keine Null eingegeben werden darf.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleErstellenAbfrage2()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "CREATE TABLE AlternativArtikel " & _
      "(Artikelname Text(30), " & _
      "Lagerbestand INTEGER NOT NULL, " & _
      "Mindestbestand INTEGER, Einzelpreis MONEY, " & _
      "Verfallsdatum DATETIME)"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Application.RefreshDatabaseWindow
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 328: Neue Tabelle mit Muss-Feldern anlegen

269 CREATE INDEX (Index definieren)

Möchten Sie einer Tabelle einen Index bzw. Mehrfachindex zuweisen, setzen Sie die SQL-Anweisung `CREATE INDEX` ein. Ein Index beschleunigt das Suchen und Sortieren von Datenfeldern.

```
CREATE UNIQUE INDEX NameIndex ON Tabellennamen (Feld)
```

Im Argument `NameIndex` geben Sie den Namen des Indexes an, den Sie einfügen möchten. Wenn Sie das Schlüsselwort `UNIQUE` weglassen, dann können auch doppelte Sätze angelegt werden, ansonsten ist der Index eindeutig. Nach dem Argument `ON` geben Sie den Namen der Tabelle sowie das Datenfeld an, das Sie mit dem Index ausstatten möchten.

Im folgenden Beispiel aus Listing 329 wird zunächst eine neue Tabelle mit dem Namen `ArtikelNeu2` erstellt und danach ein Datenfeld mit einem eindeutigen Index ausgestattet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub IndexDefinieren()
   Dim strSQL As String

   On Error GoTo fehler
   'Erster Schritt - Tabelle anlegen
   strSQL = "CREATE TABLE ArtikelNeu2 " & _
      "(ArtNr INTEGER, Artikelname Text(30), " & _
      "Lagerbestand INTEGER NOT NULL, " & _
      "Mindestbestand INTEGER, Einzelpreis MONEY, " & _
      "Verfallsdatum DATETIME)"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True

   'Zweiter Schritt - Index einbauen
   strSQL = "CREATE UNIQUE INDEX IndexDoppelt ON ArtikelNeu2 (ArtNr)"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Application.RefreshDatabaseWindow
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 329: Neue Tabelle mit Index erstellen über CREATE UNIQUE INDEX

270 DROP INDEX (Index entfernen)

Möchten Sie einen bereits gesetzten Index aus einer Tabelle entfernen, setzen Sie die SQL-Anweisung `DROP INDEX` ein. Die Syntax dieser Anweisung lautet:

```
DROP INDEX NameIndex ON TabellenNamen
```

Im Argument `NameIndex` geben Sie den Namen des Indexes an, den Sie entfernen möchten. Im Argument `TabellenNamen` geben Sie den Namen der Tabelle an, die den Index enthält.

Im folgenden Beispiel aus Listing 330 wird der eindeutige Index des Datenfelds `Art-Nr` entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub IndexEntfernen()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "DROP INDEX IndexDoppelt ON ArtikelNeu2"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Application.RefreshDatabaseWindow
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 330: Einen Index entfernen über DROP INDEX

271 DROP TABLE (Tabelle entfernen)

Verwenden Sie die SQL-Anweisung `DROP TABLE`, um eine Tabelle aus Ihrer Datenbank zu entfernen. Die Syntax dieser Anweisung lautet:

```
DROP TABLE TabellenName
```

Geben Sie im Argument `TabellenName` den Namen der Tabelle an, die Sie löschen möchten.

Im folgenden Beispiel aus Listing 331 wird die Tabelle `ArtikelNeu2` aus der Datenbank entfernt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'====================================================

Sub TabelleEntfernenAbfrage()
   Dim strSQL As String

   On Error GoTo fehler
   strSQL = "DROP TABLE ArtikelNeu2"

   DoCmd.SetWarnings False
   DoCmd.RunSQL strSQL
   DoCmd.SetWarnings True
   Application.RefreshDatabaseWindow
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 331: Eine Tabelle entfernen über DROP TABLE

272 Abfrage erstellen und speichern (ADO)

Im folgenden Beispiel aus Listing 332 wird die Tabelle `Artikel` ausgewertet, indem alle Artikel mit einem Einzelpreis größer 20 ermittelt und mit der Abfrage gespeichert werden.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADO
'====================================================

Sub AbfrageErstellen()
   Dim DBS As ADODB.Recordset
   On Error GoTo fehler
   Set DBS = _
      CurrentProject.Connection.Execute _
      ("CREATE VIEW ErsteView (Artikelname, Einzelpreis, Lagerbestand) " & _
      " AS SELECT Artikelname, Einzelpreis, Lagerbestand FROM Artikel " & _
      " Where Einzelpreis > 20", , adCmdText)
   Set DBS = Nothing
   Exit Sub
```

Listing 332: Eine Abfrage erstellen (ADO)

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 332: Eine Abfrage erstellen (ADO) (Forts.)

Über die SQL-Anweisung `CREATE VIEW` können Sie eine Abfrage anlegen und speichern. Geben Sie hinter dieser Anweisung den Namen der Abfrage an. Danach nennen Sie – getrennt durch Kommas – die Datenfelder, die in der Abfrage angezeigt werden sollen. Nach der Anweisung `AS SELECT` wiederholen Sie die Felder und geben bei `FROM` an, aus welcher Tabelle Sie die Felder nehmen möchten. In der abschließenden `WHERE`-Klausel können Sie das Ergebnis noch weiter einschränken.

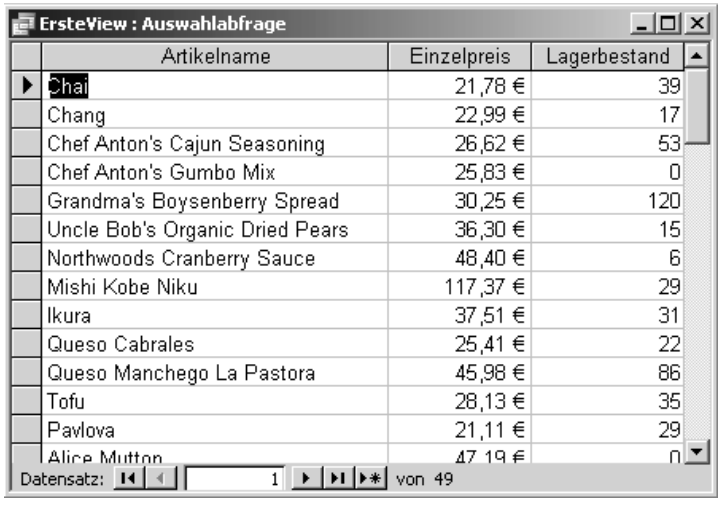


Abbildung 184: Die erste Abfrage wurde erstellt.

273 Abfrage erstellen und speichern (DAO)

Soll eine Abfrage über DAO erstellt und gespeichert werden, dann brauchen Sie hierfür ein `QueryDef`-Objekt.

Im folgenden Beispiel aus Listing 333 werden alle Daten aus der Tabelle `Kunden` extrahiert, bei denen die Kunden aus Deutschland kommen.

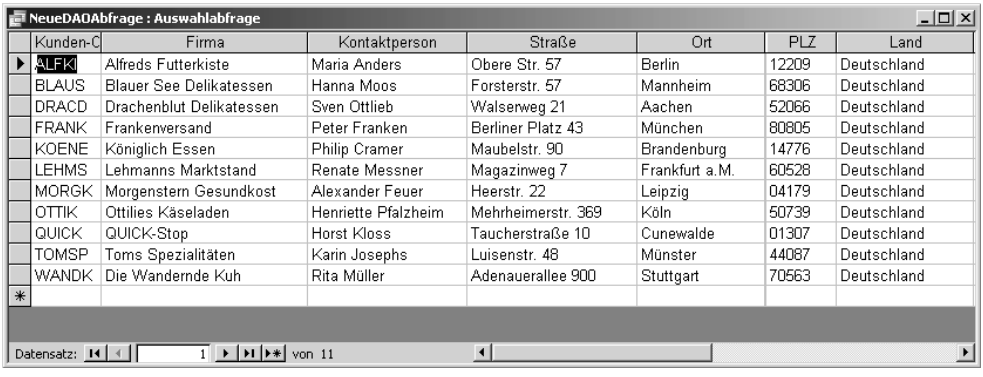
```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDAO
'=====================================================

Sub AbfrageErstellenDAO()
   Dim DBS As DAO.Database
   Dim qdf As DAO.QueryDef

   On Error GoTo fehler
   Set DBS = Application.CurrentDb

   With DBS
      Set qdf = .CreateQueryDef("NeueDAOAbfrage", _
         "SELECT * FROM KUNDEN WHERE Land ='Deutschland'")
      .Close
   End With

   Set qdf = Nothing
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 333: Eine Abfrage erstellen (DAO)

Deklarieren Sie zu Beginn des Makros aus Listing 333 zwei Objektvariablen. In der Objektvariablen `DBS` geben Sie die Datenbank bekannt, in der Sie die Abfrage erstellen möchten. Da es sich bei diesem Beispiel um die aktuell geöffnete Datenbank handelt, wenden Sie die Methode `CurrentDb` an, die eine Objektvariable vom Typ `Database` zurückgibt und somit der Datenbank entspricht, die gerade geöffnet ist.

Der Objektvariablen `qdf` weisen Sie die neue Abfrage zu, die Sie über die Methode `CreateQueryDef` anlegen. Bei dieser Methode geben Sie im ersten Argument den Namen der neuen Abfrage bekannt. Im zweiten Argument haben Sie die Möglichkeit, über eine SQL-Anweisung genau zu bestimmen, was die Abfrage machen soll. Nach dem Ausführen dieser Methode wird die Abfrage automatisch angelegt und gespeichert.

Heben Sie die Objektverweise zu den Objekten am Ende des Makros auf, indem Sie das Schlüsselwort `Nothing` den Objektvariablen zuweisen.


NeueDAOAbfrage : Auswahlabfrage

Kunden-C	Firma	Kontaktperson	Straße	Ort	PLZ	Land
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Deutschland
BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Deutschland
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Walserweg 21	Aachen	52066	Deutschland
FRANK	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Deutschland
KOENE	Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg	14776	Deutschland
LEHMS	Lehmanns Marktstand	Renate Messner	Magazinweg 7	Frankfurt a.M.	60528	Deutschland
MORGK	Morgenstern Gesundkost	Alexander Feuer	Heerstr. 22	Leipzig	04179	Deutschland
OTTIK	Ottilies Käseladen	Henriette Pfalzheim	Mehrheimerstr. 369	Köln	50739	Deutschland
QUICK	QUICK-Stop	Horst Kloss	Taucherstraße 10	Cunewalde	01307	Deutschland
TOMSP	Toms Spezialitäten	Karin Josephs	Luisenstr. 48	Münster	44087	Deutschland
WANDK	Die Wandernde Kuh	Rita Müller	Adenauerallee 900	Stuttgart	70563	Deutschland

Datensatz: 1 von 11

Abbildung 185: Alle Kunden aus Deutschland sind über eine Abfrage schnell verfügbar.

274 Abfragen über das Katalog-Objekt realisieren

Möchten Sie eine Abfrage erstellen und diese dann auch als Abfrage in Access speichern, dann müssen Sie unter dem Menü EXTRAS/VERWEISE die Bibliothek MICROSOFT ADO EXT. 2.5 FOR DDL AND SECURITY (Access 2002) bzw. die Bibliothek MICROSOFT ADO EXT. 2.7 FOR DDL AND SECURITY (Access 2003) einbinden.

Über die Eigenschaft `CommandText` können Sie die SQL-Anweisung formulieren. Die entsprechende Syntax gibt es in zwei Varianten:

```
SQLCmd.CommandText =" SQL-Anweisung"
```

Im Argument `SQL-Anweisung` geben Sie die SQL-Anweisung an, die Sie ausführen möchten.

```
SQLCmd.CommandText ="PARAMETERS [Datenfeld] text"
```

Über das Schlüsselwort `PARAMETERS` geben Sie an, dass es sich um eine Abfrage handeln soll, bei welcher der Anwender noch einen Wert eingeben muss. Setzen Sie einen Begriff in eckige Klammern. Dieser Begriff wird dann später in der Meldung angezeigt, sobald Sie die Abfrage durch einen Doppelklick starten möchten. Gleich danach legen Sie fest, um welche Art Dateneingabe es sich dabei handeln soll.

275 Normale Abfragen erstellen

Im Makro aus Listing 334 wird zunächst eine recht einfache Abfrage erstellt, die alle Artikel aus der Tabelle `Artikel` 1:1 übernimmt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================
```

Listing 334: Abfrage erstellen und speichern

```
Sub AbfrageAnlegen1()
   Dim Katalog As New ADOX.Catalog
   Dim SQLcmd As ADODB.Command

   On Error GoTo fehler
   Katalog.ActiveConnection = CurrentProject.Connection
   Set SQLcmd = New ADODB.Command

   With SQLcmd
      .CommandText = "SELECT * FROM Artikel"
   End With

   Katalog.Procedures.Append "ArtikelAbfr1", SQLcmd
   Set SQLcmd = Nothing
   Set Katalog = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 334: Abfrage erstellen und speichern (Forts.)

Definieren Sie im ersten Schritt eine Objektvariable vom Typ `ADOX`. Zusätzlich benötigen Sie noch ein ADODB-Objekt vom Typ `Command`. Darunter speichern Sie später mithilfe der Eigenschaft `CommandText` Ihre SQL-Anweisung. In der Eigenschaft `ActiveConnection` geben Sie auch hier wieder Ihre aktuelle Datenbank als Quelle an. Über das Objekt `Procedures` legen Sie eine gespeicherte Prozedur an. Setzen Sie die Methode `Append` ein, um diesem Objekt die Abfrage hinzuzufügen. Als weiteres Argument benötigt die Methode noch die Information, was konkret die Abfrage machen soll. Diese Information haben Sie bereits als SQL-Anweisung in der Variablen `SQLcmd` bekannt gegeben. Vergessen Sie nicht, die Objekte nach ihrem Gebrauch wieder freizugeben.

Kundenabfrage durchführen

Beim folgenden Makro aus Listing 335 wird die Tabelle `Kunden` ausgewertet. Dabei sollen alle Kunden aus Berlin und München ermittelt werden.

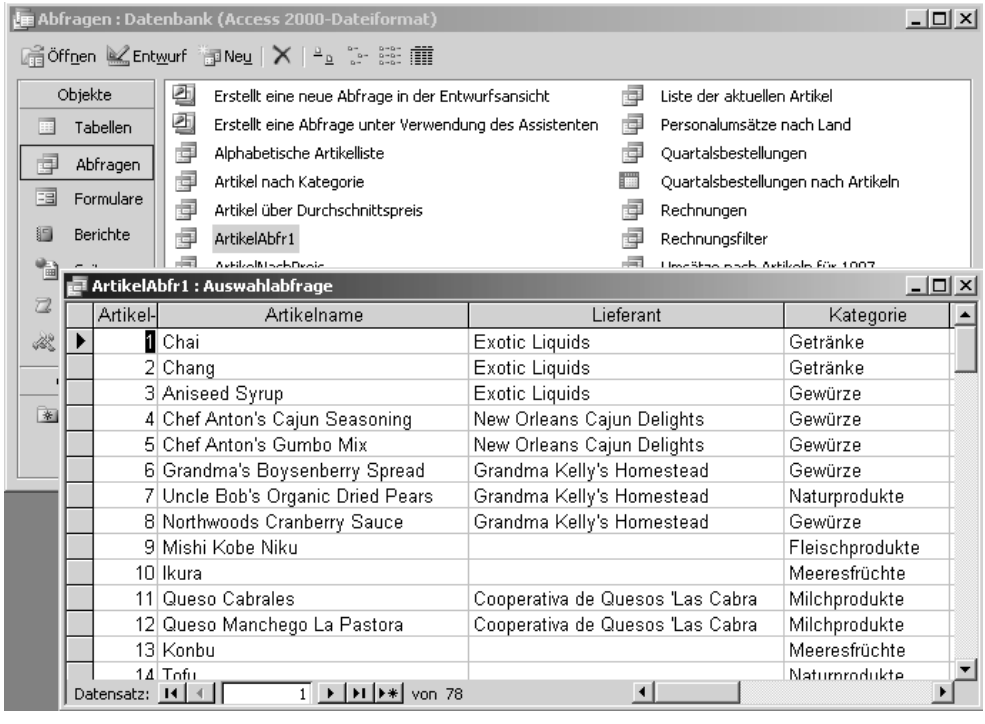


Abbildung 186: Die Abfrage ArtikelAbfr1 wurde angelegt und kann per Klick gestartet werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub AbfrageAnlegen2()
   Dim Katalog As New ADOX.Catalog
   Dim SQLcmd As ADODB.Command

   On Error GoTo fehler
   Katalog.ActiveConnection = CurrentProject.Connection
   Set SQLcmd = New ADODB.Command

   With SQLcmd
      .CommandText = "SELECT Firma, Kontaktperson, " & _
      "Straße,  Ort FROM Kunden " & _
      "WHERE (Ort = 'Berlin' OR Ort = 'München')"
   End With

   Katalog.Procedures.Append "KundenAbfr1", SQLcmd
   Application.RefreshDatabaseWindow
```

Listing 335: Alle Kunden aus bestimmten Orten auflisten

```
    Set SQLcmd = Nothing
    Set Katalog = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 335: Alle Kunden aus bestimmten Orten auflisten (Forts.)

Abbildung 187: Die Ortsabfrage wurde aufgesetzt.

Lieferanten und Kundenabfrage ausführen

Im folgenden Beispiel aus Listing 336 werden alle Kunden und Lieferanten, die aus Deutschland, Spanien oder Frankreich kommen, in einer Abfrage verarbeitet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub AbfrageAnlegen3()
   Dim Katalog As New ADOX.Catalog
   Dim SQLcmd As ADODB.Command

   On Error GoTo fehler
   Katalog.ActiveConnection = CurrentProject.Connection
   Set SQLcmd = New ADODB.Command

   With SQLcmd
      .CommandText = "SELECT Land, Ort, Firma, " & _
      "Kontaktperson AS Ansprechpartner " & _
      "FROM Kunden WHERE LAND = 'Deutschland' " & _
      "UNION ALL SELECT Land, Ort, Firma, " & _
```

Listing 336: Mehrere Tabellen auswerten

```
        "Kontaktperson FROM Lieferanten " & _
        "WHERE LAND = 'Deutschland' OR " & _
        "LAND = 'Spanien' OR Land = 'Frankreich' " & _
        "ORDER BY Ansprechpartner"
    End With

    Katalog.Procedures.Append "KundenUndLieferantenAbfr1", SQLcmd
    Application.RefreshDatabaseWindow
    Set SQLcmd = Nothing
    Set Katalog = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 336: Mehrere Tabellen auswerten (Forts.)

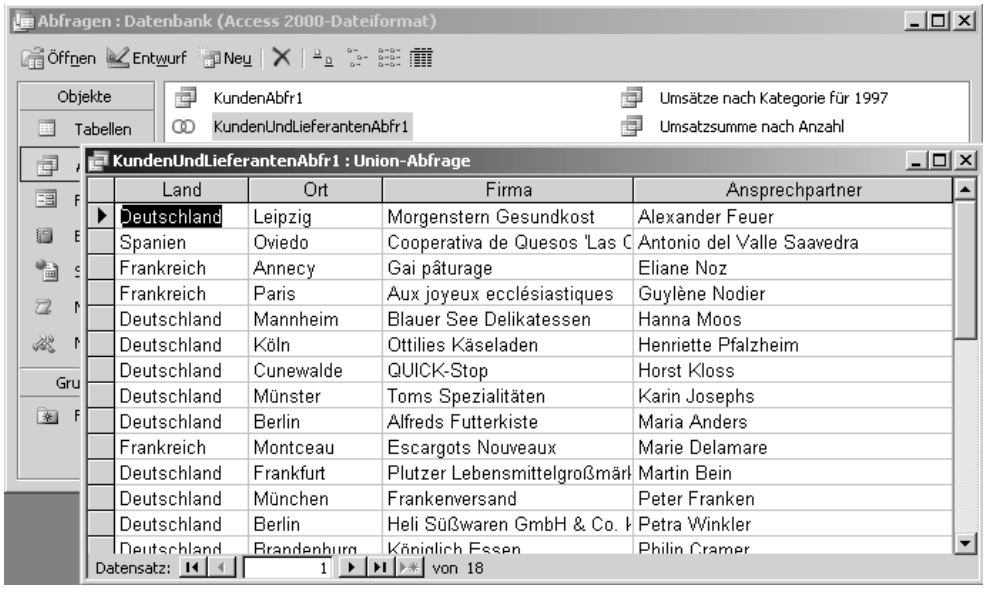


Abbildung 188: Eine UNION-Abfrage wurde erstellt (siehe Ring-Symbole).

276 Parameterabfragen erstellen (ADOX)

Bei Parameterabfragen müssen Sie über ein Eingabefeld das Kriterium eingeben, nach dem die Abfrage arbeiten soll.

Artikelstamm nach Preisgrenze abfragen

Im Beispiel aus Listing 337 wird die Tabelle `Artikel` ausgewertet. Dabei werden alle Artikel, die über einer bestimmten Preisgrenze liegen, ermittelt und angezeigt. Im Ergebnis sollen alle Felder ausgegeben werden.

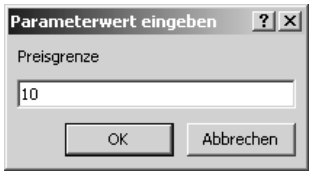


Abbildung 189: Alle Artikel über 10 € sind gefordert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=================================================

Sub ParameterAbfrageAnlegen()
   Dim Katalog As New ADOX.Catalog
   Dim SQLcmd As ADODB.Command

   Katalog.ActiveConnection = CurrentProject.Connection
   Set SQLcmd = New ADODB.Command

   With SQLcmd
      .CommandText = "PARAMETERS [Preisgrenze] MONEY; " & _
      "SELECT * FROM Artikel WHERE Einzelpreis > [Preisgrenze]"
   End With

   Katalog.Procedures.Append "ParameterAbfr1", SQLcmd
   Application.RefreshDatabaseWindow

   Set SQLcmd = Nothing
   Set Katalog = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 337: Eine Parameterabfrage erstellen – Datentyp Money (ADOX)

Über das Schlüsselwort `PARAMETERS` geben Sie an, dass es sich um eine Abfrage handeln soll, bei welcher der Anwender noch einen Wert eingeben muss. Setzen Sie einen Begriff in eckige Klammern. Dieser Begriff wird dann später in der Meldung angezeigt, sobald Sie die Abfrage durch einen Doppelklick starten möchten. Gleich danach legen Sie fest, um welche Art Dateneingabe es sich dabei handeln soll. Da es um einen Preis geht, können Sie hier den Datentyp `Money` angeben.

Lieferantenabfrage nach Land

Im nächsten Beispiel aus Listing 338 wird die Tabelle `Lieferanten` ausgewertet. Dabei sollen alle Lieferanten aus einem bestimmten Land ermittelt und angezeigt werden. Die Eingabe

ParameterAbfr1 : Auswahlabfrage

Artikel-	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €
2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	22,99 €
3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	12,10 €
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	26,62 €
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	30,25 €
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	36,30 €
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	48,40 €
9	Mishi Kobe Niku		Fleischprodukte	18 x 500-g-Packungen	117,37 €
10	Ikura		Meeresfrüchte	12 x 200-ml-Gläser	37,51 €
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabı	Milchprodukte	1-kg-Paket	25,41 €
12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabı	Milchprodukte	10 x 500-g-Packungen	45,98 €
14	Tofu		Naturprodukte	40 x 100-g-Packungen	28,13 €
15	Genen Shouyu		Gewürze	24 x 250-ml-Flaschen	18,76 €
16	Pavlova	Pavlova, Ltd.	Süßwaren	32 x 500-g-Kartons	21,11 €
17	Alice Mutton	Pavlova, Ltd.	Fleischprodukte	20 x 1-kg-Dosen	47,19 €
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte	16-kg-Paket	75,63 €
19	Teatime Chocolate Biscuits	Specialty Biscuits, Ltd.	Süßwaren	10 Kartons x 12 Stück	11,13 €

Datensatz: 1 von 72

Abbildung 190: Alle Artikel mit einem Einzelpreis von mehr als 10 € werden ausgegeben.

des gewünschten Lands erfolgt über eine Inputbox. Als Ergebnis werden alle Felder der Tabelle `Lieferanten` angefordert.

Abbildung 191: Die Eingabe des gewünschten Lands

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub ParameterAbfrageAnlegen2()
   Dim Katalog As New ADOX.Catalog
   Dim SQLcmd As ADODB.Command

   On Error GoTo fehler
   Katalog.ActiveConnection = CurrentProject.Connection
   Set SQLcmd = New ADODB.Command

   With SQLcmd
      .CommandText = "PARAMETERS [Land Eingabe] Text; " & _
      "SELECT * FROM Lieferanten WHERE Land = [Land Eingabe]"
   End With
```

Listing 338: Eine Parameterabfrage erstellen – Datentyp Text (ADOX)

```
    Katalog.Procedures.Append "ParameterAbfr2", SQLcmd
    Application.RefreshDatabaseWindow

    Set SQLcmd = Nothing
    Set Katalog = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 338: Eine Parameterabfrage erstellen – Datentyp Text (ADOX) (Forts.)

Geben Sie bei der Angabe `PARAMETERS` den Datentyp `Text` an.

Abbildung 192: Alle Lieferanten aus Deutschland stehen zur Verfügung.

277 Parameterabfragen erstellen (DAO)

Für die Erstellung von Parameterabfragen mit DAO greifen Sie auf die Methode `CreateQueryDef` zurück und übergeben dieser Methode die Eingabedaten über `PARAMETERS`.

Kundenabfrage erstellen

Im folgenden Makro aus Listing 339 wird eine Parameterabfrage auf Basis der Tabelle `Kunden` erstellt. Der Anwender soll dabei nach dem Start der Parameterabfrage das Land eingeben, anhand dessen die Kundenliste erstellt werden soll. Im Ergebnis sollen die Felder `Land`, `Ort` und `Kontaktperson` aufgeführt werden.

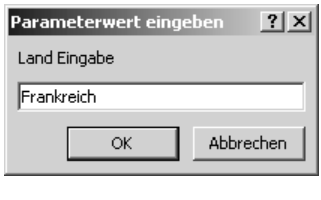


Abbildung 193: Als Parameter wird Frankreich erfasst.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDAO
'=================================================

Sub ParameterAbfrageDAO()
   Dim DBS As DAO.Database
   Dim qdf As DAO.QueryDef

   On Error GoTo fehler
   Set DBS = Application.CurrentDb

   Set qdf = DBS.CreateQueryDef("ParamterAbfrageDAO", _
      "PARAMETERS [Land Eingabe] Text; " & _
      "SELECT Land, Ort, Kontaktperson " & _
      "FROM Kunden WHERE Land = [Land Eingabe]")

   DBS.Close
   Set DBS = Nothing
   Set qdf = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 339: Eine Parameterabfrage anlegen (DAO)

Über das Schlüsselwort `PARAMETERS` geben Sie an, dass es sich um eine Abfrage handeln soll, bei welcher der Anwender noch einen Wert eingeben muss. Setzen Sie einen Begriff in eckige Klammern. Dieser Begriff wird dann später in der Meldung angezeigt, sobald Sie die Abfrage durch einen Doppelklick starten möchten. Gleich danach legen Sie fest, um welche Art Dateneingabe es sich dabei handeln soll. Da es hier um einen Text geht, können Sie den Datentyp `Text` angeben.

Artikel in einem Preiskorsett ausgeben

Im nächsten Beispiel aus Listing 340 wird die Tabelle `Artikel` über eine Parameterabfrage ausgewertet. Dabei sollen alle Artikel ausgegeben werden, die in einem bestimmten Preisbereich liegen. Sie müssen daher zwei Parameter definieren. Im Ergebnis sollen die Felder `Artikelname`, `Einzelpreis` und `Lagerbestand` aufgeführt werden.

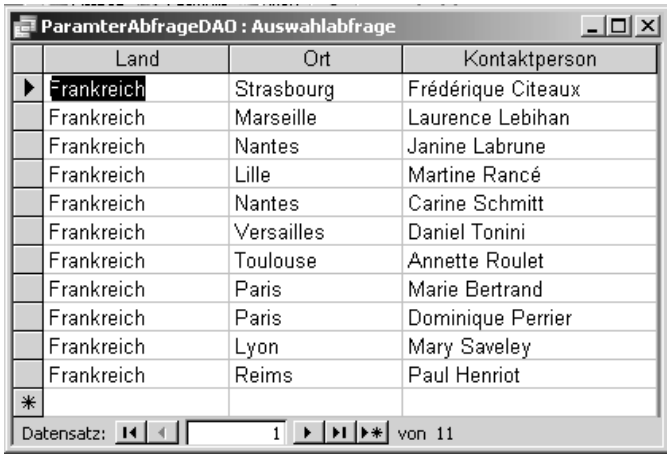


Abbildung 194: Nur Kunden aus Frankreich werden ausgegeben.

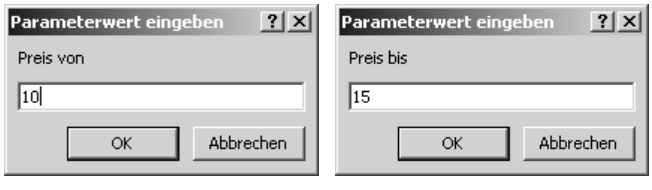


Abbildung 195: Die beiden Parameter werden nacheinander eingegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDAO
'=====================================================

Sub ParameterAbfrageDAO2()
   Dim DBS As DAO.Database
   Dim qdf As DAO.QueryDef

   On Error GoTo fehler
   Set DBS = Application.CurrentDb

   Set qdf = DBS.CreateQueryDef("ParamterAbfrageDAO2", _
      "PARAMETERS [Preis von] Money, [Preis bis] Money;  " & _
      "SELECT Artikelname, Einzelpreis, Lagerbestand " & _
      "FROM Artikel WHERE Einzelpreis BETWEEN [Preis von] AND [Preis bis]")

   DBS.Close
   Set DBS = Nothing
   Set qdf = Nothing
   Exit Sub
```

Listing 340: Eine Abfrage mit zwei Parametern anlegen (DAO)

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 340: Eine Abfrage mit zwei Parametern anlegen (DAO) (Forts.)

Wenn Sie mehrere Parameter definieren möchten, dann geben Sie diese jeweils durch ein Komma getrennt ein. Die Preisspanne können Sie über die SQL-Syntax `Einzelpreis BETWEEN [Preis von] AND [Preis bis]` bekannt geben. Da es sich hier um Preise handelt, geben Sie den Datentyp `Money` an.

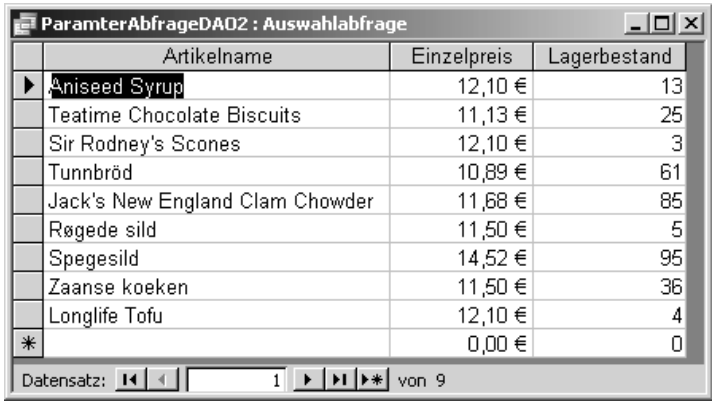


Abbildung 196: Alle Artikel, deren Preis zwischen 10 und 15 liegt, werden angezeigt.

Datumsabfrage von Bestellungen

Als letztes Beispiel für Parameterabfragen mit DAO wird im Makro aus Listing 341 die Tabelle `Bestellungen` ausgewertet. Dabei sollen alle Bestellungen in einem bestimmten Zeitfenster ausgegeben werden. Als Ergebnisfelder sind dabei die Felder `Bestelldatum`, `Empfänger`, `Straße` und `Ort` gefordert.

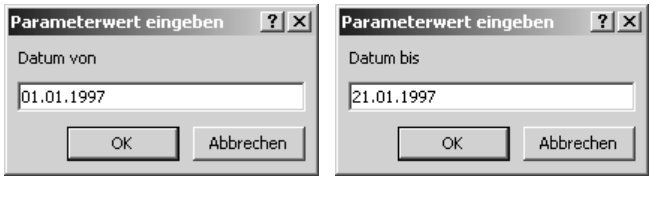


Abbildung 197: Eine Datumsspanne wird über zwei Parameter abgefragt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap05
' Dateiname  Abfragen.mdb
' Modul      mdlDAO
'=====================================================

Sub ParameterAbfrageDAO3()
   Dim DBS As DAO.Database
   Dim qdf As DAO.QueryDef

   On Error GoTo fehler
   Set DBS = Application.CurrentDb

   Set qdf = DBS.CreateQueryDef("ParamterAbfrageDAO3", _
      "PARAMETERS [Datum von] DateTime, [Datum bis] DateTime;  " & _
      "SELECT Bestelldatum, Empfänger, Straße, Ort " & _
      "FROM Bestellungen WHERE Bestelldatum BETWEEN [Datum von] AND [Datum bis]")

   DBS.Close
   Set DBS = Nothing
   Set qdf = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 341: Eine Zeitspanne wird ausgewertet (DAO)

Die Datumsspanne können Sie über die SQL-Syntax `Bestelldatum BETWEEN [Datum von] AND [Datum bis]`bekannt geben. Da es sich hier um Datumswerte handelt, geben Sie den Datentyp `DateTime` an.

Abbildung 198: Alle Bestellungen in einem bestimmten Zeitrahmen werden ausgegeben.

Steuerelemente, Dialoge und Formulare

In diesem Kapitel können Sie nachschlagen, wie Sie Formulare und Steuerelemente in Access erstellen und programmieren. Zu Beginn werden jedoch Dialoge und Meldungsfenster vorgestellt.

278 Das Meldungsfenster MsgBox

In den vorherigen Kapiteln wurde die Methode `MsgBox` bereits häufiger eingesetzt. Diese Methode wird z.B. verwendet, um den Anwender über das Ergebnis eines Makros zu informieren oder auch um eine Warnmeldung auf dem Bildschirm anzuzeigen. Dabei können Sie das Aussehen dieser Maske weitestgehend selbst bestimmen. Die Syntax dieser Methode lautet:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

Argument	Beschreibung
`Prompt`	Erforderlich. Ein Text, der als Meldung im Dialogfeld erscheinen soll.
`Buttons`	Optional. Bestimmt, welche Schaltflächen in der Meldung angezeigt werden. Diese Einstellung können Sie entweder durch eine Konstante oder einen eindeutigen Index vornehmen. Der folgenden Tabelle können Sie die möglichen Varianten entnehmen.
`Title`	Optional. Legt einen Text fest, der im Fenstertitel angezeigt wird.
`helpfile` und `context`	Optional. Wird eingesetzt, wenn Sie auf einen Hilfetext im Meldungsfenster verweisen möchten.

Tabelle 105: Die Argumente der Methode MsgBox

Konstante oder Wert	Beschreibung
`vbOKOnly` oder `0`	Zeigt nur die Schaltfläche OK an.
`vbOKCancel` oder `1`	Zeigt die Schaltflächen OK und ABBRECHEN an.
`vbAbortRetryIgnore` oder `2`	Zeigt die Schaltflächen ABBRUCH, WIEDERHOLEN und IGNORIEREN an.
`vbYesNoCancel` oder `3`	Zeigt die Schaltflächen JA, NEIN und ABBRECHEN an.
`vbYesNo` oder `4`	Zeigt die Schaltflächen JA und NEIN an.

Tabelle 106: Schaltflächen-Symbole für MsgBox

Konstante oder Wert	Beschreibung
vbRetryCancel oder 5	Zeigt die Schaltflächen WIEDERHOLEN und ABBRECHEN an.
vbCritical oder 16	Zeigt Meldungen mit STOPP-Symbol an.
vbQuestion oder 32	Zeigt Meldungen mit FRAGEZEICHEN-Symbol an.
vbExclamation oder 48	Zeigt Meldungen mit AUSRUFEZEICHEN-Symbol an.
vbInformation oder 64	Zeigt Meldungen mit INFO-Symbol an.
vbDefaultButton1 oder 0	Erste Schaltfläche ist Standardschaltfläche.
vbDefaultButton2 oder 256	Zweite Schaltfläche ist Standardschaltfläche.
vbDefaultButton3 oder 512	Dritte Schaltfläche ist Standardschaltfläche.
vbDefaultButton4 oder 768	Vierte Schaltfläche ist Standardschaltfläche.
vbApplicationModal oder 0	Der Anwender muss auf das Meldungsfeld zuerst reagieren, bevor er seine Arbeit mit der aktuellen Anwendung fortsetzen kann.
vbSystemModal oder 4096	Alle Anwendungen werden unterbrochen, bis der Benutzer auf das Meldungsfeld reagiert.
vbMsgBoxHelpButton oder 16384	Fügt dem Meldungsfenster eine Hilfeschaltfläche hinzu.

Tabelle 106: Schaltflächen-Symbole für MsgBox (Forts.)

Abhängig davon, welche Schaltfläche der Anwender im Meldungsfeld anklickt, sollen unterschiedliche Aktionen folgen. Wird z.B. die Schaltfläche ABBRECHEN angeklickt, muss das Makro sofort beendet werden. Der folgenden Tabelle können Sie die möglichen Rückgabewerte entnehmen.

Konstante oder Wert	Beschreibung
vbOK oder 1	Die Schaltfläche OK wurde angeklickt.
vbCancel oder 2	Die Schaltfläche ABBRECHEN wurde angeklickt.
vbAbort oder 3	Die Schaltfläche ABBRUCH wurde angeklickt.
vbRetry oder 4	Die Schaltfläche WIEDERHOLEN wurde angeklickt.
vbIgnore oder 5	Die Schaltfläche IGNORIEREN wurde angeklickt.
vbYes oder 6	Die Schaltfläche JA wurde angeklickt.
vbNo oder 7	Die Schaltfläche NEIN wurde angeklickt.

Tabelle 107: Rückgabewerte für Schaltflächen

Im folgenden Beispiel aus Listing 342 soll vor der Löschung eines Formulars nachgefragt werden, ob diese Aktion wirklich durchgeführt werden soll.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'====================================================

Sub RückfrageEinholen()
   Dim intS As Integer

   intS = MsgBox("Wollen Sie das Formular Artikel wirklich löschen?", _
            1 + vbQuestion, "Löschenrückfrage")

   If intS = 2 Then
      Exit Sub
   Else
      DoCmd.DeleteObject acForm, "Artikel"
   End If
End Sub
```

Listing 342: Die Rückfrage vor dem Löschen eines Formulars

Über die Variable `intS` wird festgehalten, welche Schaltfläche im Meldungsfenster angeklickt wurde. Für den Fall, dass dabei der Wert 2 zurückgegeben wird, wurde die Schaltfläche ABBRECHEN angeklickt. In diesem Fall wird das Makro über die Anweisung `Exit Sub` sofort beendet. Enthält der Inhalt der Variablen den Wert 1, dann wurde die Schaltfläche OK angeklickt. Wenn dem so ist, dann löschen Sie mithilfe der Methode `DeleteObject` das Formular `Artikel`.

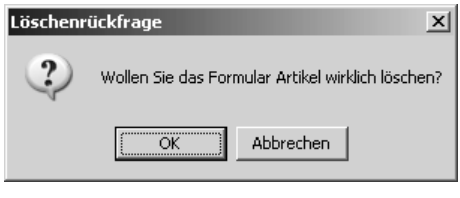


Abbildung 199: Das Meldungsfenster wird angezeigt.

Im nächsten Beispiel aus Listing 343 wird ein mehrzeiliger Meldungsdialog angezeigt. Dabei reicht es, wenn Sie eine Schaltfläche definieren.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'====================================================
```

Listing 343: Mehrere Zeilen im Meldungsfenster anzeigen

```
   Sub MehrzeiligeMsgBoxAnzeigen()
      MsgBox "Hallo Anwender " & Environ("username") _
         & vbLf & "Heute ist der " & Date & vbLf & "Genau " & Time & " Uhr!", _
         vbInformation, "Information"
End Sub
```

Listing 343: Mehrere Zeilen im Meldungsfenster anzeigen (Forts.)

Über die Funktion `Environ` können Sie auf die Umgebungsvariablen des Betriebssystems zugreifen, unter anderem auch auf den am System angemeldeten Anwender. Über die Standard-VBA-Funktionen `Date` und `Time` geben Sie das aktuelle Datum sowie die Uhrzeit aus.

Abbildung 200: Eine Bildschirmmeldung mit drei Zeilen

279 Der Eingabedialog Inputbox

Mithilfe der Methode `InputBox` versetzen Sie den Anwender in die Lage, einzelne Eingaben in einer Maske vorzunehmen. Diese Funktion eignet sich für kleinere Aufgaben hervorragend und auch hier können Sie Aussehen und Funktion des Dialogfelds selbst bestimmen. Diese Methode hat folgende Syntax:

```
InputBox(prompt, title, default, Left, Top, helpFile, helpContext)
```

Argument	Beschreibung
`Prompt`	Erforderlich. Ein Text, der als Meldung im Dialogfeld erscheinen soll.
`title..`	Optional. Legt einen Text fest, der im Fenstertitel angezeigt wird.
`default`	Optional. Hier kann eine Vorbelegung vorgenommen werden, die im Textfeld angezeigt wird, wenn der Benutzer keine Eingabe vorgenommen hat. Wenn Sie das Argument weglassen, wird ein leeres Textfeld angezeigt.

Tabelle 108: Die Argumente der Methode Inputbox

Argument	Beschreibung
`left` und `top`	Optional. Legt die Position auf dem Bildschirm fest, wo das entsprechende Dialogfeld angezeigt werden soll. So wird beim Argument `left` der horizontale Abstand des linken Rands des Dialogfelds vom linken Rand des Bildschirms festgelegt. Beim Argument `top` wird der vertikale Abstand des oberen Rands des Dialogfelds vom oberen Rand des Bildschirms festgelegt.
`helpfile` und `context`	Optional. Diese Argumente werden eingesetzt, wenn Sie auf einen Hilfetext im Eingabefenster verweisen möchten.

Tabelle 108: Die Argumente der Methode Inputbox (Forts.)

Im folgenden Beispiel aus Listing 344 wird eine Inputbox verwendet, um eine Eingabe zu erzwingen. Sollte der Anwender keine Eingabe vornehmen, dann wird das Makro sofort beendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=====================================================

Sub EingabeErzwingen()
   Dim strEingabe As String

   strEingabe = InputBox("Bitte geben Sie einen Namen ein!")
   If strEingabe = vbNullString Then
      Exit Sub
   Else
      MsgBox "Sie haben den Namen " & strEingabe & " eingegeben!", vbInformation
   End If
End Sub
```

Listing 344: Eine Eingabe über Inputbox auswerten

280 Die FileDialog-Standarddialoge

Das `FileDialog`-Objekt kann einsetzt werden, um Standarddialoge wie ÖFFNEN oder SPEICHERN aufzurufen. Als Dialoge stehen Ihnen die folgenden vier Standarddialoge zur Verfügung:

Dialogtyp	Beschreibung
`msoFileDialogFilePicker`	DATEIAUSWAHL
`msoFileDialogFolderPicker`	ORDNERAUSWAHL

Tabelle 109: Die vier Standarddialoge

Dialogtyp	Beschreibung
msoFileDialogOpen	ÖFFNEN
msoFileDialogSaveAs	SPEICHERN UNTER

Tabelle 109: Die vier Standarddialoge (Forts.)

Im folgenden Beispiel aus Listing 345 wird der Dialog DATEIAUSWAHL angezeigt und darin werden Bilder in der Miniaturansicht angeboten. Damit Sie später die ausgewählten Bilder im Internet Explorer anzeigen können, aktivieren Sie in der Entwicklungsumgebung unter EXTRAS/VERWEISE die Bibliothek MICROSOFT INTERNET CONTROLS.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=====================================================

Sub DateiAuswahlDialogAnzeigen()
   Dim dlg As FileDialog
   Dim ArrAuswahl As Variant
   Dim ie As InternetExplorer

   Set dlg = Application.FileDialog(msoFileDialogFilePicker)

   With dlg
      .AllowMultiSelect = True
      .ButtonName = "Zeigen"
      .InitialFileName = "*.jpg"
      .Filters.Clear
      .Filters.Add "Bilder", "*.jpg; *.gif"
      .InitialView = msoFileDialogViewThumbnail
      .Title = "Markieren Sie einige Bilder"

      If .Show Then
         For Each ArrAuswahl In .SelectedItems
            Set ie = New InternetExplorer
            ie.Visible = True
            ie.navigate ArrAuswahl
         Next
      Else
         MsgBox "Sie haben Abbrechen gedrückt!"
      End If
   End With
End Sub
```

Listing 345: Den Dateiauswahl-Dialog anzeigen

Setzen Sie die Eigenschaft `AllowMultiSelect` auf den Wert `True`, wenn der Benutzer die Möglichkeit hat, mehrere Dateien in einem Dateidialogfeld auszuwählen. Über die Eigenschaft `ButtonName` können Sie den angezeigten Text einer Aktionsschaltfläche des Dialogs festlegen.

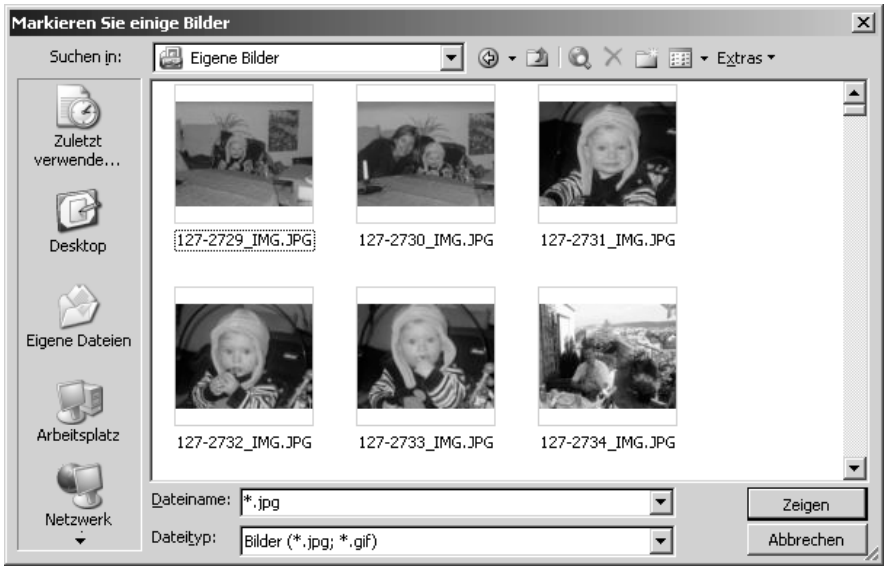


Abbildung 201: Der Standarddialog zur Dateiauswahl wird benutzerdefiniert angezeigt.

Standardmäßig wird diese Eigenschaft auf den Standardtext für den Typ des Dateidialogfelds gesetzt.

Die Eigenschaft `InitialFileName` legt den zuerst in einem Dateidialogfeld angezeigten Pfad und/oder Dateinamen fest.

Über die Eigenschaft `Filters` können Sie einen Dateifilter im Dialogfeld einstellen.

Über die Anweisung `Filters.Clear` löschen Sie den bisher eingestellten Filter. Über die Anweisung

```
Filters.Add "Bilder", "*.gif; *.jpg; *.jpeg", 1
```

fügen Sie beispielsweise einen Bildfilter ein und positionieren den ersten Eintrag im Dropdown-Menü DATEITYP auf den Eintrag *.GIF.

Über die Eigenschaft `InitialView` legen Sie fest, in welcher Ansicht die Dateien im Dialog angeboten werden sollen.

Dabei stehen folgende Ansichtstypen zur Verfügung.

Typ	Beschreibung
`msoFileDialogViewDetails`	Ansicht DETAILS
`msoFileDialogViewLargeIcons`	Ansicht GROSSE SYMBOLE
`msoFileDialogViewList`	Ansicht LISTE
`msoFileDialogViewPreview`	Ansicht VORSCHAU
`msoFileDialogViewProperties`	Ansicht EIGENSCHAFTEN

Tabelle 110: Die Ansichtstypen für Dateien

Typ	Beschreibung
msoFileDialogViewSmallIcons	Ansicht KLEINE SYMBOLE
msoFileDialogViewThumbnail	Miniaturansicht
msoFileDialogViewWebView	Webansicht

Tabelle 110: Die Ansichtstypen für Dateien (Forts.)

Über die Eigenschaft `Title` können Sie den Titel eines Dialogs festlegen. Die Methode `Show` zeigt ein Dateidialogfeld an und gibt dann einen Long-Wert zurück, der angibt, ob der Benutzer auf die Schaltfläche zum Ausführen der Aktion (–1) oder auf die Schaltfläche zum ABBRECHEN (0) geklickt hat.

Die Eigenschaft `SelectedItems` gibt eine `FileDialogSelectedItems`-Auflistung zurück. Diese Auflistung enthält eine Liste der Pfade von Dateien, die ein Benutzer in einem mithilfe der `Show`-Methode des `FileDialog`-Objekts angezeigten Dateidialogfeld ausgewählt hat.

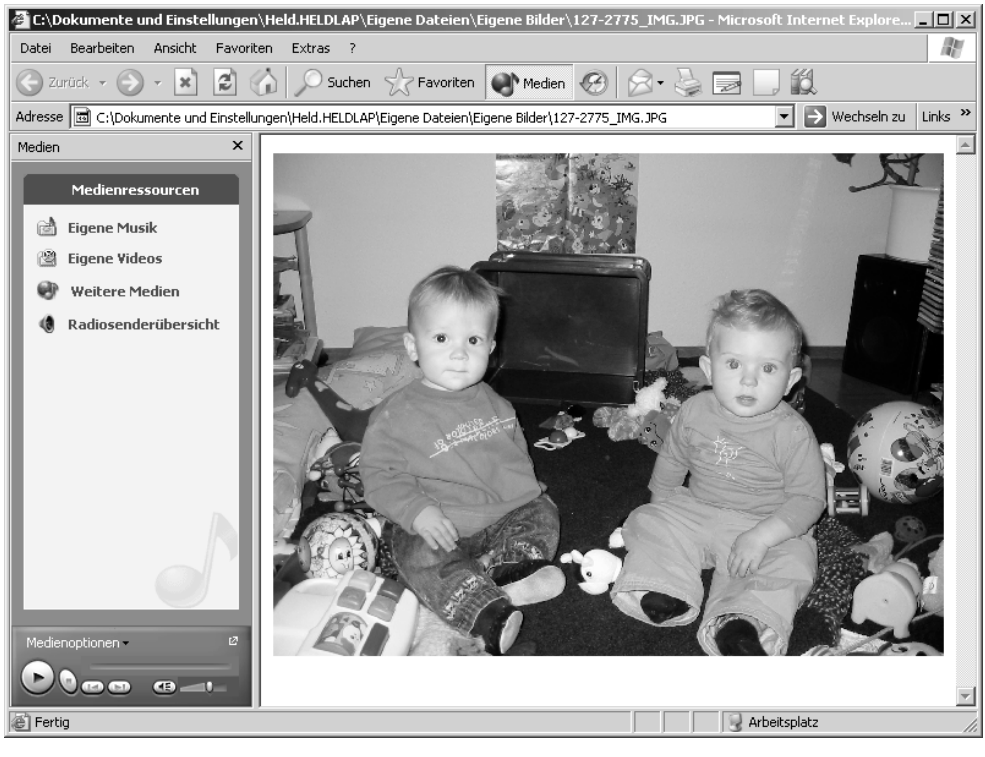


Abbildung 202: Die ausgewählten Bilder werden im Internet Explorer angezeigt.

281 Sonstige Dialoge

Eine weitere Möglichkeit, Dialoge einzusetzen, die in Access standardmäßig ja schon zur Verfügung stehen, ist der Aufruf von Dialogen über die Methode `RunCommand`.

Mithilfe der Methode RunCommand können Sie jeden Menübefehl automatisch per Code ausführen lassen. So können Sie beispielsweise auch den integrierten DRUCKEN-Dialog von Access aufrufen, indem Sie aus dem Menü DATEI den Befehl DRUCKEN über ein Makro auswählen. Fügen Sie dazu das Makro aus Listing 346 in der Entwicklungsumgebung von Access ein.

```
'================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'================================================

Sub DialogDrucken()
   DoCmd.OpenReport "Rechnung", acViewPreview
   On Error GoTo fehler
   DoCmd.RunCommand acCmdPrint
   Exit Sub


   fehler:
End Sub
```

Listing 346: Einen integrierten Dialog aufrufen

Wenden Sie im ersten Schritt das Objekt DoCmd an, um Zugriff auf weitere Methoden in Access zu erlangen. Eine dieser Methoden ist OpenReport, über die Sie einen Bericht aufrufen können. Über die Konstante acViewPreview bestimmen Sie, dass der Bericht in der Berichtsvorschau aufgerufen wird. Dieser Bericht soll nun gedruckt werden, jedoch möchten Sie vorab noch einige Einstellungen am Drucker über den Dialog DRUCKEN einstellen.

Um nun den Dialog DRUCKEN richtig abfangen zu können, müssen Sie dafür sorgen, dass es beim Abbruch dieses Dialogs über die Schaltfläche ABBRECHEN nicht zu einem Laufzeitfehler kommt. Dazu integrieren Sie eine On Error-Klausel vor dem Aufruf des Dialogs und verzweigen im Fehlerfall (wenn die Schaltfläche ABBRECHEN angeklickt bzw. die Taste ESC gedrückt wird) zum Paragraphen Fehler. Dort angekommen, springen Sie über die Anweisung Exit Sub ohne weitere Aktionen direkt aus dem Makro. Alle weiteren Schaltflächen im Dialog DRUCKEN können wie gewohnt bedient werden und müssen nicht weiter abgefangen werden.

282 Formulare

Möchten Sie mit Formularen in Access arbeiten, um sich beispielsweise die Eingabe von Daten zu erleichtern oder Suchfunktionen in Access einzusetzen, dann können Sie sich Ihre eigenen Formulare zusammenbasteln. Für diese Aufgabe können Sie den Formular-Assistenten von Access einsetzen und später das erzeugte Formular weiter verfeinern.

Um ein Formular über den Formular-Assistenten zu erzeugen, befolgen Sie die nächsten Arbeitsschritte:

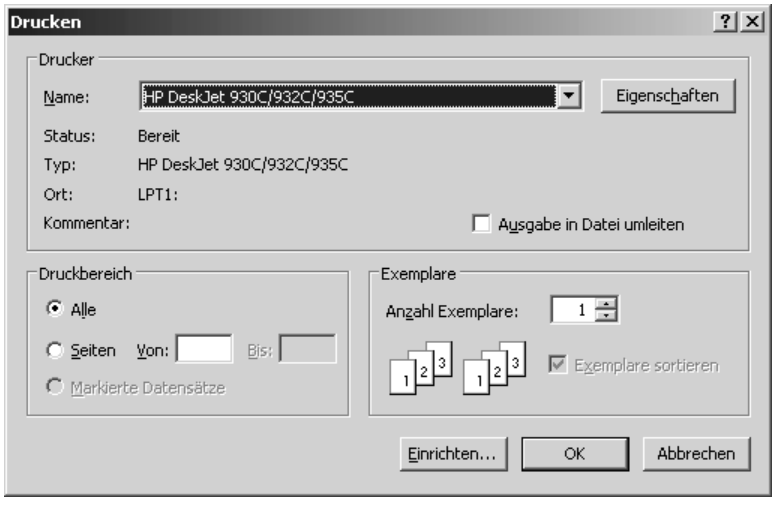


Abbildung 203: Der integrierte Dialog Drucken wurde aufgerufen.

1. Begeben Sie sich in das Datenbankfenster und wählen Sie aus dem Menü ANSICHT den Befehl DATENBANKOBJEKTE/FORMULARE aus.
2. Klicken Sie auf die Schaltfläche NEU.
3. Markieren Sie im Listenfeld den Eintrag FORMULAR-ASSISTENT und wählen Sie aus dem Dropdown-Menü die Tabelle oder die Abfrage aus, auf deren Grundlage Sie das Formular erstellen möchten. Stellen Sie dort jetzt beispielsweise einmal die Tabelle ARTIKEL ein.
4. Klicken Sie danach auf OK.
5. Markieren Sie die einzelnen Datenfelder, die Ihr Formular anzeigen soll, und klicken Sie auf die Pfeilsymbole, um die Felder einzufügen.
6. Klicken Sie danach auf WEITER.
7. Bestimmen Sie das Layout Ihres Formulars, indem Sie eine der angebotenen Optionen auswählen.
8. Klicken Sie danach auf WEITER, um zum nächsten Schritt des Assistenten zu gelangen.
9. Wählen Sie ein gewünschtes Format für das Formular aus und betrachten Sie das Resultat im linken Vorschaufenster.
10. Klicken Sie danach auf WEITER.
11. Geben Sie im letzten Schritt des Assistenten dem Formular einen Namen und lassen Sie die Einstellungen wirksam werden, indem Sie auf die Schaltfläche FERTIG STELLEN klicken. Das Formular wird gleich danach standardmäßig aufgerufen.

Abbildung 204: Den Formular-Assistenten einsetzen

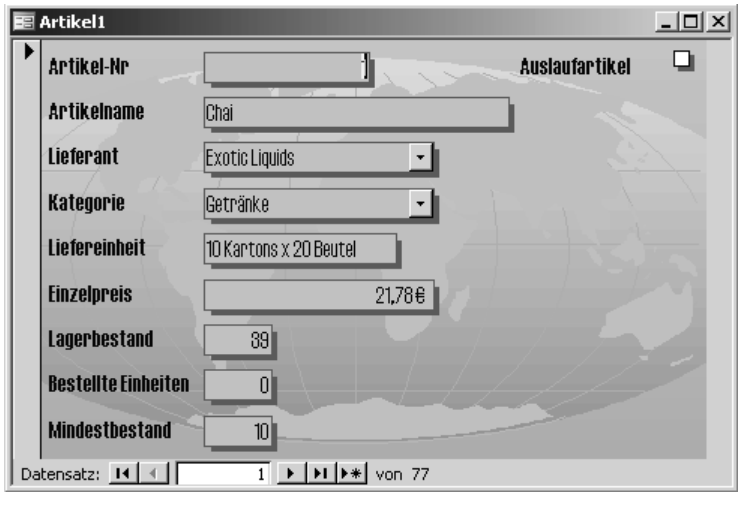


Abbildung 205: Das fertige Formular

Hinweis

Selbstverständlich können Sie ein Formular auch ohne Verwendung des Formular-Assistenten entwerfen und von Grund auf neu erstellen. Wechseln Sie dazu in das Datenbankfenster und klicken Sie in der Objektliste auf den Eintrag FORMULAR. Klicken Sie danach auf das Symbol NEU. Wählen Sie im Dialogfeld NEUES FORMULAR im Listenfeld den Eintrag ENTWURFSANSICHT, wählen Sie die Tabelle bzw. Abfrage aus, auf die sich das Formular beziehen soll, und bestätigen Sie über die Schaltfläche OK.

283 Die Steuerelemente für Formulare

Sobald Sie in die Entwurfsansicht eines Formulars wechseln, wird die Symbolleiste TOOLBOX eingeblendet. Diese Toolbox enthält die Steuerelemente für Ihr Formular.

Entnehmen Sie der folgenden Tabelle die zur Verfügung stehenden Steuerelemente.

Element	Beschreibung
	Über das Symbol OBJEKTE MARKIEREN können Sie Formularfelder markieren.
	Das Symbol BEZEICHNUNG setzen Sie ein, um beschreibenden Text, z.B. Titel, Beschriftungen oder kurze Anweisungen, für andere Formularfelder anzuzeigen. Bezeichnungsfelder zeigen keine Werte aus Feldern oder Ausdrücken an. Sie sind stets ungebunden. Es handelt sich dabei standardmäßig um konstante Texte.
	Das Symbol TEXTFELD verwenden Sie, um Daten aus einer Datenherkunft anzuzeigen. Dieser Textfeldtyp wird als gebundenes Textfeld bezeichnet, da es an Daten eines Felds (Daten aus einer Tabelle oder einer Abfrage) gebunden ist. Textfelder können jedoch auch ungebunden sein. Sie können beispielsweise ein ungebundenes Textfeld erstellen, um das Ergebnis einer Berechnung anzuzeigen oder Benutzereingaben aufzunehmen. Daten in einem ungebundenen Textfeld werden dann jedoch nicht in einer Tabelle gespeichert.
	Sie können eine OPTIONSGRUPPE in einem Formular dazu verwenden, eine begrenzte Anzahl an Auswahlmöglichkeiten anzuzeigen. Eine Optionsgruppe erleichtert das Auswählen eines Werts, da Sie nur auf den gewünschten Wert klicken müssen. In einer Optionsgruppe kann immer nur eine Option ausgewählt werden.
	Bei der UMSCHALTFLÄCHE handelt es sich um eine Schaltfläche, die einem Lichtschalter ähnelt. Sie kann immer nur einen Zustand annehmen. Wenn Sie diese Schaltfläche anklicken, wird sie gedrückt dargestellt. Ein nochmaliger Klick darauf entrastet diese Schaltfläche wieder.
	Das OPTIONSFELD, auch bekannt als RADIOBUTTON, kann aktiviert oder nicht aktiviert sein. Im aktivierten Zustand ist das Optionsfeld mit einem schwarzen Punkt ausgefüllt. Sind Optionsfelder gruppiert, kann immer nur ein Optionsfeld aktiviert sein.
	Das KONTROLLKÄSTCHEN kann entweder aktiviert oder nicht aktiviert sein. Im aktivierten Zustand erscheint im Kästchen ein Häkchen. Wenn Sie Kontrollkästchen in einer Gruppe verwenden, können eines oder auch mehrere Kontrollkästchen aktiviert sein.
	Ein Kombinationsfeld besteht streng genommen aus einem Eingabefeld, welches mit einem Listenfeld gekoppelt ist. Kombinationsfelder erkennen Sie daran, dass sich rechts neben dem Eingabefeld ein kleiner Pfeil nach unten befindet. Mit einem Klick darauf werden Ihnen weitere Auswahlmöglichkeiten angeboten. In einem Kombinationsfeld kann immer nur ein Eintrag gewählt werden.
	Verwandt mit dem Kombinationsfeld ist auch das Listenfeld. Das Listenfeld benötigt jedoch mehr Platz, weil mehrere Einträge gleichzeitig angezeigt werden. Ein Listenfeld kann so eingestellt werden, dass sich mehrere Einträge auswählen lassen.

Tabelle 111: Die Steuerelemente für Formulare

Element	Beschreibung
	Befehlsschaltflächen bieten Ihnen die Möglichkeit, Aktionen auszuführen, indem Sie einfach auf die Schaltflächen klicken.
	Das Symbol BILD setzen Sie ein, wenn Sie in Ihrem Formular ein Bild anzeigen möchten. Nach dem Einfügen dieses Symbols in Ihr Formular öffnet sich ein Dialogfeld, in dem Sie das gewünschte Bild von Ihrer Festplatte auswählen können. Dieses Bild wird dann unverknüpft eingefügt.
	Das Symbol OBJEKTFELD kommt dann zum Einsatz, wenn Sie Ihrem Formular ein ungebundenes Bild hinzufügen wollen und dabei die Verknüpfung zur Quelle erhalten möchten. Bei einem ungebundenen Objekt besteht keine Beziehung zu den Datenfeldern in der Tabelle bzw. der Abfrage.
	Das Symbol GEBUNDENES OBJEKTFELD setzen Sie ein, wenn sich das eingefügte Objekt direkt auf ein Datenfeld in Ihrer Tabelle oder Abfrage bezieht.
	Mithilfe des Symbols SEITENUMBRUCH legen Sie bei mehrseitigen Formularen den Seitenumbruch fest.
	Das Symbol REGISTERSTEUERELEMENT kann eingesetzt werden, um ein Formular mithilfe von mehreren Registerkarten übersichtlicher zu gestalten. Sie können somit Informationen sinngemäß trennen und in Gruppen aufsplitten.
	Über das Symbol UNTERFORMULAR/-BERICHT können Sie ausgehend von Ihrem Formular ein weiteres Unterformular anzeigen.
	Mithilfe des Symbols LINIE können Sie Ihr Formular optisch strukturieren. Diese Linie können Sie beispielsweise mit einem Spezialeffekt versehen, um einen räumlichen Trenneffekt zu erzeugen.
	Auch über das Symbol RECHTECK können Sie schöne gestalterische Effekte in Ihrem Formular erzeugen. Fassen Sie beispielsweise mehrere zusammenhängende Informationen in einem Rahmenrechteck zusammen, um diese Infos als Gruppe darzustellen.
	Über das Symbol WEITERE STEUERELEMENTE haben Sie Zugriff auf weitere mögliche Elemente, die Sie für Ihr Formular einsetzen können.

Tabelle 111: Die Steuerelemente für Formulare (Forts.)

284 Formulare auslesen

Um auf einzelne Steuerelemente in Formularen zuzugreifen, müssen Sie erst einmal wissen, wie Sie diese ansprechen können. Über die Auflistung `Controls` lassen sich alle Steuerelemente auf einem Formular auslesen.

Das folgende Beispiel aus Listing 347 ermittelt die Namen aller Steuerelemente, die sich im Formular `Artikel` befinden. Dazu durchlaufen Sie alle Steuerelemente, die in der Auflistung `Controls` verzeichnet sind, und setzen Sie die Eigenschaft `Name` ein, um die Namen der Steuerelemente zu ermitteln.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=====================================================

Sub SteuerelementeAuslesen()
   Dim frm As Form
   Dim ctl As Control

   Set frm = Form_Artikel
   For Each ctl In frm.Controls
      Debug.Print ctl.Name
   Next ctl
End Sub
```

Listing 347: Steuerelemente auslesen

Deklarieren Sie zunächst zwei Objektvariablen. In der Variablen `frm` geben Sie über die Anweisung `Set` den Namen des Formulars an, auf das Sie zugreifen möchten. Über eine `For each Next`-Schleife arbeiten Sie alle Steuerelemente, die über die Objektvariable `ctl` angesprochen werden können, im Formular ab und ermitteln den Namen des jeweiligen Steuerelements mithilfe der Eigenschaft `Name`. Über die Anweisung `Debug.Print` geben Sie die Steuerelemente im Direktfenster der Entwicklungsumgebung aus.

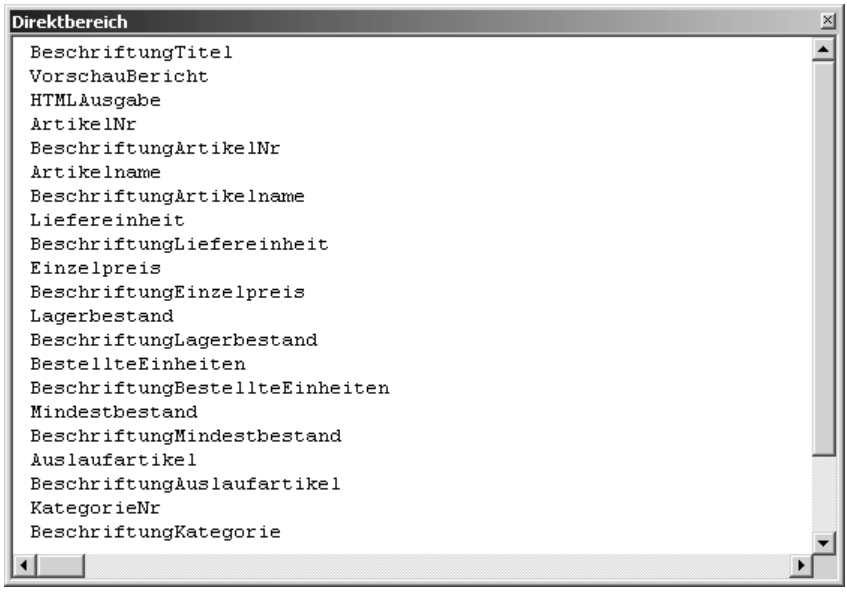


Abbildung 206: Alle Steuerelemente des Formulars Artikel

Wollen Sie ermitteln, um welches Steuerelement es sich handelt, setzen Sie die Eigenschaft `ControlType` ein. Diese Eigenschaft gibt einen Indexwert zurück. So haben Textfelder den

Index 109, Beschriftungsfelder den Wert 100, Schaltflächen den Wert 104 und Kontrollkästchen den Wert 106.

Im folgenden Beispiel aus Listing 348 werden die Steuerelemente des Formulars `Artikel` identifiziert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=====================================================

Sub SteuerelementeIDentifizieren()
   Dim frm As Form
   Dim ctl As Control

   Set frm = Form_Artikel
   For Each ctl In frm.Controls
      Debug.Print "Typ: " & ctl.ControlType & " Name: " & ctl.Name
   Next ctl
End Sub
```

Listing 348: Identifikation der Steuerelemente

Mithilfe der Eigenschaft `ControlType` können Sie die einzelnen Steuerelemente im Formular unterscheiden.

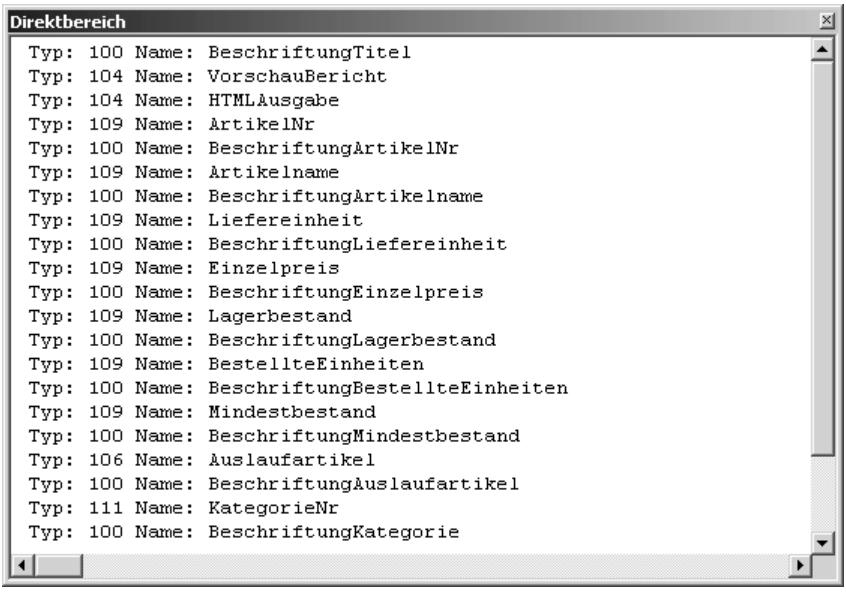


Abbildung 207: Der Typ der Steuerelemente wird ermittelt.

Die Typen der einzelnen Steuerelemente können Sie aber auch über Konstanten ermitteln. Sehen Sie sich dazu die folgende Tabelle an.

Konstante	Steuerelement
acBoundObjectFrame	Gebundenes Objektfeld
acCheckBox	Kontrollkästchen
acComboBox	Kombinationsfeld
acCommandButton	Befehlsschaltfläche
acCustomControl	ActiveX-Steuerelement
acImage	Bild
acLabel	Bezeichnungsfeld
acLine	Linie
acListBox	Listenfeld
acObjectFrame	Ungebundenes Objektfeld oder Diagramm
acOptionButton	Optionsschaltfläche
acOptionGroup	Optionsgruppe
acPage	Seite
acPageBreak	Seitenwechsel
acRectangle	Rechteck
acSubform	Unterformular/-bericht
acTabCtl	Registersteuerelement
acTextBox	Textfeld
acToggleButton	Umschaltfläche

Tabelle 112: Die Konstanten, um die Steuerelemente zu identifizieren

Im folgenden Beispiel aus Listing 349 sollen nur die Textfelder des Formulars `Artikel` ermittelt und im Direktfenster ausgegeben werden.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlForm
'======================================================

Sub NurTextfelderAusgeben()
   Dim frm As Form
   Dim ctl As Control

   Set frm = Form_Artikel
```

Listing 349: Nur Textfelder über die Konstante acTextBox ermitteln

```
    For Each ctl In frm.Controls
       If ctl.ControlType = acTextBox Then
          Debug.Print "Typ: " & ctl.ControlType & " Name: " & ctl.Name
       End If
    Next ctl
End Sub
```

Listing 349: Nur Textfelder über die Konstante acTextBox ermitteln (Forts.)

Die Eigenschaft `ControlType` kann auch direkt über Konstanten, die Sie in Tabelle 112 sehen können, abgefragt werden.

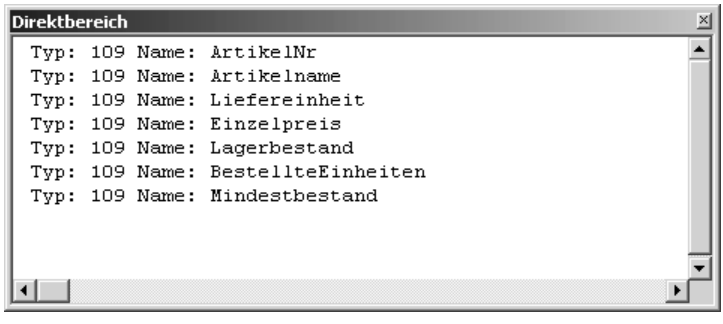


Abbildung 208: Es werden nur Textfelder ausgegeben.

285 Formulare öffnen und schließen

Um ein Formular mit VBA aufzurufen, setzen Sie die Methode `OpenForm` ein. Die Syntax dieser Methode lautet:

```
OpenForm(Formularname, Ansicht, Filtername, Bedingung, Datenmodus, Fenstermodus,
Öffnungsargumente)
```

Argument	Beschreibung
`Formularname`	Erforderlich. Gibt den Namen des Formulars an, das geöffnet werden soll.
`Ansicht`	Optional. Legt die Art der Ansicht des Formulars fest. `acDesign`: öffnet das Formular in der Entwurfsansicht. `acNormal` öffnet das Formular wie gewohnt. Sie können diese Konstante aber auch weglassen.
`Filtername` und `Bedingung`	Optional. Stellt einen Filter ein, um nur bestimmte Datensätze im Formular anzuzeigen.

Tabelle 113: Die Argumente der Methode OpenForm

Argument	Beschreibung
Datenmodus	Optional. Legt fest, welche Aufgaben der Anwender im Formular durchführen kann. Dabei stehen Ihnen folgende Konstanten zur Verfügung. acFormAdd: Der Anwender kann neue Datensätze im Formular hinzufügen, jedoch keine bestehenden Datensätze bearbeiten. acFormEdit: Der Anwender kann bestehende Datensätze im Formular bearbeiten und neue Datensätze hinzufügen. AcFormPropertySettings: Mit dieser Standardkonstante können Sie alle Aktionen im Formular durchführen acFormReadOnly: Der Anwender kann die Datensätze im Formular nur ansehen.
Fenstermodus	Optional. Legt fest, wie das Formular angezeigt werden soll. acWindowNormal: öffnet das Formular in der Standardansicht. acHidden: blendet das Formular aus. acIcon: Das Formular wird unten am Bildschirm in Form eines kleinen Symbols in der Titelleiste angezeigt. acDialog: Öffnet das Formular als Dialogfeld. Dabei können Sie mit einer anderen Aufgabe erst weiterarbeiten, wenn Sie das Formular wieder geschlossen haben.
ÖffnenArgumente	Optional. Dabei können Sie dem Formular bereits beim Öffnen bestimmte Werte bzw. Einstellungen mitgeben.

Tabelle 113: Die Argumente der Methode OpenForm (Forts.)

Im folgenden Beispiel aus Listing 350 wird das Formular Artikel in der Dialogansicht aufgerufen und ein Filter voreingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=====================================================

Sub FormularAufrufen()
   On Error GoTo fehler
   DoCmd.OpenForm "Artikel", acNormal, "Einzelpreis", _
      "Einzelpreis > 50", , acDialog
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 350: Formular öffnen und Filter einstellen

Übergeben Sie der Methode OpenForm den Namen des Formulars, das Sie öffnen möchten. Danach legen Sie den Anzeigemodus fest und übergeben der Methode einen Filter.

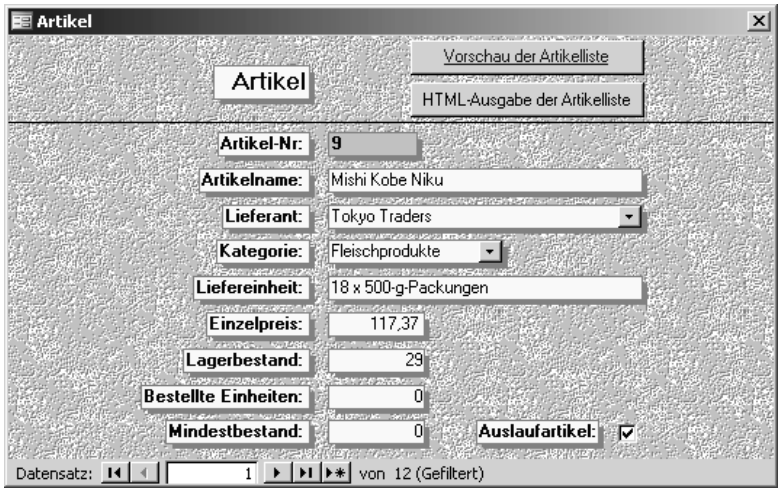


Abbildung 209: Das Formular mit eingestelltem Filter öffnen

Soll ein Formular wieder geschlossen werden, dann setzen Sie hierfür die Methode `Close` ein. Die Syntax dieser Methode lautet:

```
Close(Objekttyp, Objektname, Speichern)
```

Argument	Beschreibung
`Objekttyp`	Erforderlich. Gibt den Typ des Objekts an. Zur Verfügung stehen dabei folgende Konstanten: `acDataAccessPage`, `acDefault` (Standard), `acDiagram`, `acForm`, `acFunction`, `acMacro`, `acModule`, `acQuery`, `acReport`, `acServerView`, `acStoredProcedure`, `acTable`
`Objektname`	Optional. Enthält den Namen des Formulars.
`Speichern`	Optional. Gibt an, wie bzw. ob Änderungen vor dem Schließen des Formulars gespeichert werden sollen. Dazu stehen folgende Konstanten zur Verfügung: `AcSaveNo`: Bei dieser Konstante werden Änderungen nicht gespeichert. `AcSavePrompt`: Hier wird nachgefragt, ob Änderungen am Formular gespeichert werden sollen. Dies ist die Standardeinstellung. `AcSaveYes`: Bei dieser Konstante werden Änderungen beim Schließen des Formulars ohne Rückfrage gespeichert.

Tabelle 114: Die Argumente der Methode Close

Im Makro aus Listing 351 wird das Formular `Artikel` geschlossen. Vorher erfolgt jedoch eine Prüfung, ob das betreffende Formular überhaupt geöffnet ist.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=================================================

Sub FormularSchließen()
   On Error GoTo fehler
   If CurrentProject.AllForms("Artikel").IsLoaded = True Then
      DoCmd.Close acForm, "Artikel", acSaveYes
   End If
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 351: Formular schließen

Über die Funktion `IsLoaded` können Sie feststellen, ob ein bestimmtes Formular geöffnet ist. Wenn ja, dann meldet diese Funktion den Wert `True` zurück. In diesem Fall wenden Sie die Methode `Close` an, um das Formular zu schließen.

286 Aktivierreihenfolge anpassen

Wenn Sie alle Formularfelder in ein Formular integriert haben, können Sie die Aktivierreihenfolge festlegen. Damit ist die Reihenfolge der Formularfelder gemeint, wenn Sie das Formular öffnen und mit der Taste ⇆ von einem Formularfeld zum anderen springen. Möchten Sie rückwärts springen, drücken Sie die Tastenkombination ⇧ + ⇆. Standardmäßig ist diese Reihenfolge schon vorgegeben. Die Aktivierfolge entspricht der Reihenfolge, in der Sie die einzelnen Formularfelder in Ihr Formular eingefügt haben.

Die Aktivierreihenfolge wird wie folgt festgelegt:

1. Wechseln Sie in die Entwurfsansicht des Formulars.
2. Klicken Sie mit der rechten Maustaste mitten ins Formular und wählen Sie aus dem Kontextmenü den Befehl AKTIVIERREIHENFOLGE.
3. Möchten Sie die Reihenfolge von Access selbst bestimmen lassen, klicken Sie auf die Schaltfläche AUTOMATISCH. Damit legt Access die Aktivierreihenfolge der Steuerelemente von links nach rechts und von oben nach unten fest. Die Felder werden in der Datenblattansicht des Formulars entsprechend der neuen Aktivierreihenfolge angeordnet.
4. Möchten Sie hingegen eine andere Aktivierreihenfolge für Ihre Formularfelder haben, klicken Sie auf die jeweilige graue Schaltfläche und ziehen Sie diese an die gewünschte Stelle.
5. Bestätigen Sie Ihre Einstellung mit OK.

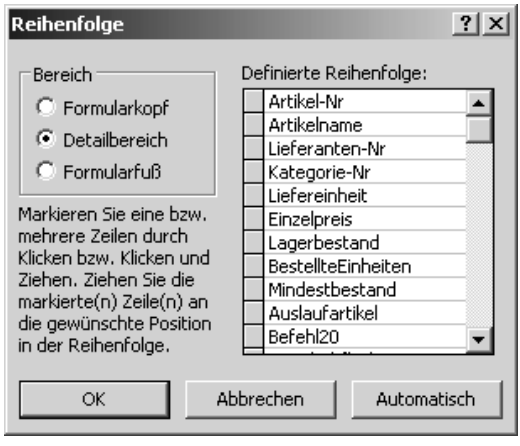


Abbildung 210: Aktivierreihenfolge festlegen

Um die Aktivierreihenfolge zu testen, drücken Sie in der Entwurfsansicht die Taste F5. Das Formular wird geöffnet und Sie können die Aktivierreihenfolge nun testen, indem Sie mit der Taste ⇆ die einzelnen Felder anspringen. Um wieder zum Formularentwurf zurückzugelangen, klicken Sie das Formular mit der rechten Maustaste an und wählen im Kontextmenü den Befehl FORMULARENTWURF aus.

287 Formularhintergrund einstellen

Sie sind nicht an die Hintergrundfarbe GRAU gebunden, die zwar standardmäßig für Formulare in Access verwendet wird, aber nicht vorgeschrieben ist. Die Hintergrundfarbe für Formulare stellen Sie in Access über die Eigenschaft `BackColor` ein. Für die Farbzusammensetzung wenden Sie die Funktion `RGB` an. Bei `RGB` handelt es sich um ein System von Farbwerten, das verwendet wird, um Farben als eine Mischung aus Rot (R), Grün (G) und Blau (B) zusammenzusetzen. Die Farbe wird als eine Gruppe aus drei ganzen Zahlen definiert (R,G,B), wobei jede ganze Zahl im Bereich von 0 bis 255 liegt. Ein Wert von 0 bedeutet, dass diese Farbkomponente nicht vorhanden ist. Ein Wert von 255 zeigt dagegen an, dass diese Farbkomponente mit ihrer höchsten Intensität verwendet wird.

Im Formular aus Listing 352 wird nach jedem Klicken auf die Schaltfläche eine andere Hintergrundfarbe eingestellt. Die Auswahl der Farbe erfolgt jeweils nach dem Zufallsprinzip. Dabei werden die verwendeten `RGB`-Werte in der Titelleiste des Formulars ausgegeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Textfelder
'=================================================
```

Listing 352: Hintergrundfarbe des Formulars festlegen

```
Private Sub Befehl7_Click()
   'Hintergrund der Form anpassen
   Dim intRot As Integer
   Dim intGrün As Integer
   Dim intBlau As Integer

   intRot = Int((255 * Rnd) + 0)
   intGrün = Int((255 * Rnd) + 0)
   intBlau = Int((255 * Rnd) + 0)

   Detailbereich.BackColor = RGB(intRot, intGrün, intBlau)
   Form.Caption = _
      "Rot: " & intRot & " Grün: " & intGrün & " Blau: " & intBlau
End Sub
```

Listing 352: Hintergrundfarbe des Formulars festlegen (Forts.)

Setzen Sie bei jedem Klick auf die Schaltfläche FARBE die Funktion `Rnd` ein, um Zufallszahlen im Bereich zwischen 0 und 255 zu erzeugen. Mithilfe der Funktion `Int` wird die Zufallszahl in einen ganzzahligen Wert umgewandelt. Alle so ermittelten Werte werden in den Variablen `Rot`, `Grün` und `Blau` gespeichert und im Anschluss als Hintergrund der Eigenschaft `BackColor` und der Funktion `RGB` übergeben. Die Ausgabe der Zusammensetzung der `RGB`-Funktion erfolgt über die Eigenschaft `Caption` des Formulars.

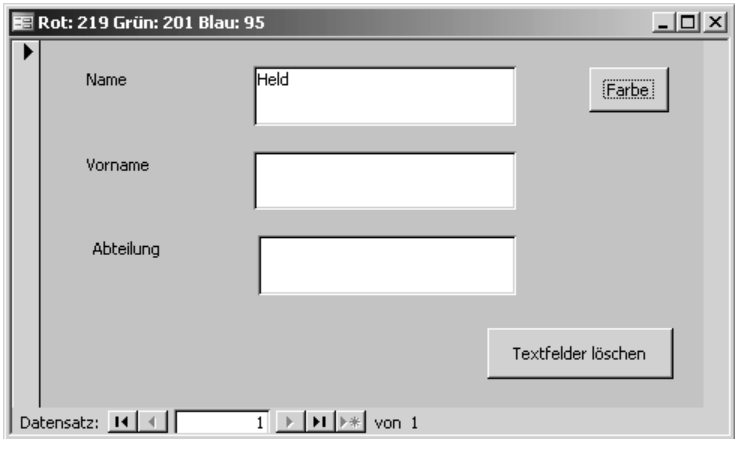


Abbildung 211: Zufallsfarbe per Mausklick einstellen

288 Formular ohne Navigationsschaltflächen aufrufen

Möchten Sie verhindern, dass ein Anwender die Navigationsschaltflächen benutzen kann, dann können Sie die Anzeige dieser Schaltflächen unterdrücken. Dazu setzen Sie das Formularereignis des Formulars `Form_Load` ein, das automatisch ausgeführt wird, wenn das Formular angezeigt wird. Dort blenden Sie die Schaltflächen aus.

Um das Ereignis einzustellen, gehen Sie wie folgt vor:

1. Öffnen Sie das Formular `Artikel2` in der Entwurfsansicht.
2. Klicken Sie mit der rechten Maustaste auf das Formular und wählen Sie den Eintrag EIGENSCHAFTEN aus dem Kontextmenü.
3. Stellen Sie im Kombinationsfeld den Eintrag FORMULAR ein.
4. Wechseln Sie auf die Registerkarte EREIGNIS.

Abbildung 212: Ein Formularereignis einstellen

5. Setzen Sie den Mauszeiger in das Feld `Beim Laden`.
6. Klicken Sie auf das Symbol mit den drei Punkten.
7. Im Dialog GENERATOR AUSWÄHLEN wählen Sie den Eintrag CODE-GENERATOR aus dem Listenfeld und bestätigen Sie mit OK.

Ergänzen Sie den noch leeren Ereignisrahmen wie in Listing 353 angezeigt:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Form_Load()
   With Me
```

Listing 353: Navigationsschaltflächen ausblenden

```
        .NavigationButtons = False
        .RecordSelectors = False
    End With
End Sub
```

Listing 353: Navigationsschaltflächen ausblenden (Forts.)

Setzen Sie die Eigenschaft `NavigationButtons` auf den Wert `False`, um die Navigationsschaltflächen am unteren Rand des Formulars auszublenden. Möchten Sie zusätzlich auch noch den Datensatzmarkierer am linken Dialograd ausblenden, dann weisen Sie der Eigenschaft `RecordSelectors` ebenso den Wert `False` zu.

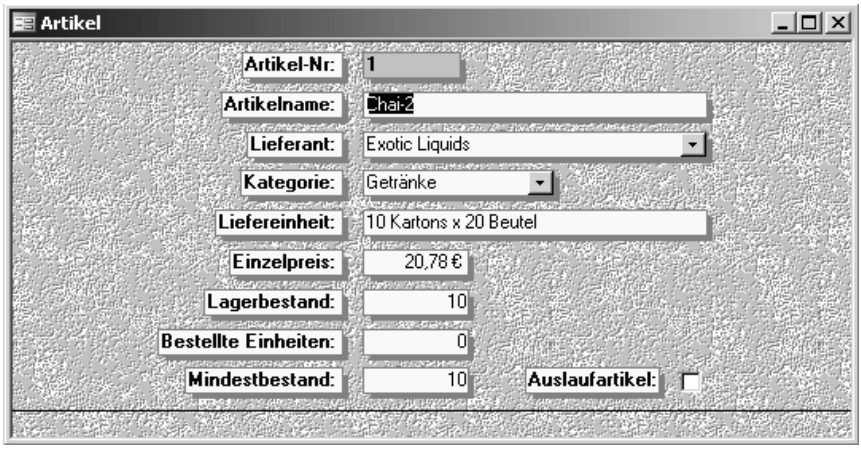


Abbildung 213: Die Steuerungsschaltflächen wurden ausgeblendet.

Hinweis

In Kapitel 8 dieses Buchs können Sie noch mehr Beispiele zu Ereignissen nachschlagen.

289 Hintergrund und Schriftfarbe von Textfeldern anpassen

Im folgenden Beispiel aus Listing 354 werden alle Textfelder des Formulars `Artikel1` mit der Hintergrundfarbe Gelb sowie der Schriftfarbe Rot ausgestattet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
```

Listing 354: Hintergrund gelb und Schriftfarbe rot

```
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Sub HintergrundUndSchriftAnpassen()
   Dim frm As Form
   Dim ctl As Control
   Dim lngFarbeRot As Long
   Dim lngFarbeGelb As Long

   lngFarbeRot = RGB(255, 0, 0)
   lngFarbeGelb = RGB(255, 255, 0)
   Set frm = Form_Artikel1

   For Each ctl In frm.Controls
      If ctl.ControlType = acTextBox Then
         ctl.BackColor = lngFarbeGelb
         ctl.ForeColor = lngFarbeRot
      End If
   Next ctl
End Sub
```

Listing 354: Hintergrund gelb und Schriftfarbe rot (Forts.)

Über die Eigenschaft `BackColor` können Sie die Hintergrundfarbe festlegen. Mithilfe der Eigenschaft `ForeColor` bestimmen Sie die Schriftfarbe.

Über den Einsatz der Funktion `RGB` können Sie sich eine Farbe zusammenmischen. Als Argumente müssen Sie hierbei drei Werte angeben. Der erste Wert steht für den Rotanteil der Farbe, der zweite für den grünen Anteil und der dritte für den blauen Anteil der gewählten Farbe. Die folgende Tabelle listet einige typische Farbmischungen auf.

Farbe	Rotwert	Grünwert	Blauwert
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 115: Die Standardfarben über die Funktion RGB

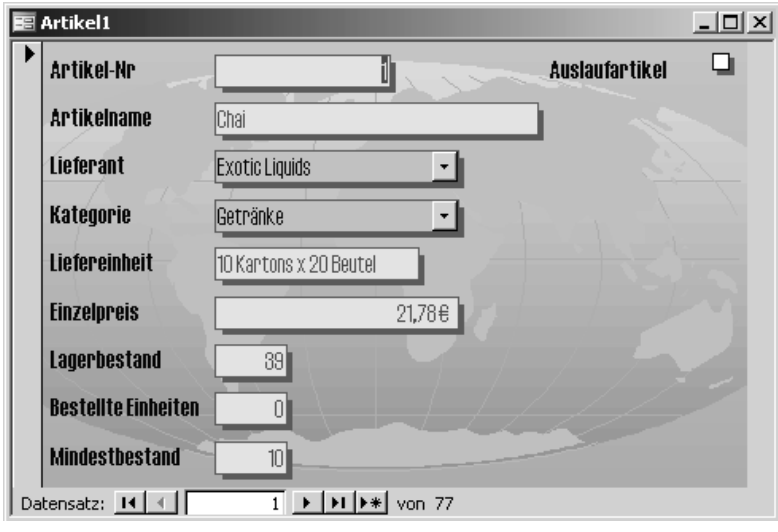


Abbildung 214: Die Formatierungen der Felder wurden vorgenommen.

290 Startposition des Cursors festhalten

Mithilfe der Eigenschaft `SelStart` können Sie die Startposition des markierten Textabschnitts eines Textfelds festlegen. Sollte noch kein Text eingefügt worden sein, legen Sie über diese Eigenschaft die Einfügemarke des Mauszeigers fest. Standardmäßig wird beim Aktivieren eines Textfelds immer der komplette Text markiert.

Folgende alternative Möglichkeiten der Markierung von Text gibt es bei Textfeldern.

Anweisung	Beschreibung
`Me!Name.SelStart = 0`	Setzt den Mauszeiger vor das erste Zeichen im Feld.
`Me!Name.SelStart = Me!Name.SelLength`	Setzt den Mauszeiger vor das erste Zeichen im Feld.

Tabelle 116: Alternative Markierungsformen für Textfelder

291 Auswahl aufheben

Standardmäßig wird in einem Textfeld immer der ganze Text beim Anspringen markiert. Diese Markierung können Sie aufheben, indem Sie die Eigenschaft `SelStart` sowie die Eigenschaft `SelLength` auf den Wert `0` setzen.

Im folgenden Beispiel aus Listing 355 wird das Ereignis aus Listing 353 noch ein wenig erweitert. Beim Laden des Formulars wird neben dem Ausblenden der Navigationsschaltflächen auch der Mauscursor auf das zweite Feld des Formulars gesetzt und dabei keine Markierung des Textes vorgenommen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=================================================

Private Sub Form_Load()
   With Me
      .NavigationButtons = False
      .RecordSelectors = False
      .Artikelname.SetFocus
      .Artikelname.SelStart = 0
      .Artikelname.SelLength = 0
   End With
End Sub
```

Listing 355: Die Standardmarkierung in Textfeldern aufheben

Über die Methode `SetFocus` setzen Sie den Mauszeiger in das gewünschte Textfeld.

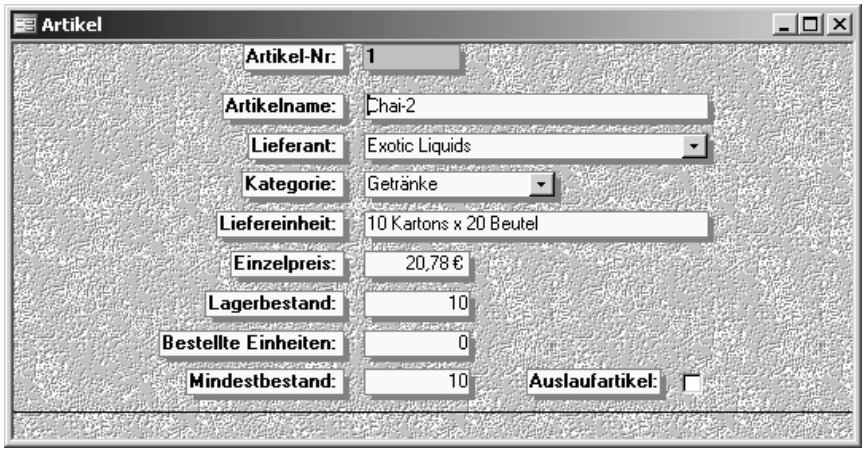


Abbildung 215: Der Mauszeiger steht zu Beginn des Textfelds Artikelname.

292 Alte Werte festhalten

Bevor Sie einen Datensatz in einem Formular ändern, können Sie die alten Formularinhalte über die Eigenschaft `OldValue` sichern lassen. Damit haben Sie die Möglichkeit, Ihre Eingaben bzw. Änderungen rückgängig zu machen.

Im folgenden Beispiel aus Listing 356 wurde im Formular `Artikel1` eine Schaltfläche eingefügt, die beim Klicken die aktuellen sowie die geänderten Werte in das Direktfenster der Entwicklungsumgebung schreibt.

```
'================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'================================================

Private Sub Befehl20_Click()
   Dim ctl As Control
   On Error GoTo Err_Befehl20_Click

   For Each ctl In Me.Controls
      With ctl
         If .ControlType = acTextBox Then
            Debug.Print "alter Wert" & .OldValue & vbLf & "neuer Wert" & .Value
         End If
      End With
   Next ctl
   Exit Sub


   Err_Befehl20_Click:
      MsgBox Err.Description
End Sub
```

Listing 356: Alte und neue Werte festhalten

Die Eigenschaft `ControlType` gibt Auskunft darüber, ob es sich um ein Textfeld handelt. Wenn ja, dann lässt sich über die Eigenschaft `OldValue` der »alte« Wert vor der Änderung eines Textfelds festhalten. Den aktuellen Wert fragen Sie über die Eigenschaft `Value` ab.

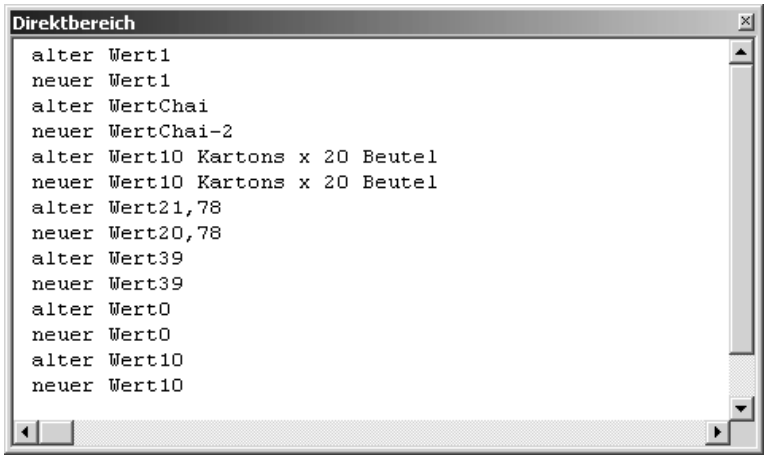


Abbildung 216: Der Preis und die Bezeichnung wurden geändert.

293 QuickInfo hinzufügen

Setzen Sie die Eigenschaft `ControlTipText` ein, um einem Textfeld eine QuickInfo hinzuzufügen.

Im folgenden Beispiel aus Listing 357 wird jedem Textfeld des Formulars `Artikel1` eine QuickInfo zugewiesen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=================================================

Sub QuickInfoHinzufügen()
   Dim frm As Form
   Dim ctl As Control

   Set frm = Form_Artikel1

   For Each ctl In frm.Controls
      With ctl
         If .ControlType = acTextBox Then
            .ControlTipText = "Hier bitte Eingabe vornehmen!"
         End If
      End With
   Next ctl
End Sub
```

Listing 357: Kurzinformationen in Textfelder einfügen

Über die Eigenschaft `ControlType` prüfen Sie, ob es sich bei dem jeweiligen Steuerelement um ein Textfeld handelt. Wenn ja, dann weisen Sie der Eigenschaft `ControlTipText` einen Text zu, der dann im Formular angezeigt wird, wenn Sie den Mauszeiger in das entsprechende Textfeld setzen.

294 Textfelder sperren

Setzen Sie die Eigenschaft `Enabled` ein, die Sie auf den Wert `True` setzen. Damit deaktivieren Sie ein Textfeld. Es wird dann abgeblendet im Formular angezeigt und kann nicht editiert werden.

Mithilfe der Eigenschaft `Locked`, die Sie auf den Wert `True` setzen, können Sie ein Textfeld vor Veränderungen schützen. Dieses können Sie zwar standardmäßig noch ansteuern, eine Bearbeitung ist aber nicht möglich.

In den meisten Fällen werden die Eigenschaften `Enabled` und `Locked` gemeinsam eingesetzt, denn es macht nicht immer Sinn, ein Textfeld aktiviert zu lassen, wenn der Anwender keine Änderungen daran ausführen darf. In diesem Fall ist es besser, beide Eigenschaften kombiniert einzusetzen.

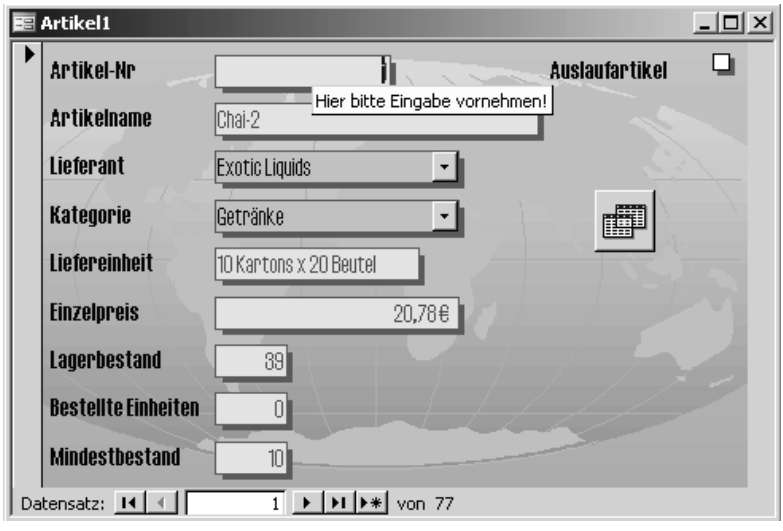


Abbildung 217: QuickInfos hinzufügen

Im folgenden Beispiel aus Listing 358 wird für das Feld ARTIKELNAME im Formular ARTIKEL1 festgelegt, dass darin keine Änderungen vorgenommen werden dürfen. Diese Aufgabe führen Sie am besten gleich direkt beim Öffnen des Formulars durch.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=================================================

Private Sub Form_Open(Cancel As Integer)
   On Error GoTo fehler

   Me!Artikelname.Locked = True
   Me!Artikelname.Enabled = True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 358: Das Feld Artikelname wurde gesperrt.

Soll die Sperre auf alle Textfelder des Formulars erweitert werden, dann passen Sie das Makro aus Listing 358 wie in Listing 359 gezeigt an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Private Sub Form_Open(Cancel As Integer)
   Dim ctl As Control
   On Error GoTo fehler

   For Each ctl In Me.Controls
      If ctl.ControlType = acTextBox Then
         ctl.Locked = True
         ctl.Enabled = True
      End If
   Next ctl
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 359: Alle Textfelder eines Formulars werden gesperrt.

Das Sperren von Textfeldern kann beispielsweise auch abhängig vom Anwender vorgenommen werden. In Kapitel 8 dieses Buchs können Sie nachschlagen, wie Sie mithilfe von Ereignissen diesbezüglich eine anwenderorientierte Lösung mit Textfeldern aufsetzen können.

295 Textfelder ein- und ausblenden

Textfelder können ausgeblendet werden, indem die Eigenschaft `Visible` auf den Wert `False` gesetzt wird. Um diese wieder einzublenden, wird der Wert dieser Eigenschaft auf den Wert `True` zurückgesetzt.

Im folgenden Beispiel aus Listing 360 wurde in das Formular `Artikel1` eine Umschaltfläche eingefügt, um dynamisch Felder ein- und auszublenden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Sub Umschaltfläche23_Click()
   'Textfelder ein- und ausblenden
```

Listing 360: Textfelder ein- und ausblenden

```
    If Umschaltfläche23.Value = 0 Then
       Umschaltfläche23.Caption = "Lager Ein"
       Me!Lagerbestand.Visible = False
       Me!BestellteEinheiten.Visible = False
       Me!Mindestbestand.Visible = False
       Me!Auslaufartikel.Visible = False
    Else
       Umschaltfläche23.Caption = "Lager aus"
       Me!Lagerbestand.Visible = True
       Me!BestellteEinheiten.Visible = True
       Me!Mindestbestand.Visible = True
       Me!Auslaufartikel.Visible = True
    End If
End Sub
```

Listing 360: Textfelder ein- und ausblenden (Forts.)

Das Ein- und Ausblenden der Textfelder wird je nach Status der Textfelder vorgenommen. Sind die Textfelder gerade ausgeblendet, dann werden sie beim nächsten Klick auf die Umschaltfläche wieder eingeblendet. Sind die Textfelder momentan sichtbar, dann werden sie mit einem Klick ausgeblendet.

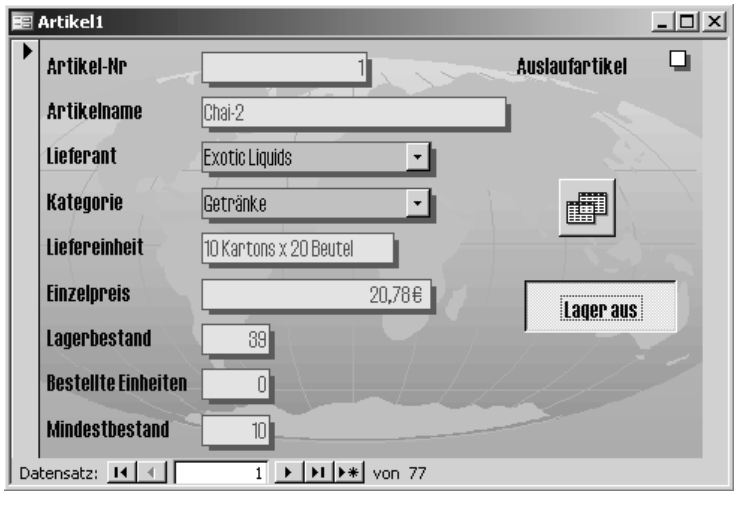


Abbildung 218: Lagerdaten beim nächsten Klick ausblenden

296 Textfelder begrenzen

Möchten Sie nur eine bestimmte Anzahl von Zeichen in einem Textfeld zulassen, dann können Sie schon während der Eingabe überprüfen, wann diese Grenze erreicht wird, und dementsprechend darauf reagieren.

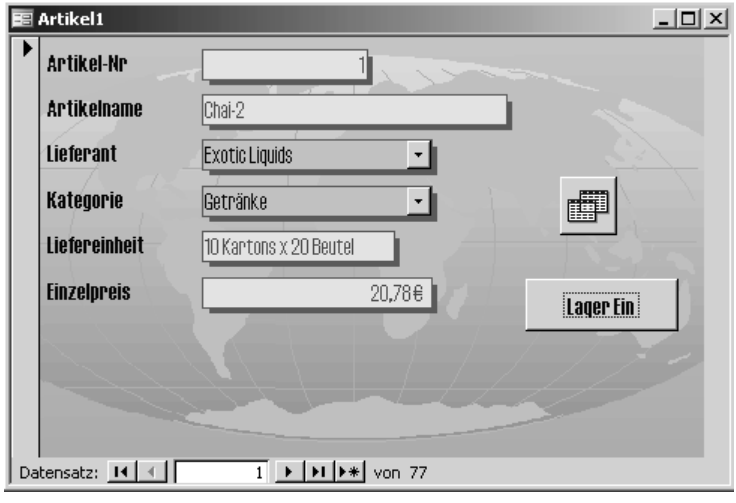


Abbildung 219: Lagerdaten beim nächsten Klick einblenden

Im folgenden Beispiel aus Listing 361 wurde in `Artikel1` das Textfeld `Lagerort` eingefügt, welches über das Ereignis `KeyPress` überwacht wird. In diesem Feld dürfen maximal zehn Zeichen erfasst werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Private Sub Text24_KeyPress(KeyAscii As Integer)
   If Len(Text24.Text) > 10 Then
      MsgBox "Es dürfen nur 10 Zeichen eingegeben werden!"
      Text24.Text = Left(Text24.Text, 10)
   End If
End Sub
```

Listing 361: Die Eingabe auf zehn Zeichen beschränken

Die Überprüfung der Eingabelänge des Textfelds erfolgt bei jeder Eingabe neu über die Funktion `Len`. Umfasst die Eingabe mehr als zehn Zeichen, dann wird automatisch eine Kürzung vorgenommen, indem mithilfe der Funktion `Left` vom linken Rand des Textfelds aus genau zehn Zeichen übertragen werden.

297 Textfelder überwachen

Soll eine Änderung an einem Textfeld überwacht werden, dann können Sie dies über ein Ereignis umsetzen. Im folgenden Beispiel aus Listing 362 wird im Formular `Artikel1` das Feld `Bestand` überwacht. Um dieses Feld so einzustellen, gehen Sie wie folgt vor:

1. Öffnen Sie Ihr Formular in der Entwurfsansicht.
2. Klicken Sie mit der rechten Maustaste das Datenfeld im Formular an, das Sie auf diese Weise überwachen möchten, und wählen Sie aus dem Kontextmenü den Befehl EIGENSCHAFTEN.
3. Wechseln Sie zur Registerkarte EREIGNIS.
4. Setzen Sie den Mauszeiger ins Feld `Bei Geändert` und klicken Sie auf die Schaltfläche ganz rechts im Feld.
5. Wählen Sie im Dialogfeld GENERATOR AUSWÄHLEN den Eintrag CODE-GENERATOR.
6. Bestätigen Sie Ihre Wahl mit OK.
7. Ergänzen Sie den noch leeren Ereignisrahmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Private Sub Lagerbestand_Dirty(Cancel As Integer)
   'Lagerbestandsänderung
   Debug.Print "Änderung von : " & Environ("Username") & vbLf & _
      " am " & Date & " um " & time & vblf & _
      "Artikelname: " & Me.Artikelname & vbLf & _
      "Lagerbestand: " & Me.Lagerbestand.Value
End Sub
```

Listing 362: Lagerbestandsänderung festhalten

Über die Funktion `Environ` und die Konstante `Username` können Sie feststellen, wer die Änderung vorgenommen hat. Die Funktionen `Date` und `Time` liefern das Änderungsdatum sowie die Uhrzeit der Änderung.

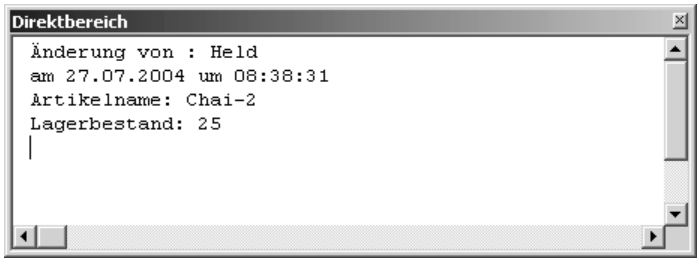


Abbildung 220: Das Feld Lagerbestand wurde überwacht.

298 Textfelder initialisieren

Bei ungebundenen Formularen mit diversen Textfeldern können Sie eine Funktion schreiben, die alle Textfelder in einem Aufwasch initialisiert.

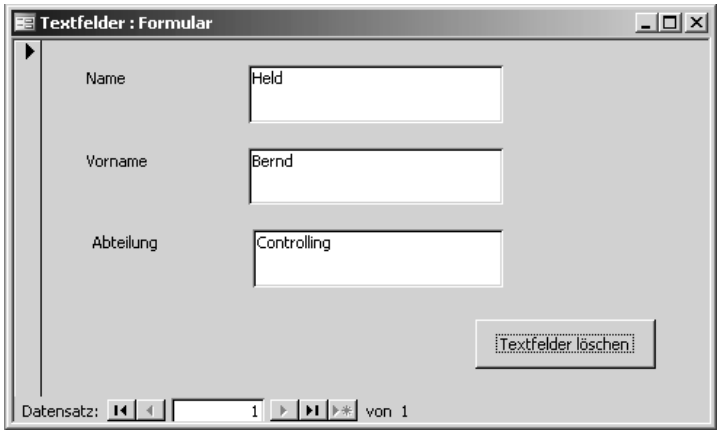


Abbildung 221: Die Textfelder sollen geleert werden.

Um alle Textfelder in einem Formular zu leeren, starten Sie das Makro aus Listing 363.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Textfelder
'=====================================================


Private Sub Befehl6_Click()
   'Textfelder löschen
   Dim tb As Textbox

   For Each tb In Me.Controls
      If TypeName(tb) = "TextBox" Then tb.Value = ""
   Next tb
End Sub
```

Listing 363: Alle Textfelder leeren

Über die Funktion `Typename` können Sie ermitteln, ob es sich beim jeweiligen Steuerelement um ein Textfeld handelt. Wenn ja, dann gibt diese Funktion den Text `TextBox` zurück. Achten Sie bei der Abfrage auf die richtige Schreibweise: `Textbox` ist nicht gleich `TextBox`.

299 Textfelder prüfen

Beim Verlassen eines Textfelds können Sie überprüfen, ob eine richtige Eingabe vorgenommen wurde. Im Beispiel aus Listing 364 wird überprüft, ob überhaupt eine Eingabe vorgenommen wurde. Wenn ja, dann erfolgt die zweite Prüfung, ob ein alphanumerischer Eintrag erfasst wurde.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Textfelder
'=================================================

Private Sub Text0_Exit(Cancel As Integer)
   If Len(Text0.Text) = 0 Then
      MsgBox "Sie müssen eine Eingabe vornehmen!"
      Exit Sub
   End If
   If IsNumeric(Text0.Text) = True Then
       MsgBox "Sie müssen einen alphanumerischen Wert eingeben!"
       Cancel = True
   End If
End Sub
```

Listing 364: Überprüfung des Textfelds

Prüfen Sie im ersten Schritt, ob überhaupt eine Eingabe gemacht wurde. Wenn nicht, geben Sie eine Meldung aus und beenden Sie das Makro mit der Anweisung `Exit Sub`. Wurde eine Eingabe vorgenommen, überprüfen Sie mithilfe der Funktion `IsNumeric`, ob es sich dabei um einen gültigen numerischen Wert handelt. Gibt der Anwender versehentlich eine Zahl ein, dann geben Sie eine Meldung auf dem Bildschirm aus.

300 Listenfelder

Die wichtigsten Eigenschaften, Funktionen und Methoden für Listenfelder werden im Folgenden aufgeführt. Als Voraussetzung dafür legen Sie ein neues Formular an und fügen ein ungebundenes Listenfeld ein. Des Weiteren fügen Sie eine Schaltfläche sowie das Steuerelement BILD hinzu.

301 Listenfeld füllen

Über die Methode `AddItem` können Sie einem Listenfeld einen Eintrag hinzufügen. Die Syntax lautet:

```
Ausdruck.AddItem(Item, Index)
```

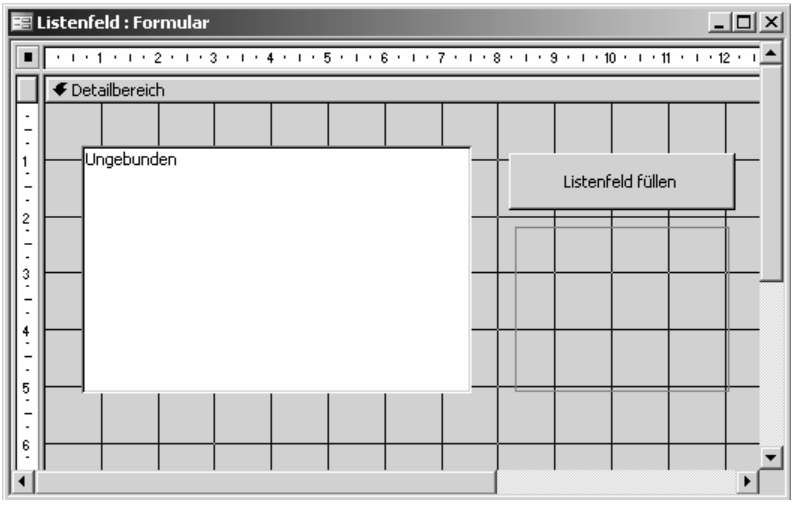


Abbildung 222: Die Ausgangssituation

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der unter anderem ein Listenfeld oder eine Kombinationsfeldliste zurückgibt.
Item	Erforderlich. Der angezeigte Text für das neue Element.
Index	Optional. Die Position des Elements in der Liste. Wird dieses Argument nicht angegeben, wird das Element am Ende der Liste hinzugefügt.

Tabelle 117: Die Argumente der Methode AddItem

Im Beispiel aus Listing 365 wird im Formular `Listenfeld` der Schaltfläche ein Makro zugewiesen, welches die Namen sowie den Pfad aller Bilddateien aus einem Verzeichnis sowie den darunter liegenden Verzeichnissen im Listenfeld einfügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl2_Click()
   'Bilddateien einlesen
   Dim varDatei As Variant

   With Application.FileSearch
      .NewSearch
      .LookIn = "c:\Eigene Dateien"
      .FileName = "*.jpg"
```

Listing 365: Listenfeld mit Namen von Bilddateien füllen

```
        .SearchSubFolders = True

        If .Execute() > 0 Then
           For Each varDatei In .FoundFiles
              Me!Liste0.AddItem varDatei
           Next varDatei
        End If
     End With
  End Sub
```

Listing 365: Listenfeld mit Namen von Bilddateien füllen (Forts.)

Mithilfe des Objekts `FileSearch` kann nach bestimmten Dateien in Verzeichnissen gesucht werden. Dieses Objekt verfügt über einige Eigenschaften, die angegeben werden können. Über die Eigenschaft `FileName` geben Sie vor, nach welchen Dateien gesucht werden soll. Dabei kann die Endung von Grafikdateien `JPG` mit einem Sternchen angegeben werden. Über die Eigenschaft `LookIn` legen Sie fest, wo Access nach den Grafikdateien suchen soll. Hier können Sie das Laufwerk sowie das Verzeichnis angeben. Sollen noch darunter liegende Verzeichnisse durchsucht werden, dann kann dies über die Eigenschaft `SearchSubFolders` festgelegt werden. Diese Eigenschaft wird auf den Wert `True` gesetzt, wenn die angegebene Suche alle Unterordner im durch die `LookIn`-Eigenschaft angegebenen Ordner einschließen soll. Die Methode `Execute` führt die jetzt näher spezifizierte Suche anschließend aus.

Nach der Suche sind alle gefundenen Dateien im Objekt `FoundFiles` verzeichnet. Diese gefundenen Dateien werden in einer anschließenden Schleife nacheinander verarbeitet. Innerhalb der Schleife werden die einzelnen Dateinamen über die Methode `AddItem` ins Listenfeld eingefügt.

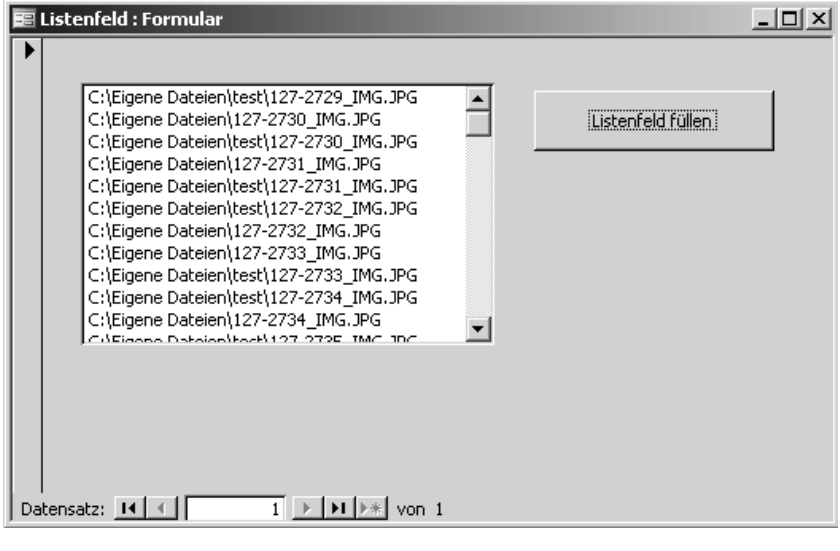


Abbildung 223: Das Listenfeld wurde gefüllt.

302 Listenfeldauswahl verarbeiten

Im nächsten Schritt soll bei der Auswahl eines Listenfeldeintrags das entsprechende Bild im Kleinformat in das Steuerelement BILD eingefügt werden. Die Auswahl eines Bilds erfolgt über das Ereignis `Click` des Listenfelds. Bevor Sie aber an die Programmierung des Ereignisses gehen, bereiten Sie das Bildsteuerelement wie folgt für diese Aufgabe vor:

1. Klicken Sie das Bildsteuerelement in der Entwurfsansicht mit der rechten Maustaste an und wählen Sie den Befehl EIGENSCHAFTEN aus dem Kontextmenü.
2. Wechseln Sie auf die Registerkarte FORMAT.
3. Stellen Sie bei der Eigenschaft GRÖSSENANPASSUNG den Eintrag DEHNEN ein.
4. Wählen Sie bei der Eigenschaft BILDAUSRICHTUNG den Eintrag MITTE ein.
5. Beenden Sie diese Festlegung und speichern Sie die Änderungen.

Stellen Sie nun das `Click`-Ereignis des Listenfelds wie in Listing 366 gezeigt ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Liste0_Click()
   'Ausgewähltes Bild übertragen
   Dim intz As Integer

   intz = Me!Liste0.ListIndex
   Bild4.Picture = Me!Liste0.ItemData(intz)
End Sub
```

Listing 366: Der aktuell ausgewählte Eintrag im Listenfeld wird ausgewertet.

Mit der `ListIndex`-Eigenschaft können Sie herausfinden, welches Element in einem Listenfeld oder Kombinationsfeld ausgewählt ist. Diesen numerischen Wert übergeben Sie der Variablen `intZ`.

Die `ItemData`-Eigenschaft gibt die Daten in der gebundenen Spalte für die angegebene Zeile in einem Kombinationsfeld oder Listenfeld zurück. Die Syntax dieser Eigenschaft lautet:

```
Ausdruck.ItemData(Index)
```

Argument	Beschreibung
Ausdruck	Erforderlich. Ein Ausdruck, der unter anderem ein Listenfeld oder eine Kombinationsfeldliste zurückgibt.
Index	Erforderlich. Die Zeile im Kombinations- oder Listenfeld, die die zurückzugebenden Daten enthält. Zeilen in Kombinations- und Listenfeldern sind, beginnend bei Null, indiziert.

Tabelle 118: Die Argumente der Eigenschaft ItemData

Übergeben Sie den kompletten Pfad des Bilds aus dem Listenfeld direkt dem Bildsteuerelement, indem Sie der Eigenschaft `Picture` diese Information geben.

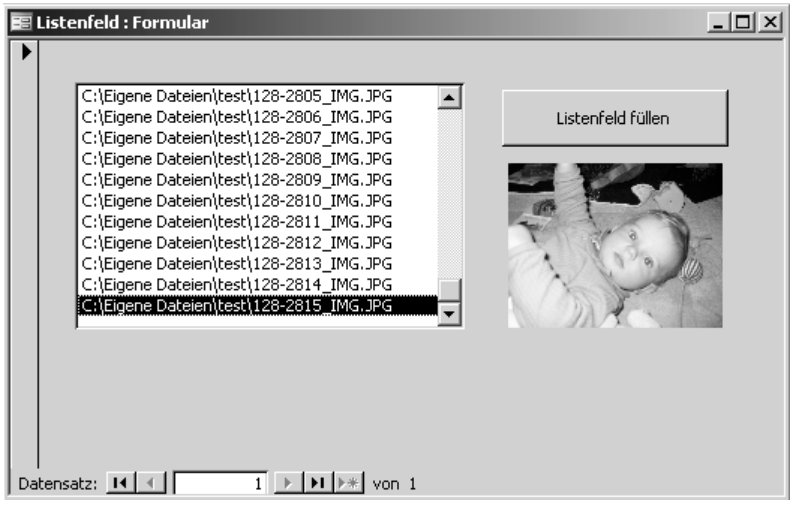


Abbildung 224: Bild nach Klick im Kleinformat anzeigen

303 Listenfeldeinträge zählen

Mit der `ListCount`-Eigenschaft können Sie feststellen, wie viele Zeilen ein Listenfeld oder der Listenfeldteil eines Kombinationsfelds enthält. Der erste Eintrag im Listenfeld fängt mit der 0 an.

304 Aktuellen Listenfeldeintrag löschen

Beim Löschen von Einträgen aus einem Listenfeld können Sie selbstverständlich auch einzelne Einträge löschen. Hier kommt es darauf an, ob Sie es zulassen, dass auch mehrere Einträge im Listenfeld gelöscht werden können. Im nächsten Makro aus Listing 367 werden jeweils der markierte Listeneintrag aus dem Listenfeld sowie die Grafikdatei auf der Festplatte entfernt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=================================================

Private Sub Befehl5_Click()
   'Ausgewählte Einträge löschen
   Dim intZ As Integer

   On Error GoTo fehler
   inz = Me!Liste0.ListIndex
   Kill Me!Liste0.ItemData(intZ)
   Me!Liste0.RemoveItem (intZ)
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 367: Markierten Eintrag im Listenfeld entfernen

Mithilfe der Eigenschaft `ListIndex` können Sie herausfinden, welcher Eintrag in einem Listenfeld gerade markiert ist. Der markierte Eintrag im Listenfeld gibt die genaue Position im Listenfeld an. Diese Position übergeben Sie der Methode `RemoveItem`, die diesen Eintrag dann aus dem Listenfeld entfernt. Die Syntax dieser Methode lautet:

```
Ausdruck.RemoveItem(Index)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der unter anderem ein Listenfeld oder eine Kombinationsfeldliste zurückgibt.
`Index`	Erforderlich. Das aus der Liste zu entfernende Element, das als Elementnummer oder Listenelementtext angegeben ist.

Tabelle 119: Die Argumente der Methode RemoveItem

Über die Anweisung `Kill` können Sie die dazugehörige Datei auf der Festplatte entfernen. Übergeben Sie hierfür den kompletten Pfad der Grafikdatei, den Sie direkt aus dem Listenfeld auslesen können.

305 Alle markierten Listenfeldeinträge löschen

Um aus einem Listenfeld, bei dem lediglich jeweils ein Eintrag ausgewählt werden kann, ein Listenfeld mit Mehrfachauswahl zu machen, klicken Sie das Listenfeld in der Entwurfsansicht mit der rechten Maustaste an, wählen den Befehl EIGENSCHAFTEN aus dem Kontext-

menü, wechseln zur Registerkarte ANDERE und aktivieren im Feld MEHRFACHAUSWAHL den Eintrag EINZELN.

Bei der folgenden Lösung aus Listing 368 wurde eine weitere Schaltfläche im Formular eingefügt und das `Click`-Ereignis eingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl6_Click()
   'Ausgewählte Einträge im Listenfeld löschen
   Dim intZeile As Integer
   Dim varI() As Integer
   Dim intZ As Integer

   With Me
      If .Liste0.ItemsSelected.Count = 0 Then
         Exit Sub
      Else
         ReDim varI(.Liste0.ItemsSelected.Count)
         For intZeile = 0 To .Liste0.ListCount - 1
            If .Liste0.Selected(intZeile) Then
               varI(intZ) = intZeile
               intZ = intZ + 1
            End If
         Next

         For intZ = .Liste0.ItemsSelected.Count - 1 To 0 Step -1
            .Liste0.RemoveItem varI(intZ)
         Next
      End If
   End With
End Sub
```

Listing 368: Alle markierten Listenfeldeinträge entfernen

Da in diesem Fall mehrere Einträge im Listenfeld markiert werden können, definieren Sie zu Beginn des Makros eine Array-Variable `VarI`. In dieser Variablen werden alle markierten Zeilen im Listenfeld gespeichert. Prüfen Sie zunächst, ob überhaupt ein Eintrag im Listenfeld markiert ist. Wenn ja, dann meldet die Eigenschaft `ListCount` einen Wert größer 0. In diesem Fall legen Sie das Datenfeld so groß an, dass alle markierten Einträge darin Platz finden. Diese Größe können Sie direkt über die Eigenschaft `ItemsSelected` abfragen. Im nächsten Schritt füllen Sie das Datenfeld mit den markierten Positionen. Ist das komplette Datenfeld `varI` gefüllt, entfernen Sie über eine Schleife, die am Ende des Listenfelds beginnt, die vorher ermittelten Listenfeldeinträge über die Methode `RemoveItem`.

306 Alle Listenfeldeinträge (de)markieren

Mithilfe der `ItemsSelected`-Eigenschaft können Sie ermitteln, welche Einträge bei einer Mehrfachauswahl im Listenfeld markiert sind.

Im folgenden Beispiel aus Listing 369 werden alle markierten Einträge im Listenfeld demarkiert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=================================================

Private Sub Befehl7_Click()
   'Alle Einträge demarkieren
   Dim varI As Variant
   Dim ctrl As Control

   Set ctrl = Me!Liste0
   For Each varI In ctrl.ItemsSelected
      ctrl.Selected(varI) = False
   Next varI
End Sub
```

Listing 369: Markierung im Listenfeld entfernen

Über die Eigenschaft `ItemsSelected` können Sie die markierten Einträge im Listenfeld prüfen. In einer Schleife arbeiten Sie alle markierten Einträge im Listenfeld ab und entfernen dabei die Markierung, indem Sie die Eigenschaft `Selected` auf den Wert `False` setzen.

Wenn Sie sehr viele Einträge in einem Listenfeld haben, dann können Sie auch die Funktion aus Listing 370 zur Verfügung stellen, mit der man sehr schnell alle Listenfeldeinträge markieren kann.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=================================================

Private Sub Befehl8_Click()
   'Alle Einträge markieren
   Dim ctrl As Control
   Dim intz As Integer

   For intz = 0 To Liste0.ListCount
```

Listing 370: Alle Listenfeldeinträge markieren

```
        Me!Liste0.Selected(intz) = True
    Next intz
End Sub
```

Listing 370: Alle Listenfeldeinträge markieren (Forts.)

In einer Schleife arbeiten Sie alle Listenfeldeinträge nacheinander ab. Innerhalb der Schleife setzen Sie die Eigenschaft `Selected` auf den Wert `True`, um den jeweiligen Listenfeldeintrag zu markieren.

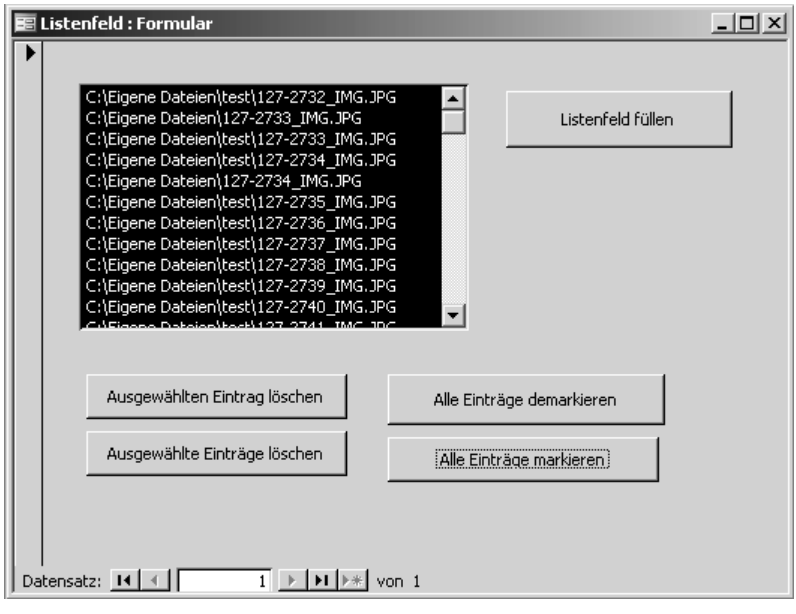


Abbildung 225: Alle Listenfeldeinträge blitzschnell markieren

307 Mehrspaltiges Listenfeld füllen

Im folgenden Beispiel aus Listing 371 werden in ein mehrspaltiges Listenfeld eines Formulars die in der Datenbank vorhandenen Tabellen sowie Abfragen eingelesen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_ListenfeldMehrspaltig
'=================================================

Private Sub Befehl2_Click()
```

Listing 371: Ein mehrspaltiges Listenfeld füllen

```
    'Tabellen und Abfragen in mehrspaltiges Listenfeld einlesen
    Dim Katalog As ADOX.Catalog
    Dim TabInfo As ADOX.Table

    Set Katalog = New ADOX.Catalog
    Katalog.ActiveConnection = CurrentProject.Connection

    For Each TabInfo In Katalog.Tables
       With TabInfo
          Me!Liste0.AddItem .Name & ";" & .Type
       End With
    Next

    Set Katalog = Nothing
End Sub
```

Listing 371: Ein mehrspaltiges Listenfeld füllen (Forts.)

Um ein mehrspaltiges Listenfeld zu füllen, können Sie die einzelnen Informationen getrennt durch ein Semikolon an die Methode `AddItem` übergeben. Dabei liefert die Eigenschaft `Type` die Information, ob es sich um eine Tabelle (= `TABLE`) oder um eine Abfrage (= `VIEW`) handelt.

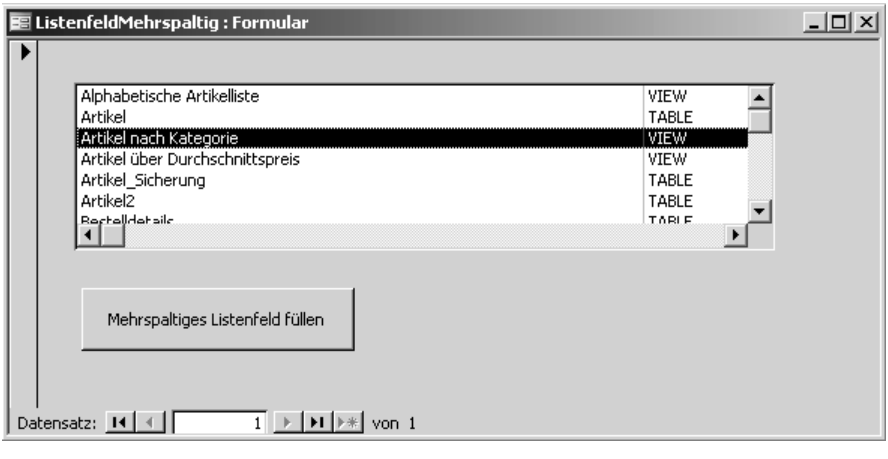


Abbildung 226: Das mehrspaltige Listenfeld wurde gefüllt.

308 Auf einzelne Spalten zugreifen

Wenn Sie das vorherige Beispiel noch um zwei Textfelder erweitern, dann können Sie das Beispiel ausbauen. Wenn ein Eintrag im Listenfeld angeklickt wird, dann sollen die einzelnen Spalteninhalte des Listenfelds in die dafür vorgesehenen Textfelder übertragen werden.

Um diese Aufgabe zu lösen, erfassen Sie das `Click`-Ereignis in Listing 372 für das Listenfeld.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_ListenfeldMehrspaltig
'=================================================

Private Sub Liste0_Click()
   'Spalteninhalte in Textfelder übertragen
   With Me
     .Text7.Value = .Liste0.Column(0, .Liste0.ListIndex)
     .Text9.Value = .Liste0.Column(1, .Liste0.ListIndex)
   End With
End Sub
```

Listing 372: Über die Eigenschaft Column einzelne Spalten auslesen

Sie können mit der Eigenschaft `Column` auf eine bestimmte Zeile oder eine Kombination aus Spalten und Zeilen in einem mehrspaltigen Kombinationsfeld oder einem Listenfeld verweisen. Die Syntax dieser Eigenschaft lautet:

```
Ausdruck.Column(Index, Row)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Ein Ausdruck, der unter anderem ein Listenfeld oder eine Kombinationsfeldliste zurückgibt.
`Index`	Erforderlicher Wert vom Typ Long. Ein Wert vom Typ Long Integer zwischen 0 und der Einstellung der `ColumnCount`-Eigenschaft minus 1.
`Row`	Optionaler Wert vom Typ Variant. Ein ganzzahliger Wert zwischen 0 und der Einstellung der `ListCount`-Eigenschaft minus 1.

Tabelle 120: Die Argumente der Methode RemoveItem

Über die Eigenschaft `ListIndex` haben Sie Zugriff auf den aktuell ausgewählten Eintrag im Listenfeld.

309 Kombinationsfeld füllen – Variante 1

In der folgenden Aufgabe wird ein Formular (KOMBINATIONSFELD) mit einem Kombinationsfeld aufgerufen. Im Kombinationsfeld befinden sich die Monate Januar bis Dezember. In einem Textfeld sollen nun jeweils der erste Tag sowie der letzte Tag des Monats angezeigt werden. Sehen Sie sich dazu die Ausgangssituation in Abbildung 228 an.

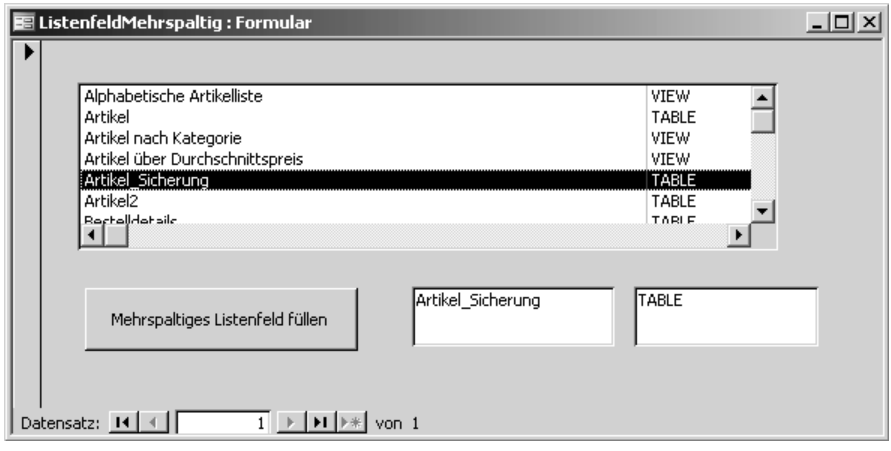


Abbildung 227: Die Spalteninhalte wurden in die Textboxen übertragen.

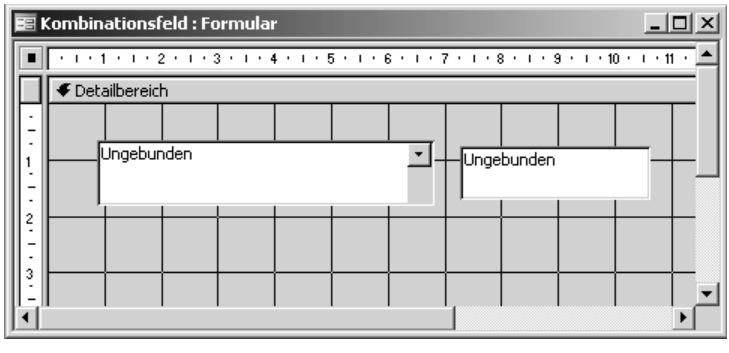


Abbildung 228: Die Monate müssen noch eingelesen werden.

Für das Füllen des Kombinationsfelds können Sie das Ereignis `Form_Load` aus Listing 373 nützen, das automatisch beim Öffnen des Formulars ausgeführt wird – der ideale Zeitpunkt, um die einzelnen Monate ins Kombinationsfeld zu bringen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld
'=====================================================

Private Sub Form_Load()
   With Me.Kombinationsfeld0
      .AddItem "Januar"
      .AddItem "Februar"
      .AddItem "März"
      .AddItem "April"
```

Listing 373: Ein Kombinationsfeld füllen – Variante 1

```
        .AddItem "Mai"
        .AddItem "Juni"
        .AddItem "Juli"
        .AddItem "August"
        .AddItem "September"
        .AddItem "Oktober"
        .AddItem "November"
        .AddItem "Dezember"
        .ListRows = 6
        .Value = MonthName(Month(Date))
    End With
End Sub
```

Listing 373: Ein Kombinationsfeld füllen – Variante 1 (Forts.)

Über die Methode `AddItem` füllen Sie die einzelnen Monate direkt in das Kombinationsfeld. Über die Eigenschaft `ListRows` können Sie festlegen, wie viele Einträge im Kombinationsfeld standardmäßig angezeigt werden sollen. Die restlichen Einträge können dann mit der vertikalen Leiste erreicht werden. Um den aktuellen Monat im Kombinationsfeld einzustellen, können Sie über die Funktion `Month` den aktuellen Monat (1–12) abfragen, indem Sie das aktuelle Datum über die Funktion `Date` dafür heranziehen. Da im Kombinationsfeld aber keine Zahlen zwischen 1 und 12, sondern eben der Monatsname in ausgeschriebener Form stehen soll, also Januar bis Dezember, übergeben Sie diese »Monatsnummer« an die Funktion `Monthname`, die daraus den ausgeschriebenen Monatsnamen macht.

Im zweiten Schritt füllen Sie das Textfeld, welches den ersten und letzten Tag im Monat ausgibt. Erfassen Sie dazu die Funktion aus Listing 374, die den letzten Tag eines Monats ermittelt, sowie das Ereignis `Change`, das automatisch ausgeführt wird, wenn Sie einen Eintrag im Kombinationsfeld auswählen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld
'=====================================================

'***In Standardmodul erfassen
Public Function LetzterTagImMonat(Edatum As Date) As Date
    LetzterTagImMonat = DateSerial(Year(Edatum), Month(Edatum) + 1, 0)
End Function


'***Direkt hinter dem Formular erfassen
Private Sub Kombinationsfeld0_Change()
    Dim strText As String
    Dim Edatum As Date
    Dim strMonat As String
```

Listing 374: Den letzten Tag im Monat ermitteln

```
    Dim strJahr As String

    strMonat = Me.Kombinationsfeld0.Value
    strJahr = Year(Date)
    Edatum = CDate(strMonat & " " & strJahr)
    Edatum = LetzterTagImMonat(Edatum)
    Text2.Value = "vom 01. " & strMonat & " " & _
       strJahr & " bis zum " & Format(Edatum, "DD.MMMM YYYY")
End Sub
```

Listing 374: Den letzten Tag im Monat ermitteln (Forts.)

Die Funktion `LetzterTagImMonat` erwartet als Übergabe einen Datumswert und liefert auch wieder einen Datumswert zurück. Das Datum bekommen Sie aus der markierten Zeile des Kombinationsfelds. Dieser Eintrag muss aber noch in ein gültiges Datumsformat umgewandelt werden. Dafür sorgt die Funktion `CDate`. Reagieren Sie nun auf eine Veränderung im Kombinationsfeld, indem Sie das Ereignis `Change` nützen.

Abbildung 229: Der letzte Tag im Monat wurde ermittelt.

Im folgenden Beispiel aus Listing 375 wird ein Kombinationsfeld im Formular `Kombinationsfeld2` mit den einzelnen Tagen des aktuellen Monats gefüllt. Auch diese Füllung wird bereits beim Öffnen des Formulars durchgeführt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld2
'=================================================

Private Sub Form_Load()
   Dim intTage As Integer

   For intTage = 1 To Day(LetzterTagImMonat(Date))
      With Me.Kombinationsfeld0
```

Listing 375: Die einzelnen Tage des aktuellen Monats im Kombinationsfeld anbieten

```
            .AddItem CDate((intTage) & " " & Month(Date) & " " & Year(Date))
        End With
    Next intTage
End Sub
```

Listing 375: Die einzelnen Tage des aktuellen Monats im Kombinationsfeld anbieten (Forts.)

Über eine Schleife lesen Sie die einzelnen Tage des aktuellen Monats in das Kombinationsfeld über die Methode `AddItem` ein. Dabei können Sie auch für diese Aufgabe die Funktion `LetzterTagImMonat` nützen, die Sie bereits im Makro aus Listing 374 angesprochen haben. Setzen Sie das Datum aus den Einzelteilen (Tages-, Monats- und Jahresanteil) zusammen und wandeln Sie diesen Text in ein gültiges Datum um, indem Sie die Funktion `CDate` einsetzen.

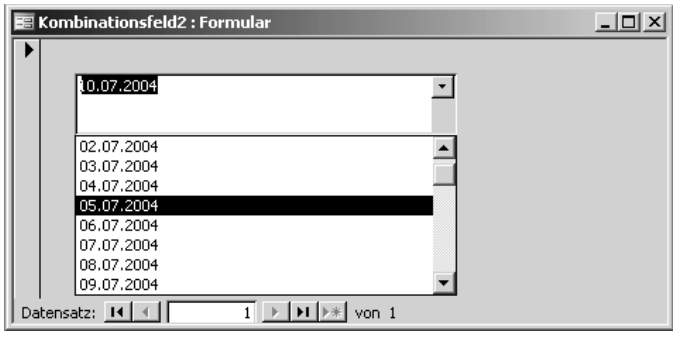


Abbildung 230: Die Tage des aktuellen Monats wurden eingefügt.

310 Kombinationsfeld füllen – Variante 2

Das folgende Beispiel aus Listing 376 zeigt eine weitere Variante, ein Kombinationsfeld zu füllen. Dazu wurde ein neues Formular mit dem Namen `Kombinationsfeld3` erstellt und danach ein Kombinationsfeld sowie eine Schaltfläche integriert. Über einen Klick auf die Schaltfläche wird das Kombinationsfeld mit Werten gefüllt.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld3
'======================================================

Private Sub Befehl2_Click()
    Dim Combo As ComboBox

    Set Combo = Me!Kombinationsfeld0

    Combo.RowSourceType = "Value List"
    Combo.RowSource = "Sehr gut;Gut;Befriedigend;Ausreichend;" _
```

Listing 376: Ein Kombinationsfeld füllen – Variante 2

```
      & "Mangelhaft;Ungenügend"
End Sub
```

Listing 376: Ein Kombinationsfeld füllen – Variante 2 (Forts.)

Setzen Sie die Eigenschaft `RowSourceTyp` ein, um dem Kombinationsfeld die Datenquelle mitzuteilen. Da es sich bei diesem Beispiel um ein ungebundenes Steuerelement handelt, geben Sie den String `Value List` ein. Damit wird festgelegt, dass Sie Einträge aus einer Liste dem Kombinationsfeld hinzufügen. Mithilfe der Eigenschaft `RowSource` geben Sie konkret an, was in das Kombinationsfeld eingefügt werden soll. Die einzelnen Einträge geben Sie nacheinander durch Semikola getrennt ein.

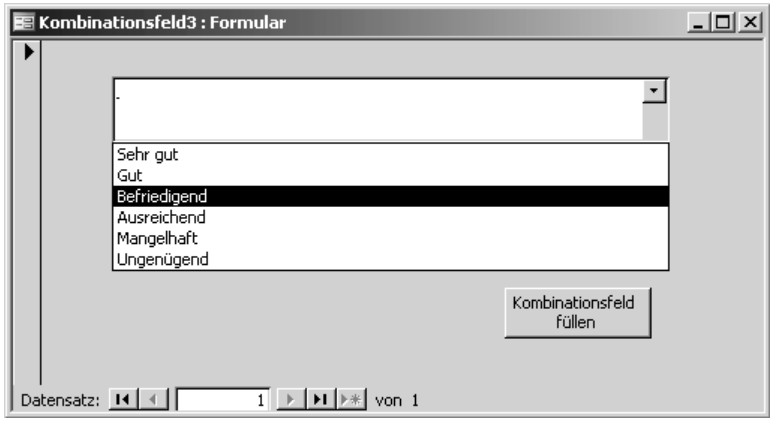


Abbildung 231: Die Alternative, ein Kombinationsfeld zu füllen

311 Editieren zulassen

Standardmäßig ist es bei Kombinationsfeldern so, dass Sie die Einträge definieren, die ausgewählt werden können. Von Fall zu Fall mag es aber auch einmal sinnvoll sein, dem Kombinationsfeld zusätzliche Einträge hinzuzufügen, die bisher noch nicht eingestellt sind.

Im nachfolgenden Beispiel aus Listing 377 wurden ein neues Formular mit dem Namen `Kombinationsfeld4` sowie ein Kombinationsfeld und ein Textfeld eingefügt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=================================================

Private Sub Form_Load()
   Dim Combo As ComboBox
```

Listing 377: Das Listenfeld wird beim Öffnen des Formulars gefüllt.

```
    Set Combo = Me!Kombinationsfeld5

    With Combo
      .RowSourceType = "Value List"
      .RowSource = "Spanien;Italien;Griechenland;Deutschland"
      .LimitToList = True
    End With
End Sub
```

Listing 377: Das Listenfeld wird beim Öffnen des Formulars gefüllt. (Forts.)

Das Ereignis `Form_Open` tritt automatisch beim Öffnen des Formulars ein. Ein idealer Zeitpunkt also, das Kombinationsfeld mit den Einträgen über die Eigenschaft `RowSource` zu füllen. Dabei stellen Sie zusätzlich bei der Eigenschaft `RowSourceTyp` den Text `Value List` ein. Über die Eigenschaft `LimitToList` legen Sie standardmäßig fest, dass keine zusätzlichen Einträge im Kombinationsfeld vorgenommen werden dürfen.

Über das Ereignis `NotInList` aus Listing 378 können Sie die gerade vorgenommene Einschränkung aber ein wenig aufweichen. Dabei haben Sie die Möglichkeit, die Standardfehlermeldung von Access durch eine Rückfrage zu ersetzen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=================================================

Private Sub Kombinationsfeld5_NotInList(NewData As String, Response As Integer)
   Dim ctl As Control

   Set ctl = Me!Kombinationsfeld5
   If MsgBox("Dieses Land gibt es bis jetzt noch nicht, Hinzufügen?", _
      vbOKCancel) = vbOK Then
      Response = acDataErrAdded
      ctl.RowSource = ctl.RowSource & ";" & NewData
   Else
      Response = acDataErrContinue
      ctl.Undo
   End If
End Sub
```

Listing 378: Auf die Eingabe neuer Einträge im Kombinationsfeld reagieren

Speichern Sie zunächst den gerade neu erfassten Eintrag in der Variablen `ctl`. Danach setzen Sie die Methode `MsgBox` ein, um eine Rückfrage einzuholen. Bestätigt der Anwender die Neueingabe, dann setzen Sie das Argument `Response` des Ereignisses auf den Wert `acDataErrAdded`. Damit wird die Neuanlage des Eintrags im Kombinationsfeld zugelassen. Danach werden über die Eigenschaft `RowSource` die bisherigen Einträge um den neuen Eintrag erweitert. Dieser steht automatisch im Argument `NewData` des Ereignisses `NotInList` bereit und kann so direkt übergeben werden.

Abbildung 232: Neuen Eintrag zulassen oder verwerfen?

Widerruft der Anwender seine Eingabe im Kombinationsfeld, dann weisen Sie dem Argument `Response` die Konstante `acDataErrContinue` zu. Dies bewirkt, dass Access die Eingabe widerruft und weiterhin nur Einträge zulässt, die bereits im Kombinationsfeld stehen.

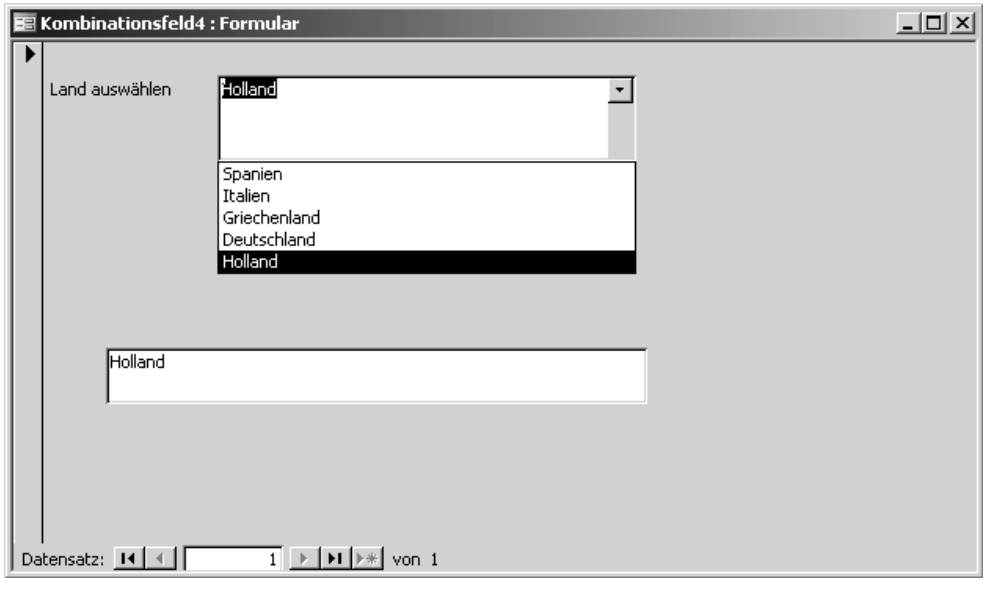


Abbildung 233: Der neue Eintrag lässt sich jetzt auch auswählen.

Damit der ausgewählte Eintrag in das darunter liegende Textfeld übertragen wird, setzen Sie das Ereignis `AfterUpdate` aus Listing 379 ein.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=================================================

Private Sub Kombinationsfeld5_AfterUpdate()
   Text3.Value = Kombinationsfeld5.Value
End Sub
```

Listing 379: Kombinationsfeld und Textfeld verknüpfen

312 Kontrollkästchen

Setzen Sie Kontrollkästchen in Formularen ein, um mehrere Optionen zuzulassen. Ein Kontrollkästchen kann entweder aktiviert oder nicht aktiviert sein. Bei aktiviertem Zustand erscheint im Kästchen ein Häkchen. Wenn Sie Kontrollkästchen in einer Gruppe verwenden, können sowohl eines als auch mehrere Kontrollkästchen aktiviert sein.

Wenn Sie die Kontrollkästchen in einer Optionsgruppe positionieren, dann kann jeweils nur ein Kontrollkästchen aktiviert werden. Stehen diese außerhalb einer Optionsgruppe, dann können standardmäßig alle Kontrollkästchen aktiviert werden. Möchten Sie trotzdem nicht auf die optischen Vorteile einer Optionsgruppe verzichten, ziehen Sie über die Toolbox ein Rechteck auf und fügen die Kontrollkästchen danach ein. Jetzt können Sie alle Kontrollkästchen aktivieren und trotzdem kommt die Optik nicht zu kurz.

313 Kontrollkästchen auswerten

Im folgenden Makro aus Listing 380 wurde ein Formular mit drei Kontrollkästchen und einer Schaltfläche eingefügt. Damit haben Sie die Möglichkeit, bis zu drei Standardabfragen hintereinander zu starten.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kontrollkästchen
'=================================================

Private Sub Befehl8_Click()
   Const Abfr01 = "Alphabetische Artikelliste"
   Const Abfr02 = "Artikel nach Kategorie"
   Const Abfr03 = "Die zehn teuersten Artikel"
```

Listing 380: Standardabfragen per Klick starten

```
    If Me!Kontrollkästchen1.Value = -1 Then DoCmd.OpenQuery Abfr01, acViewNormal
    If Me!Kontrollkästchen2.Value = -1 Then DoCmd.OpenQuery Abfr02, acViewNormal
    If Me!Kontrollkästchen3.Value = -1 Then DoCmd.OpenQuery Abfr03, acViewNormal
End Sub
```

Listing 380: Standardabfragen per Klick starten (Forts.)

Definieren Sie zu Beginn des Makros die Namen der Abfragen, die Sie starten möchten. Selbstverständlich hätten Sie auch den Text bei den Kontrollkästchen dazu benutzen können, um die einzelnen Abfragen danach zu starten. In diesem Fall müssten die Beschriftungen aber mit den Namen der Abfragen übereinstimmen. Mithilfe von Konstanten haben Sie aber die Möglichkeit, die Beschriftungsfelder bei den Kontrollkästchen flexibel zu halten.

Wenn Sie nicht sicher sind, welche Namen Ihre Kontrollkästchen beim Einfügen ins Formular bekommen haben, bzw. wenn Sie andere Namen vergeben möchten, wechseln Sie in die Entwurfsansicht Ihres Formulars, klicken das betreffende Kontrollkästchen an und sehen auf der Registerkarte ANDERE im Feld NAME nach. Dort können Sie einen neuen Namen zuweisen und mit der Taste ↹ bestätigen.

Weisen die Kontrollkästchen den Wert –1 auf, so sind sie aktiviert. Starten Sie danach die entsprechende Abfrage mithilfe der Methode `OpenQuery`.

Abbildung 234: Kontrollkästchen auswerten

314 Kontrollkästchen initialisieren

Wenn Sie das Formular `Kontrollkästchen` mit einem Doppelklick starten, dann sind die Kontrollkästchen alle zu Beginn mit einem grauen Farbton ausgefüllt, d.h., sie wurden nicht initialisiert. Dies können Sie jetzt nachholen, indem Sie das Ereignis `Form_Load` wie in Listing 381 einstellen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kontrollkästchen
'=================================================

Private Sub Form_Load()
   Dim frm As Form
   Dim ctrl As Object

   For Each ctrl In Me.Controls
      If ctrl.ControlType = acCheckBox Then ctrl.Value = 0
   Next ctrl
End Sub
```

Listing 381: Kontrollkästchen initialisieren

Indem Sie die Eigenschaft `Value` auf den Wert 0 setzen, werden die Kontrollkästchen initialisiert.

315 Optionsfelder

Die Optionsschaltflächen in Access haben dieselben Eigenschaften wie die Kontrollkästchen. Als eine Eigenart von Access im Vergleich zu den anderen Office-Programmen können Sie Optionsschaltflächen dazu einsetzen, gleich mehrere Optionen auf einmal zu aktivieren. Möchten Sie hingegen die ursprüngliche Funktion einstellen, dass Sie jeweils nur eine Option auswählen können, müssen Sie die einzelnen Optionsschaltflächen in eine Optionsgruppe einfügen.

Im folgenden Beispiel soll eine dynamische Dateisuche programmiert werden. Dazu legen Sie ein neues Formular mit einem Gruppenfeld und vier Optionsfeldern sowie einer Schaltfläche an. Orientieren Sie sich dabei an Abbildung 235.

Legen Sie nun das Makro aus Listing 382 hinter die SUCHEN-Schaltfläche.

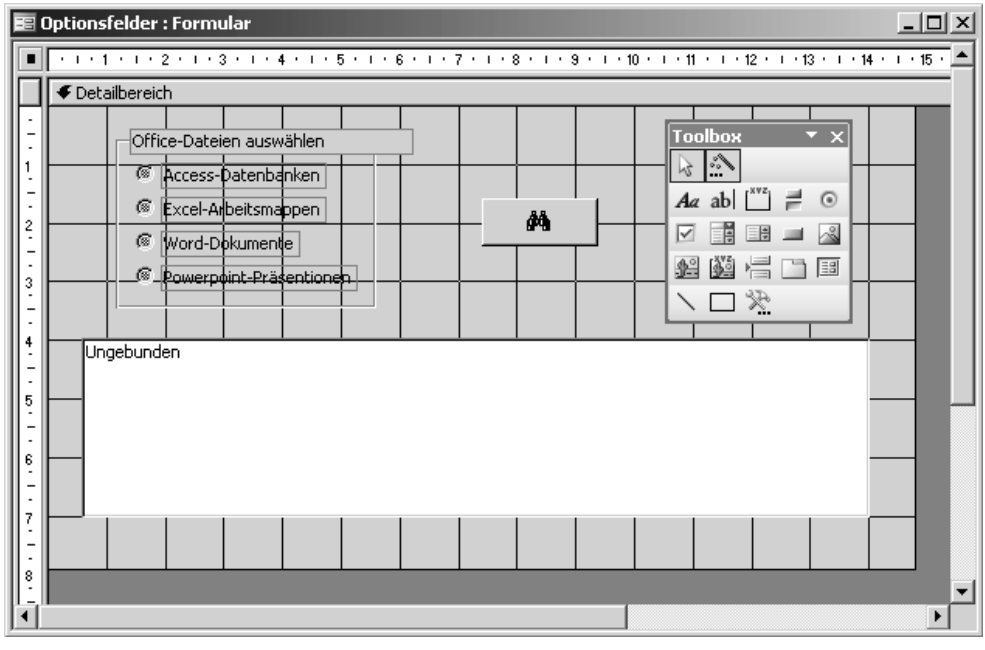


Abbildung 235: Die Optionsfelder werden in einem Gruppenfeld erstellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Optionsfelder
'=====================================================

Private Sub Befehl13_Click()
   'Office-Dateien suchen
   Dim varDatei As Variant
   Dim intZ As Integer

   For intZ = Me!Liste11.ListCount - 1 To 0 Step -1
      Me!Liste11.RemoveItem (intZ)
   Next intZ

   With Application.FileSearch
      .NewSearch
      .LookIn = "c:\Eigene Dateien\"
      Select Case Me.Rahmen0
         Case 1
            .FileType = msoFileTypeDatabases
         Case 2
            .FileType = msoFileTypeExcelWorkbooks
         Case 3
            .FileType = msoFileTypeWordDocuments
```

Listing 382: Office-Dateien optional suchen

```
         Case 4
            .FileType = msoFileTypePowerPointPresentations
         Case Else
            .FileType = msoFileTypeAllFiles
      End Select
      .SearchSubFolders = True

      If .Execute() > 0 Then
         For Each varDatei In .FoundFiles
            Me!Liste11.AddItem varDatei
         Next varDatei
      End If
      Me.Text15.Value = .FoundFiles.Count & " Dateien gefunden!"
   End With
End Sub
```

Listing 382: Office-Dateien optional suchen (Forts.)

Im ersten Schritt entfernen Sie zunächst alle eventuell schon angezeigten Einträge im Listenfeld. Dies bietet den Vorteil, dass Sie die Suche mehrfach ausführen können und dabei jeweils zu Beginn ein geleertes Listenfeld vorliegen haben.

Mithilfe des Objekts `FileSearch` kann nach bestimmten Office-Dateien in Verzeichnissen gesucht werden. Über die Eigenschaft `FileType` können Sie den zu suchenden Dateityp näher bestimmen. Im Anhang finden Sie eine komplette Auflistung dieser Konstanten.

Über die Eigenschaft `LookIn` kann festgelegt werden, wo Access nach den Dateien suchen soll. Hier können Sie das Laufwerk sowie das Verzeichnis angeben. Sollen noch darunter liegende Verzeichnisse durchsucht werden, dann kann dies über die Eigenschaft `SearchSubFolders` festgelegt werden. Diese Eigenschaft wird auf den Wert `True` gesetzt, wenn die angegebene Suche alle Unterordner im durch die `LookIn`-Eigenschaft angegebenen Ordner einschließen soll. Die Methode `Execute` führt die jetzt näher spezifizierte Suche anschließend aus.

Nach der Suche sind alle gefundenen Dateien im Objekt `FoundFiles` verzeichnet. Diese gefundenen Dateien werden in einem Textfeld ausgeben.

316 Schaltflächen

Um die Programmierung Ihrer Schaltflächen brauchen Sie sich in Access keine Sorgen zu machen. Der Befehlsschaltflächen-Assistent hat eine Menge an fertigen Codes, die Sie für Ihre Formulare einsetzen können.

Um beispielsweise ein Makro für die Speicherung von Daten über ein Formular in eine Tabelle zu erstellen, verfahren Sie wie folgt:

1. Öffnen Sie das Formular ARTIKEL der Datenbank FORMULAR.MDB in der Entwurfsansicht.
2. Klicken Sie in der Toolbox auf das Symbol BEFEHLSSCHALTFLÄCHE und ziehen Sie es auf Ihrem Formular auf.
3. Markieren Sie unter KATEGORIEN den Eintrag DATENSATZOPERATIONEN.

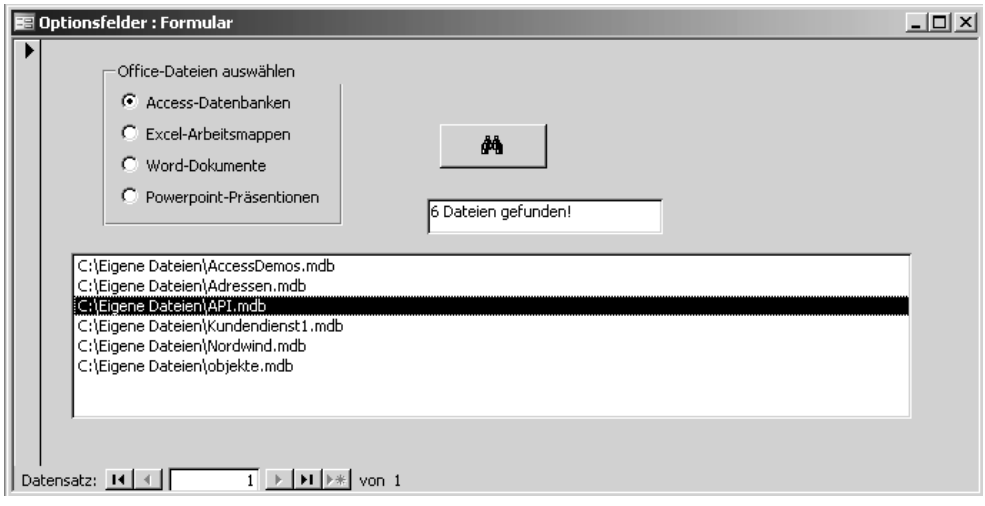


Abbildung 236: Es wurden nur Access-Datenbanken gesucht.

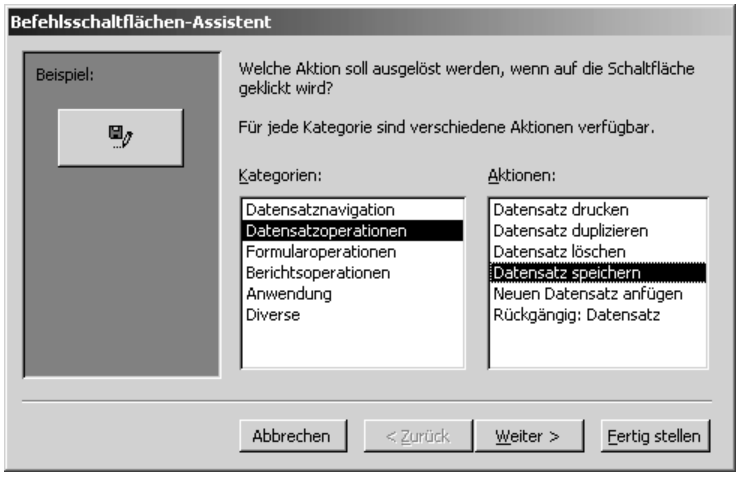


Abbildung 237: Schaltfläche einfügen

4. Wählen Sie im Feld AKTIONEN den Eintrag DATENSATZ SPEICHERN.
5. Klicken Sie auf die Schaltfläche WEITER.
6. Entscheiden Sie sich für das Layout der Schaltfläche, indem Sie entweder nur Text anzeigen oder ein Symbol auf der Schaltfläche anzeigen lassen.
7. Klicken Sie auf WEITER.
8. Weisen Sie im nächsten Dialogfeld der Schaltfläche einen Namen zu und klicken Sie auf FERTIG STELLEN, um die Aufzeichnung des Makros zu beenden.

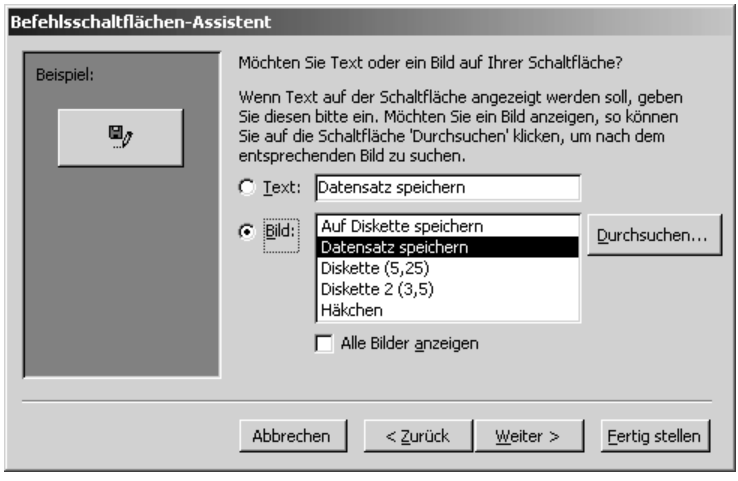


Abbildung 238: Symbol auswählen

Die Schaltfläche wurde nun in Ihr Formular eingefügt. Klicken Sie mit der rechten Maustaste auf die neu eingefügte Schaltfläche und wählen Sie aus dem Kontextmenü den Befehl EREIGNIS. Sie gelangen daraufhin automatisch in die Entwicklungsumgebung von Access und haben dann folgendes Makro aus Listing 383 vor Augen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Befehl43_Click()
   On Error GoTo Err_Befehl43_Click
   DoCmd.DoMenuItem acFormBar, acRecordsMenu, acSaveRecord, , acMenuVer70

   Exit_Befehl43_Click:
      Exit Sub

   Err_Befehl43_Click:
      MsgBox Err.Description
      Resume Exit_Befehl43_Click
End Sub
```

Listing 383: Datensatz speichern über DoMenuItem

Dabei wird die Methode `DoMenuItem` eingesetzt. Diese Methode führt den angegebenen Menübefehl aus. Diese Methode hat folgende Syntax:

```
DoMenuItem(Menüleiste, Menüname, Befehl, Unterbefehl, Version)
```

Dem Argument Menüleiste weisen Sie die Konstante acFormBar zu. Damit sprechen Sie die komplette Menüleiste von Access an, die automatisch dann eingeblendet wird, wenn Sie ein Formular aufrufen.

Im Argument Menüname geben Sie an, um welches Menü in der Menüleiste es sich handeln soll. Dazu gibt Ihnen Access drei Konstanten vor:

- acFile: das Menü DATEI.
- acEditMenu: das Menü BEARBEITEN.
- AcRecordsMenu: das Menü DATENSÄTZE.

Im Argument Befehl geben Sie den Befehl ein, der im Argument Menüname zu finden ist und den Sie ausführen möchten.

Befehl	Menü	Befehl
acNew	EINFÜGEN	NEUER DATENSATZ
acSaveForm	DATEI	SPEICHERN
acSaveFormAs	DATEI	SPEICHERN UNTER
acSaveRecord	DATENSÄTZE	DATENSATZ SPEICHERN
acUndo	BEARBEITEN	RÜCKGÄNGIG
acCut	BEARBEITEN	AUSSCHNEIDEN
acCopy	BEARBEITEN	KOPIEREN
acPaste	BEARBEITEN	EINFÜGEN
acDelete	BEARBEITEN	DATENSATZ LÖSCHEN
acSelectRecord	BEARBEITEN	DATENSATZ AUSWÄHLEN
acSelectAllRecords	BEARBEITEN	ALLE DATENSÄTZE AUSWÄHLEN
acObject	EINFÜGEN	OBJEKT
acRefresh	DATENSÄTZE	AKTUALISIEREN

Tabelle 121: Die Befehlskonstanten der Methode DoMenuItem

Das Argument Unterbefehl wird nur dann verwendet, wenn Sie den Befehl OBJEKT aus dem Menü EINFÜGEN einsetzen. Die dafür zur Verfügung stehenden Konstanten lauten acObject-Verb und acObjectUpdate.

Das Argument Version müssen Sie nur einsetzen, wenn Sie eine ältere Access-Version im Einsatz haben. Setzen Sie die Konstante acMenuVer70 für Code ein, der für Microsoft Access 95-Datenbanken geschrieben wurde, die eingebaute Konstante acMenuVer20 kommt dann zur Anwendung, wenn Code für Access-Datenbanken der Version 2.0 eingegeben wurde.

Die Methode DoMenuItem wird zwar verwendet, wenn Sie den Schaltflächen-Assistenten verwenden, um Schaltflächencode zu generieren. Diese Vorgehensweise können Sie jedoch auch beibehalten, um den Coderahmen in der Entwicklungsumgebung automatisch erzeugen zu

lassen. Danach können Sie die Methode `DoMenuItem` aber durch die aktuellere und schnellere Methode `RunCommand` austauschen. Das sieht dann wie folgt aus:

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=================================================

Private Sub Befehl43_Click()
   On Error GoTo Err_Befehl43_Click
   DoCmd.RunCommand (acCmdSaveRecord)

   Exit_Befehl43_Click:
      Exit Sub

   Err_Befehl43_Click:
      MsgBox Err.Description
      Resume Exit_Befehl43_Click
End Sub
```

Listing 384: Datensatz speichern über RunCommand

Die Methode `RunCommand` hat nur noch ein Argument, welches Sie komplett in der Online-Hilfe zu Access-VBA bzw. die wichtigsten im Anhang nachlesen können. Für das Speichern eines Datensatzes ist das in diesem Beispiel die Konstante `acCmdSaveRecord`.

317 Eine blinkende Schaltfläche erstellen

Möchten Sie auf einem Formular eine blinkende Schaltfläche unterbringen, fügen Sie diese Schaltfläche zuerst in Ihr Formular `Personal` ein und befolgen danach die nächsten Arbeitsschritte:

1. Wechseln Sie in die Entwurfsansicht des Formulars.
2. Rufen Sie das Eigenschaftenfenster des Formulars auf.
3. Stellen Sie im Kombinationsfeld den Eintrag FORMULAR ein.
4. Tragen Sie im Feld ZEITGEBERINTERVALL den Wert 250 ein. Die Einheit ist hier Millisekunden, d.h., die Schaltfläche würde somit viermal in der Sekunde blinken.
5. Erstellen Sie jetzt das folgende Ereignis aus Listing 385.

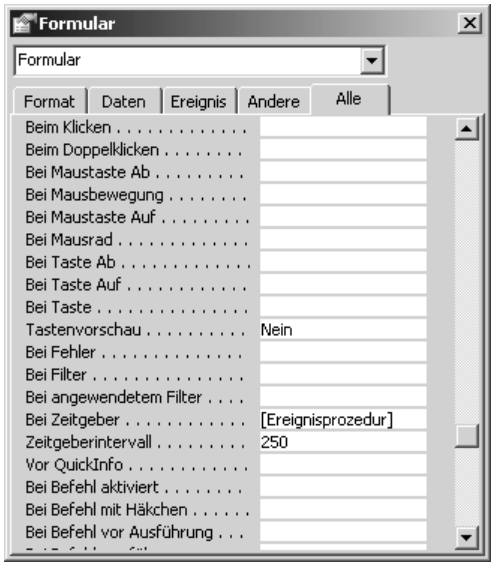


Abbildung 239: Das Zeitgeberintervall einstellen

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Personal
'=================================================

Private Sub Form_Timer()
   If Me!Befehl43.Visible = True Then
      Me!Befehl43.Visible = False
      Exit Sub
   End If
   If Me!Befehl43.Visible = False Then
      Me!Befehl43.Visible = True
   End If
End Sub
```

Listing 385: Eine blinkende Schaltfläche erstellen

Da Sie vorher das Intervall eingestellt haben, sorgen Sie im Ereignismakro lediglich dafür, dass Sie die Schaltfläche im Wechsel ein- und ausblenden. Fügen Sie nach dem Ausblenden der Schaltfläche die Anweisung `Exit Sub` aus, so dass die Schaltfläche nicht direkt danach wieder eingeblendet wird. Die Schaltfläche wird erst dann wieder eingeblendet, wenn das Ereignis das nächste Mal wieder ausgeführt wird, also genau nach 250 Millisekunden.

318 Kalendersteuerelement

Standardmäßig ist das Kalendersteuerelement nicht direkt in der Toolbox aufgeführt. Sie können dieses Steuerelement aber trotzdem hinzufügen. Legen Sie dazu ein neues Formular an und klicken Sie in der Toolbox auf das Symbol WEITERE STEUERELEMENTE. Daraufhin klappt ein Menü auf, in dem Sie das Steuerelement KALENDERSTEUERELEMENT 10.0 für Access 2002 bzw. das Steuerelement KALENDERSTEUERELEMENT 11.0 für Access 2003 auswählen können. Ziehen Sie dieses Element auf dem Formular KALENDER in der gewünschten Größe auf.

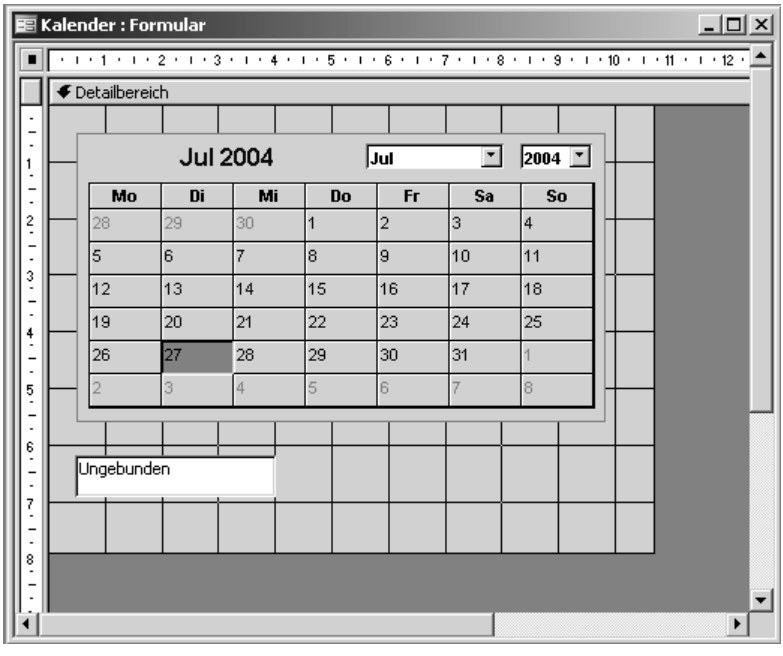


Abbildung 240: Der eingefügte Kalender

Diesen Kalender können Sie übrigens ganz individuell einstellen. Dabei können Sie die Darstellung sowie die verwendeten Farben und die Schriftart nach Ihren Wünschen anpassen. Klicken Sie dazu in der Entwurfsansicht des Formulars mit der rechten Maustaste auf das Kalendersteuerelement und wählen Sie den Befehl KALENDER-OBJEKT/EIGENSCHAFTEN.

In der Tabelle 122 werden die wichtigsten Eigenschaften und Methoden des Kalendersteuerelements noch einmal zusammengefasst.

Eigenschaft (E)/ Methode (M)	Beschreibung
`Value` (E)	Gibt den Wert aus, der momentan im Steuerelement aktiviert ist, also beispielsweise ein einzelner Tag.

Tabelle 122: Die wichtigsten Methoden und Eigenschaften für den Kalender

Eigenschaft (E)/ Methode (M)	Beschreibung
`Day` (E)	Der aktuelle Monatstag im Kalendersteuerelement kann eingestellt oder abgefragt werden.
`Month` (E)	Über diese Eigenschaft können Sie angeben, welcher Monat im Kalendersteuerelement angezeigt werden soll.
`Year` (E)	Mit dieser Eigenschaft können Sie bestimmen, welches Jahr im Kalendersteuerelement angezeigt wird. Dabei können Jahreszahlen zwischen 1900 und 2100 angegeben werden.
`ShowDateSelectors` (E)	Mit dieser Eigenschaft legen Sie fest, ob die Drop-down-Listenfelder für Monat und Jahr im Kalendersteuerelement angezeigt werden.
`DayLength` (E)	Mit dieser Eigenschaft bestimmen Sie, in welchem Format (kurz, mittel oder lang) die Wochentage im Kalendersteuerelement angezeigt werden. Die kurze Ausgabeform M, D, M wird erreicht, indem diese Eigenschaft auf den Wert 0 gesetzt wird. Setzen Sie diese Eigenschaft auf den Wert 1, wenn die Tagesnamen Mo, Di, Mi angezeigt werden sollen. Die Langform Montag, Dienstag, Mittwoch wird erreicht, indem man der Eigenschaft den Wert 2 zuweist.
`MonthLength` (E)	Mit dieser Eigenschaft können Sie bestimmen, ob die Monate im Monat/Jahr-Titel im Kalendersteuerelement im kurzen oder langen Format angezeigt werden. Wenn Sie diese Eigenschaft auf den Wert 0 setzen, dann werden die Monate in Kurzschreibweise Jan, Feb, Mär angezeigt. Wird diese Eigenschaft auf den Wert 2 gesetzt, erfolgt die Formatierung der Monate im Langformat Januar, Februar.
`FirstDay` (E)	Mit dieser Eigenschaft können Sie bestimmen, welcher Wochentag in der ersten Spalte des Kalendersteuerelements angezeigt wird. Dabei beginnt der Sonntag mit der Ziffer 1 und der Samstag endet mit der 6. Um also den Montag als ersten Tag der Woche im Kalender anzuzeigen, setzen Sie diese Eigenschaft auf den Wert 2.
`NextDay` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um einen Tag erhöhen und das Kalendersteuerelement aktualisieren.
`NextWeek` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um eine Woche erhöhen und das Kalendersteuerelement aktualisieren.
`NextMonth` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um einen Monat erhöhen und das Kalendersteuerelement aktualisieren.
`NextYear` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um ein Jahr erhöhen und das Kalendersteuerelement aktualisieren.

Tabelle 122: Die wichtigsten Methoden und Eigenschaften für den Kalender (Forts.)

Eigenschaft (E)/ Methode (M)	Beschreibung
`PreviousDay` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um einen Tag vermindern und das Kalendersteuerelement aktualisieren.
`PreviousWeek` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um eine Woche vermindern und das Kalendersteuerelement aktualisieren.
`PreviousMonth` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um einen Monat vermindern und das Kalendersteuerelement aktualisieren.
`PreviousYear` (M)	Mit dieser Methode können Sie den Wert eines Kalendersteuerelements um ein Jahr vermindern und das Kalendersteuerelement aktualisieren.
`Today` (M)	Mit dieser Methode können Sie den Wert des Kalendersteuerelements auf das heutige Datum einstellen.

Tabelle 122: Die wichtigsten Methoden und Eigenschaften für den Kalender (Forts.)

Um ein Textfeld mit dem Kalender zu verknüpfen, stellen Sie das `Click`-Ereignis des Kalenders ein. Immer wenn ein Datum im Kalender ausgewählt wird, soll das ausgewählte Datum im Textfeld angezeigt werden. Sehen Sie sich dazu das Listing 386 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kalender
'=====================================================

Private Sub Calendar0_Click()
   'Datum aus Kalender ins Textfeld übertragen
   With Me
      .Text1.Value = .Calendar0.Value
   End With
End Sub
```

Listing 386: Datum per Klick übernehmen

Im folgenden Beispiel aus Listing 387 wird ein neues Formular (`Kalender2`) angelegt und ein zusätzliches Listenfeld integriert. Per Doppelklick auf einzelne Tage soll dabei das angeklickte Datum in das Listenfeld eingefügt werden. Dabei sollen keine doppelten Tage ins Listenfeld übertragen werden.

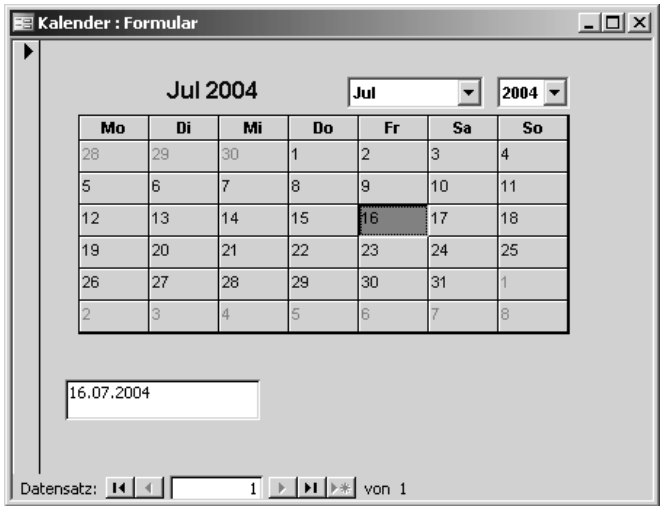


Abbildung 241: Das ausgewählte Datum wird ins Textfeld übernommen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Kalender2
'=====================================================

Private Sub Calendar0_DblClick()
   'Tage ins Listenfeld übertragen (inkl. Prüfung)
   Dim b As Boolean

   b = False
   With Me
      For intZ = 0 To .Liste1.ListCount - 1
         If .Liste1.ItemData(intZ) = Str(.Calendar0.Value) Then
            b = True
         End If
      Next intZ
      If b = False Then .Liste1.AddItem .Calendar0.Value
      b = False
   End With
End Sub
```

Listing 387: Tage nach erfolgtem Dubletten-Check ins Listenfeld übertragen

Bevor Sie die einzelnen Tage ins Listenfeld übertragen, führen Sie eine Prüfung durch, ob das Datum bereits im Listenfeld vorhanden ist. Wenn ja, dann setzen Sie den Schalter `b` auf den Wert `True`. Beim Vergleich der beiden Datumsangaben müssen Sie darauf achten, dass Sie die Datumsangabe des Kalenders in einen String über die Funktion `Str` umwandeln. Damit vergleichen Sie dann den Datums-String aus dem Listenfeld mit einem Datums-String aus dem Kalender.

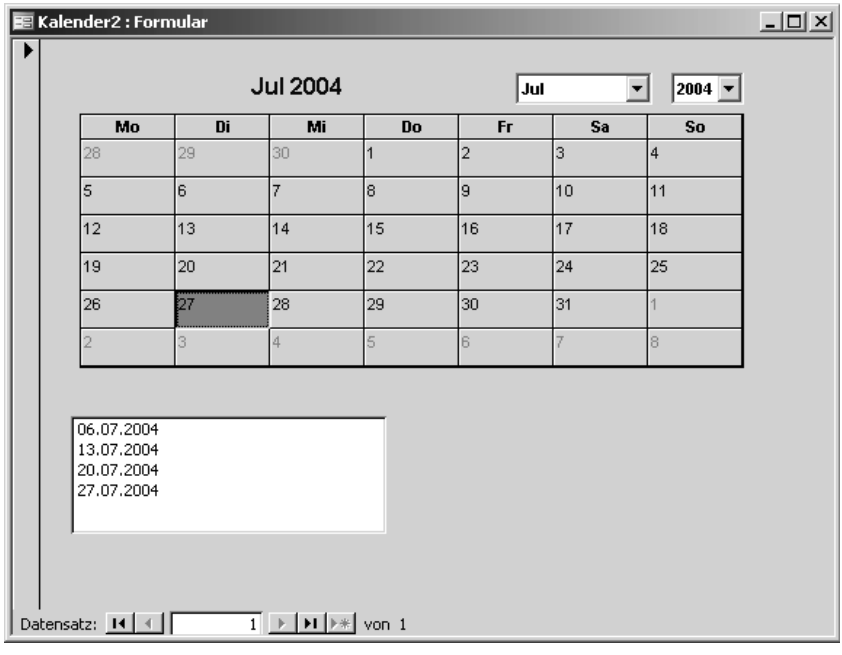


Abbildung 242: Es werden keine doppelten Tage übertragen.

319 Der Fortschrittsbalken

Werden über ein Formular längere Makros abgearbeitet, sollten Sie sich überlegen, ob Sie nicht einen Fortschrittsbalken anbieten möchten. Bei länger laufenden Makros ist es empfehlenswert, den Anwender während des Makroablaufs mit gelegentlichen Meldungen in der Statusleiste oder sogar einem Verlaufsbalken über die einzelnen Schritte bzw. den Verlauf des Makros in Kenntnis zu setzen.

Das Steuerelement MICROSOFT PROGRESSBAR CONTROL können Sie aktivieren, indem Sie das Symbol WEITERE STEUERELEMENTE in der Toolbox anklicken und das Steuerelement aus der Liste auswählen. Fügen Sie dieses Steuerelement und eine Schaltfläche in einem neuen Formular (FORTSCHRITTSBALKEN) ein. Legen Sie das Makro aus Listing 388 hinter die Schaltfläche.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Fortschrittbalken
'=================================================

Private Sub Befehl1_Click()
   'Fortschrittsbalken programmieren
```

Listing 388: Den Fortschrittsbalken animieren

```
    Dim intz As Integer

    With Me
       .ProgressBar0.Min = 0
       .ProgressBar0.Max = 10000

       For intz = ProgressBar0.Min To .ProgressBar0.Max
          .ProgressBar0.Value = intz
          intz = intz + 1
       Next intz
       .ProgressBar0.Value = 0
    End With
 End Sub
```

Listing 388: Den Fortschrittsbalken animieren (Forts.)

Mithilfe der Eigenschaften `Min` bzw. `Max` können Sie den Start- bzw. Endwert des Steuerelements bestimmen. Mit diesen beiden Informationen können Sie sich dann eine Schleife zusammenstellen, die so lange durchlaufen wird, bis der `Max`-Wert erreicht ist. Läuft Ihnen der Fortschrittsbalken zu schnell, dann erhöhen Sie den `Max`-Wert einfach. Den Fortschrittsbalken bekommen Sie automatisch zum Laufen, wenn Sie die Eigenschaft `Value` des Fortschrittbalkens füttern.

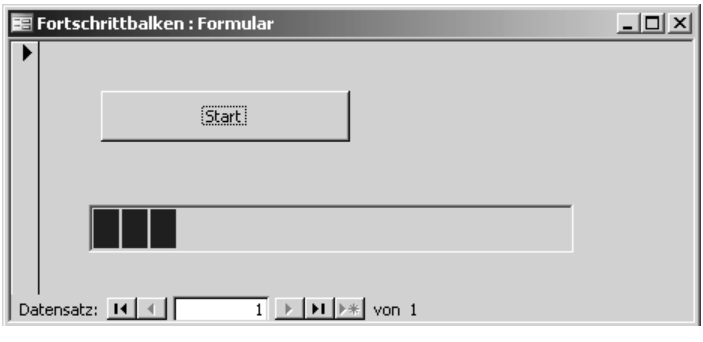


Abbildung 243: Das Steuerelement ProgressBar im Einsatz

Die Art des Fortschrittsbalkens sowie dessen Aussehen können Sie übrigens anpassen. Klicken Sie dazu in der Entwurfsansicht des Formulars mit der rechten Maustaste auf das Steuerelement und wählen Sie den Befehl PROGCTRL-OBJEKT/PROPERTIES aus dem Kontextmenü.

320 Der Webbrowser

Über den Einsatz des Steuerelements WEBBROWSER können Sie Internetseiten oder auch Office-Dateien öffnen. Das Steuerelement MICROSOFT WEBBROWSER aktivieren Sie, indem Sie das Symbol WEITERE STEUERELEMENTE in der Toolbox anklicken und das Steuerelement aus der Liste auswählen. Fügen Sie dieses Steuerelement und eine Schaltfläche in einem

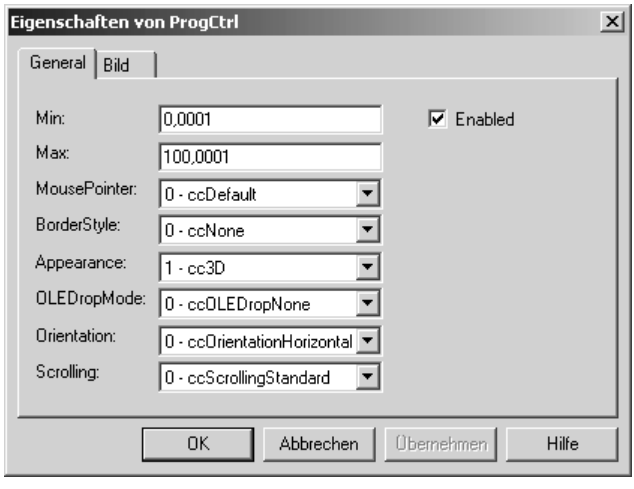


Abbildung 244: Der Fortschrittsbalken lässt sich einstellen.

neuen Formular (WEBBROWSER) ein. Legen Sie das Makro aus Listing 389 hinter die Schaltfläche.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Webbrowser
'=====================================================

Private Sub Befehl3_Click()
   'Webseite oder Office-Datei aufrufen
   With Me
      .WebBrowser0.Navigate URL:=Text1.Value
   End With
End Sub
```

Listing 389: Webseiten, Office-Dateien oder Bilder über den Webbrowser betrachten

Mithilfe der Methode `Navigate` können Sie die im Argument `URL` angegebene Webseite oder Datei im Webbrowser laden.

321 Zugriff auf Outlook-Postfach

Über das Steuerelement MICROSOFT OFFICE OUTLOOK VIEW CONTROL können Sie in einem Formular Ihr E-Mail-Postfach abrufen. Fügen Sie dieses Steuerelement in ein neues Formular (OUTLOOKPOST) ein und integrieren Sie eine zusätzliche Schaltfläche. Legen Sie danach das Makro aus Listing 390 hinter die Schaltfläche.

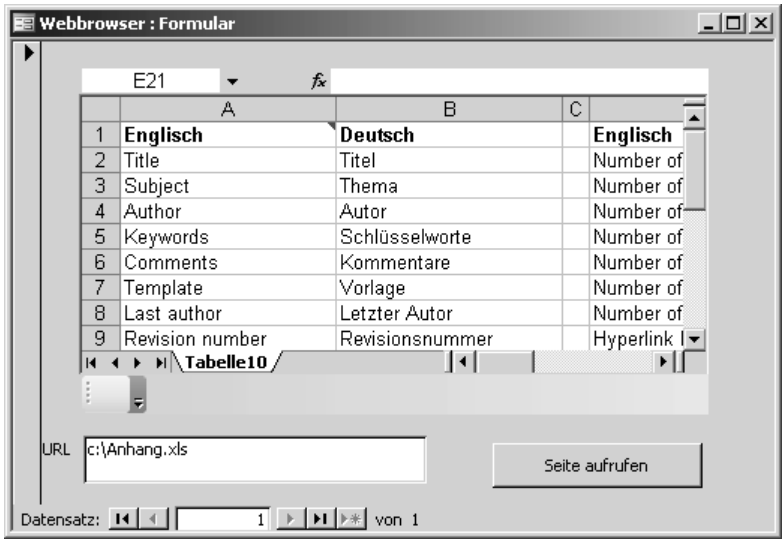


Abbildung 245: Eine Excel-Arbeitsmappe über den Webbrowser öffnen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_OutlookPost
'=====================================================

Private Sub Befehl1_Click()
   'E-Mail öffnen
   With Me
      .ViewCtl0.Open
   End With
End Sub
```

Listing 390: E-Mail per Klick öffnen

Die Mails werden beim Öffnen des Formulars automatisch in das Steuerelement geladen. Sie haben dabei die Möglichkeit, die Mails über einen Klick auf die entsprechende Rubrik zu sortieren.

Über die Methode `Open` wird die markierte Mail im Listenfeld geöffnet.

322 Das TreeView-Steuerelement

Wie der Name schon sagt, können Sie mithilfe dieses Steuerelements ganze Verzeichnisbäume auf einem Formular abbilden. Machen Sie das TREEVIEW-Steuerelement auf Ihrer Werkzeugsammlung verfügbar und fügen Sie es in einem neuen Formular (TREEVIEW) ein. Integrieren Sie zusätzlich noch eine weitere Schaltfläche.

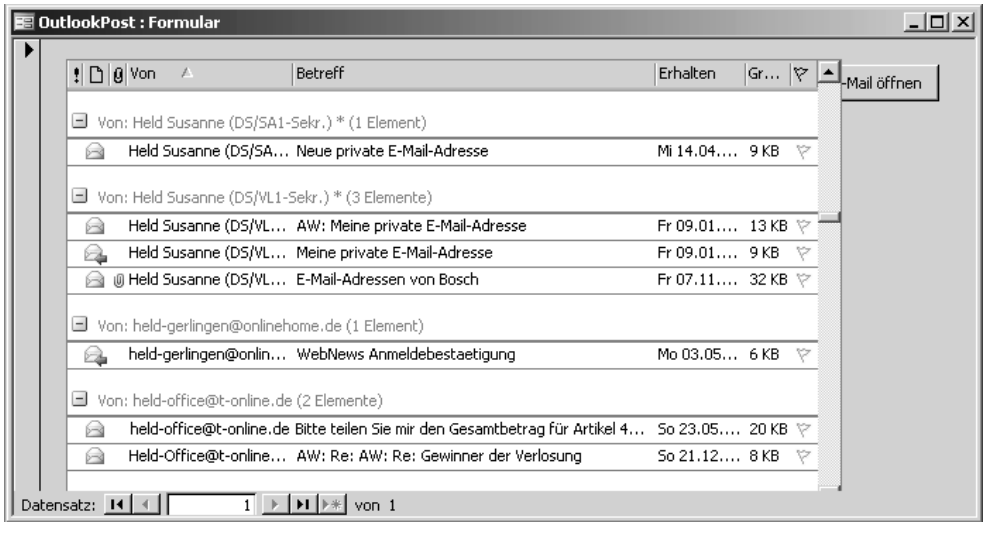


Abbildung 246: Alle Mails werden angezeigt.

Da das TREEVIEW-Steuerelement schon beim Aufruf des Formulars gefüllt sein muss, setzen Sie zu diesem Zweck auch hier das Ereignis `Form_Load` ein, welches Sie im folgenden Listing sehen können. Dabei sollen alle Module der aktiven Datenbank im TREEVIEW-Steuerelement angezeigt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_TreeView
'=====================================================

Private Sub Form_Load()
   Dim Eintrag As Node
   Dim obj As AccessObject
   Dim dbs As Object

   Set dbs = Application.CurrentProject
   Set Eintrag = TreeView0.Nodes.Add(, , , dbs.Name)

   For Each obj In dbs.AllModules
      TreeView0.Nodes.Add Eintrag.Index, tvwChild, , obj.Name
   Next obj
End Sub
```

Listing 391: Alle Module im TreeView-Steuerelement anzeigen

Ermitteln Sie zuerst den Namen der aktiven Datenbank. Fügen Sie diesen Namen über die Methode `Add` im Steuerelement TREEVIEW als neuen Zweig ein. Unterhalb dieses Zweigs

ermitteln Sie die Namen der Module, die in der Datenbank enthalten sind. Fügen Sie diese Modulnamen unterhalb des Hauptzweigs wiederum über die Methode `Add` ein.

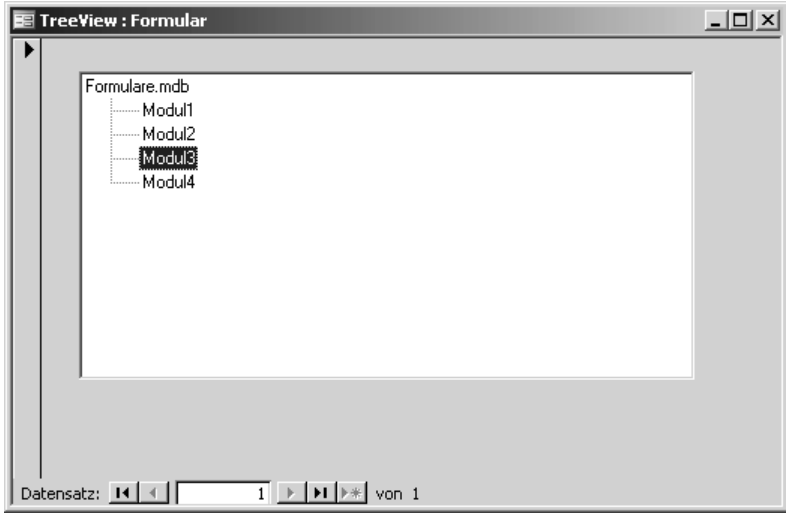


Abbildung 247: Module auflisten

323 Das Media-Player-Steuerelement

Standardmäßig wird der Media Player mit Windows ausgeliefert. Aber auch als Steuerelement steht er Ihnen in Access zur Verfügung. Im folgenden Beispiel sollen auf einem Formular über ein Listenfeld sämtliche Videodateien (*.mpg) eines Ordners angeboten werden. Über einen Doppelklick auf das gewünschte Video soll dieses dann im MEDIA-PLAYER-Steuerelement abgespielt werden.

Legen Sie zunächst ein neues, noch leeres Formular (MEDIAPLAYER) an und fügen Sie das MEDIA-PLAYER-Steuerelement, ein Listenfeld und eine Schaltfläche ein. Legen Sie das Makro aus Listing 392 hinter die Schaltfläche, um das Listenfeld mit Videodateien zu füllen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_MediaPlayer
'=====================================================

Private Sub Befehl3_Click()
   'Videodateien suchen
   Dim varfile As Variant

   With Application.FileSearch
      .NewSearch
```

Listing 392: Videodateien suchen

```
        .LookIn = "c:\Eigene Dateien\"
        .FileName = "*.mpg"
        .SearchSubFolders = True

        If .Execute() > 0 Then
           For Each varfile In .FoundFiles
              Liste1.AddItem varfile
           Next varfile
        End If
     End With
  End Sub
```

Listing 392: Videodateien suchen (Forts.)

Führen Sie eine Suche nach den MPG-Dateien durch. Verwenden Sie das Objekt `FileSearch`, um die einzelnen Dateien im angegebenen Verzeichnis zu ermitteln. Auf dieses Objekt können Sie einige Eigenschaften anwenden: Die Eigenschaft `NewSearch` setzt die Einstellungen aller Suchkriterien auf die Standardeinstellungen zurück. Mithilfe der Eigenschaft `LookIn` geben Sie bekannt, in welchem Verzeichnis die Suche beginnen soll. Die Eigenschaft `SearchSubFolders` bestimmt, ob die Suche auch in Unterverzeichnissen fortgesetzt werden soll. In diesem Fall müssen Sie diese Eigenschaft auf den Wert `True` setzen. Die Eigenschaft `FileName` gibt den Namen der Datei an. Dabei wird über das Sternchen bekannt gegeben, dass alle Dateien mit der Endung `mpg` gesucht werden sollen. Über die Methode `AddItem` fügen Sie den Namen der Videodateien im Listenfeld ein.

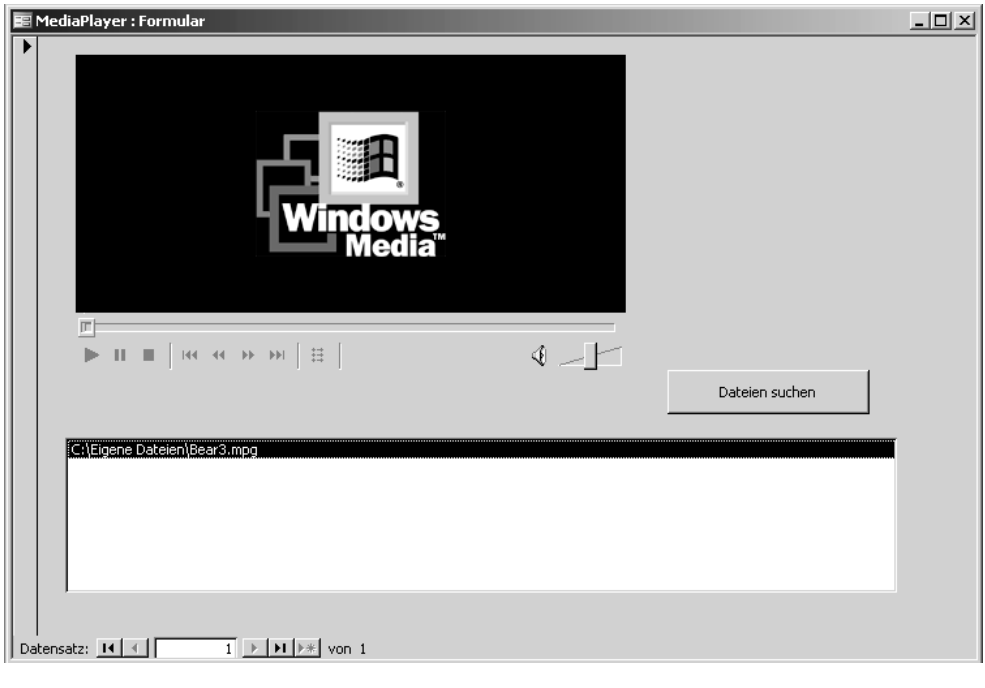


Abbildung 248: Filme im Formular abspielen

Um die Videodatei im Media-Player abzuspielen, wird das gewünschte Stück im Listenfeld doppelt angeklickt. Damit wird die Datei an den Media-Player übergeben.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_MediaPalyer
'=================================================

Private Sub Liste1_DblClick(Cancel As Integer)
   MediaPlayer0.FileName = Liste1.Value
End Sub
```

Listing 393: Videodateien abspielen

Geben Sie über die Eigenschaft `FileName` bekannt, welche Datei der Media-Player spielen soll. Dabei lesen Sie den aktuellen Eintrag im Listenfeld über die Methode `Value` aus.

324 Das Office-Chart-Steuerelement

Wenn Sie mit Diagrammen in Formularen arbeiten möchten, dann können Sie das Steuerelement MICROSOFT OFFICE CHART 10.0 (Access 2002) bzw. MICROSOFT OFFICE CHART 11.0 (Access 2003) verwenden. Fügen Sie dieses Steuerelement zunächst auf einem neuen, leeren Formular ein.

Die folgende Aufgabe besteht nun darin, die Tabelle `Artikel` der Datenbank *Formular.mdb* anzuzapfen und die fünf teuersten Artikel aus dieser Tabelle in einem Säulendiagramm anzuzeigen. Diese Aufgabe soll im Hintergrund und direkt nach dem Öffnen des Formulars stattfinden. Erfassen Sie zu diesem Zweck das Ereignis `Form_Load` aus Listing 394.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_OfficeChart
'=================================================

Private Sub Form_Load()
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim obj As AccessObject
   Dim varrWert(4) As Variant
   Dim varrBez(4) As Variant
   Dim intZ As Integer
   Dim ChSp As Object


   Set Conn = CurrentProject.Connection
```

Listing 394: Diagramm in Formular erstellen

```
    Set DBS = New ADODB.Recordset
    With DBS
       .CursorLocation = adUseClient
       .Open "Artikel", Conn, adOpenKeyset, adLockOptimistic
       .Sort = "Einzelpreis Desc"
       For intZ = 0 To 4
          varrWert(intZ) = DBS("Einzelpreis")
          varrBez(intZ) = DBS("Artikelname")
          .MoveNext
       Next intZ
       DBS.Close
    End With

    'Diagrammerstellung
    Set ChSp = ChartSpace0.Charts.Add

    With ChSp
       .Type = chChartTypeColumnClustered
       .SeriesCollection.Add
       .SeriesCollection(0).Caption = "Die teuersten Artikel"
       .SeriesCollection(0).SetData chDimCategories, chDataLiteral, varrBez
       .SeriesCollection(0).SetData chDimValues, chDataLiteral, varrWert
       .HasLegend = False
       .HasTitle = True
    End With

    Conn.Close
    Set DBS = Nothing
    Set Conn = Nothing
End Sub
```

Listing 394: Diagramm in Formular erstellen (Forts.)

Bei diesem Beispiel ist die Datenbank *Formular.mdb* bereits geöffnet. Daher können Sie sich den `Open`-Befehl sparen und stattdessen beim Öffnen der Tabelle `Artikel` auf die geöffnete Datenbank verweisen. Zu diesem Zweck haben Sie der Eigenschaft `Connection` die aktuelle Datenbank über das Objekt `CurrentProject` zugewiesen.

Über die Anweisung `Set` mit dem Zusatz `New` erstellen Sie ein neues `RecordSet`-Objekt. In diesem Objekt wird später der gefundene Satz übertragen, geändert und dann zurückgeschrieben.

Wenden Sie danach die Eigenschaft `Sort` an und geben Sie vor, nach welchen Kriterien sortiert werden soll. Haben Sie mehrere Sortierkriterien zur Auswahl, dann geben Sie diese entsprechend der Sortierreihenfolge getrennt durch Kommata ein. Bei der Sortierreihenfolge selbst können Sie entweder `ASC` für aufsteigende Sortierung oder `DESC` für absteigende Sortierung angeben. Dabei erfassen Sie nach dem Feldnamen ein Leerzeichen und hängen die gewünschte Sortierkonstante an.

In einer nachfolgenden Schleife, die genau fünf Mal durchlaufen wird, werden die Einzelpreise sowie die dazugehörenden Artikelbezeichnungen der billigsten Artikel ermittelt und in die Variablen `varrWert` und `varrBez` geschrieben.

Haben Sie die teuersten Artikel ermittelt, kann es daran gehen, diese Artikel in einem Diagramm darzustellen. Über die Methode `Add` fügen Sie ein neues Diagramm ein. Dabei geben Sie über die Eigenschaft `Type` bekannt, welcher Diagrammtyp eingesetzt werden soll. In der Online-Hilfe können Sie die dafür notwendigen Konstanten für die entsprechenden Diagramme selbst nachlesen. Es stehen Ihnen dabei weit über 50 verschiedene Diagrammtypen zur Verfügung.

Eine Datenreihe wird über die Anweisung `SeriesCollection.Add` eingefügt. Dabei erhält die erste Datenreihe den Index 0, die zweite den Index 1 usw. Da Sie für dieses Beispiel nur eine Datenreihe benötigen, in der die billigsten Artikelpreise dargestellt werden sollen, entfällt hier das Einfügen mehrerer Datenreihen. Über die Eigenschaft `Caption` wird die Beschreibung festgelegt, die später als Titel sowie auch als Legendentext verwendet werden kann.

Mithilfe der Methode `SetData` werden die Daten für das Diagramm zugewiesen. Diese Daten haben Sie vorher in den Variant-Variablen `varrWert` und `varrBez` gespeichert. Über die Eigenschaften `HasLegend` und `HasTitel` bestimmen Sie, dass eine Legende bzw. ein Diagrammtitel angezeigt werden soll.

Vergessen Sie dann nicht, die Tabelle über die Methode `Close` zu schließen und die Objektverweise wieder aufzuheben.

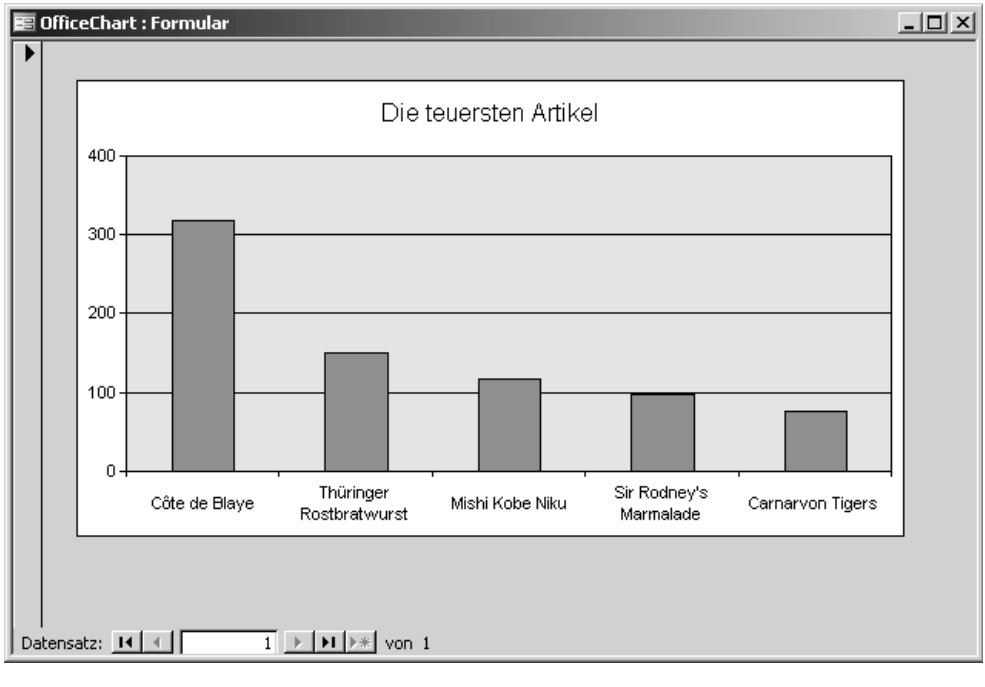


Abbildung 249: Die teuersten Artikel werden in einem Formulardiagramm angezeigt.

325 Das SpreadSheet-Steuerelement

Mithilfe des SPREADSHEET-Steuerelements können Sie auf einem Formular eine komplette Excel-Tabelle verwalten, d.h., Sie können in dieses Steuerelement einen Bereich einer Tabelle einlesen, verändern und zurückschreiben.

Im folgenden Beispiel wurden ein neues Formular mit dem Namen SPREADSHEET angelegt, das Steuerelement MICROSOFT OFFICE SPREADSHEET 11.0 (Access 2003), sowie ein Listenfeld und eine Schaltfläche eingefügt. Über den Klick auf die Schaltfläche werden alle Excel-Arbeitsmappen aus einem Verzeichnis ins Listenfeld eingelesen. Per Doppelklick auf die entsprechende Datei wird die erste Tabelle der Arbeitsmappe Zelle für Zelle in das SPREADSHEET-Steuerelement übertragen. Legen Sie jetzt das Makro aus Listing 395 direkt hinter die Schaltfläche.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_SpreadSheet
'=================================================

Private Sub Befehl5_Click()
   'Excel-Mappen ermitteln
   Dim varfile As Variant

   With Application.FileSearch
      .NewSearch
      .LookIn = "c:\Eigene Dateien\"
      .FileType = msoFileTypeExcelWorkbooks
      .SearchSubFolders = False

      If .Execute() > 0 Then
         For Each varfile In .FoundFiles
            Liste1.AddItem varfile
         Next varfile
      End If
   End With
End Sub
```

Listing 395: Excel-Arbeitsmappen einlesen

Führen Sie eine Suche nach den Excel-Arbeitsmappen durch. Dazu verwenden Sie das Objekt `FileSearch`, um die einzelnen Dateien im angegebenen Verzeichnis zu ermitteln. Auf dieses Objekt können Sie einige Eigenschaften anwenden: Die Eigenschaft `NewSearch` setzt die Einstellungen aller Suchkriterien auf die Standardeinstellungen zurück. Mithilfe der Eigenschaft `LookIn` geben Sie bekannt, in welchem Verzeichnis die Suche beginnen soll. Die Eigenschaft `SearchSubFolders` bestimmt, ob die Suche auch in Unterverzeichnissen fortgesetzt werden soll. In diesem Fall müssen Sie diese Eigenschaft auf den Wert `True` setzen. Über die Eigenschaft `FileType` können Sie ganz konkret angeben, dass Sie nur Excel-Arbeitsmappen

suchen möchten, indem Sie die Konstante `msoFileTypeExcelWorkbooks` angeben. Über die Methode `AddItem` fügen Sie den Namen der Videodateien im Listenfeld ein.

Der Doppelklick auf das Listenfeld aus Listing 396 öffnet die ausgewählte Arbeitsmappe im Hintergrund und überträgt den Inhalt der ersten Tabelle ins SPREADSHEET-Steuerelement.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_SpreadSheet
'=====================================================

Private Sub Liste1_DblClick(Cancel As Integer)
   'Tabelle in Spreadsheet-Element übertragen
   Dim intZeilen As Long
   Dim Spalten As Integer
   Dim xlApp As Object
   Dim xlMappe As Object
   Dim intz As Integer

   Set xlApp = CreateObject("Excel.Application")
   Set xlMappe = xlApp.Workbooks.Open(Me.Liste1.Value)

   Me.Spreadsheet0.Cells.ClearContents
   For intZeilen = 1 To xlMappe.Sheets(1).UsedRange.Rows.Count
      For intSpalten = 1 To xlMappe.Sheets(1).UsedRange.Columns.Count
         Me.Spreadsheet0.Cells(intZeilen, intSpalten) = _
         xlMappe.Sheets(1).Cells(intZeilen, intSpalten)
      Next intSpalten
   Next intZeilen

   xlMappe.Close savechanges:=False
   xlApp.Quit
   Set xlApp = Nothing
   Set xlMappe = Nothing
End Sub
```

Listing 396: Tabelle Zelle für Zelle übertragen

Über die Funktion `CreateObject` erstellen Sie zunächst einen Verweis auf die Microsoft Excel-Bibliothek. Damit gewinnen Sie Zugriff auf alle Methoden und Eigenschaften, die für Excel-Arbeitsmappen verfügbar sind. So öffnen Sie über die Methode `Open` die im Listenfeld markierte Excel-Arbeitsmappe. Danach löschen Sie sicherheitshalber alle Zellen des SPREADSHEET-Steuerelements, indem Sie die Methode `ClearContents` auf alle Zellen des Elements anwenden. Über die Anweisung `UsedRange.Rows.Count` ermitteln Sie die Anzahl der verwendeten Zeilen der ersten Tabelle. Analog dazu fragen Sie die Anzahl der verwendeten Spalten über die Anweisung `UsedRange.Columns.Count` ab. Mithilfe von zwei aufeinander folgenden Schleifen übertragen Sie die einzelnen Zellen in das Steuerelement SPREADSHEET. Dabei können Sie jede einzelne Zelle des Steuerelements sowie der Tabelle über die `Cells`-Auflistung ansteuern, der Sie die dynamische Zeilen- und Spaltenangabe übergeben.

Wenden Sie am Ende des Makros die Methode `Close` an, um die Arbeitsmappe zu schließen, sowie die Methode `Quit`, um die Excel-Sitzung zu beenden. Heben Sie danach die Objektverweise auf, indem Sie diese auf `Nothing` setzen (siehe Abbildung 250).

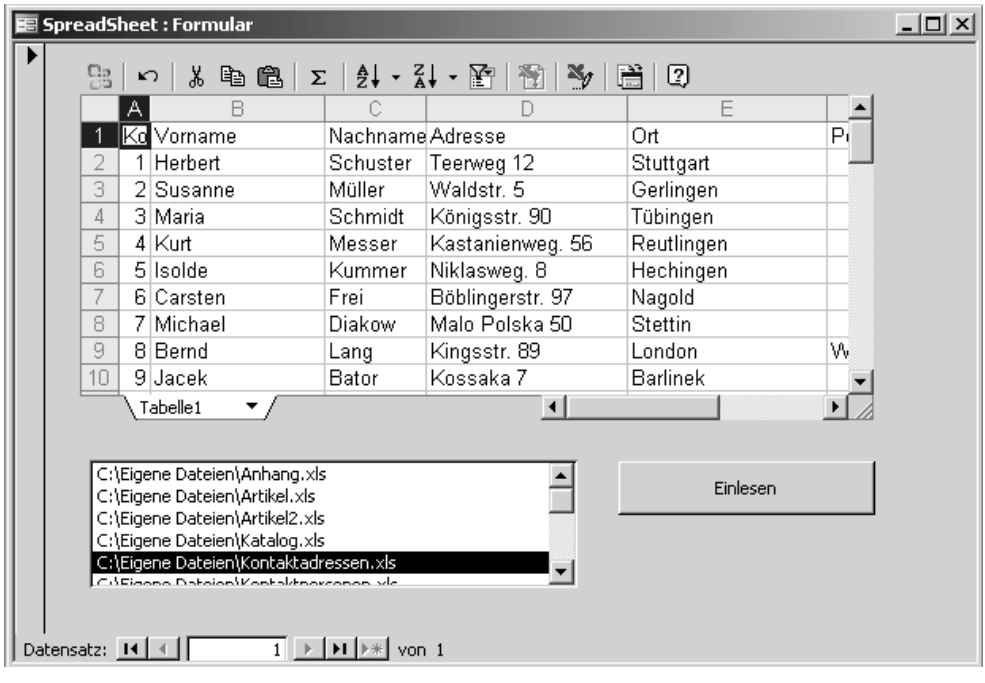


Abbildung 250: Die Excel-Tabelle wird im Access-Formular angezeigt.

326 Das Slider-Steuerelement

Unter dem Steuerelement SLIDER versteht man einen Schieberegler, über den man Werte in einem bestimmten Wertebereich, beispielsweise 0–100, einstellen kann. Dieses Steuerelement wird gerne in Verbindung mit einem Textfeld verwendet, indem man den eingestellten Wert des SLIDER anzeigt.

Das Steuerelement MICROSOFT SLIDER CONTROL können Sie aktivieren, indem Sie das Symbol WEITERE STEUERELEMENTE in der Toolbox anklicken und das Steuerelement aus der Liste auswählen. Fügen Sie dieses Steuerelement und ein Textfeld in einem neuen Formular mit dem Namen SLIDER ein.

Klicken Sie das SLIDER-Element in der Entwurfsansicht mit der rechten Maustaste an und wählen Sie den Befehl SLIDER-OBJEKT/PROPERTIES aus dem Kontextmenü (siehe Abbildung 251).

Im Dialog aus Abbildung 251 lässt sich das Slider-Steuerelement einstellen. Um nun das Textfeld mit dem SLIDER zu synchronisieren, erfassen Sie das Ereignis `Scroll` des SLIDER.

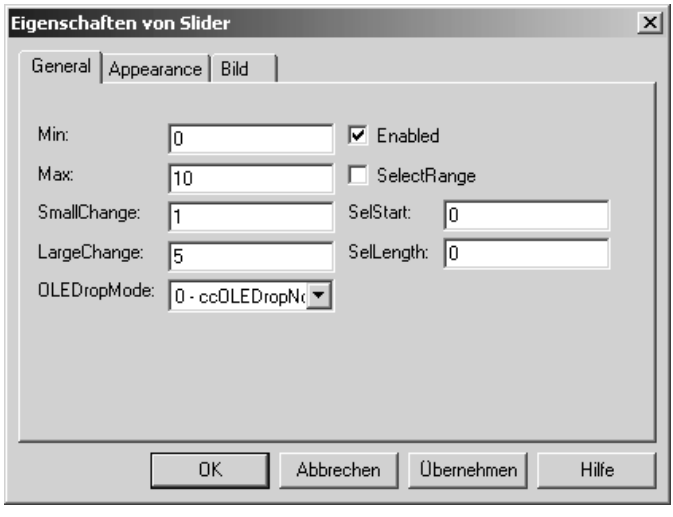


Abbildung 251: En Slider einstellen

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Slider
'=================================================

Private Sub Slider0_Scroll()
   'Textfeld mit Slider synchronisieren
   Text1.Value = Slider0.Value
End Sub
```

Listing 397: Übertragen des Slider-Status ins Textfeld

Über die Methode `Value` fragen Sie den aktuellen Status des SLIDER ab und übertragen ihn in das Textfeld.

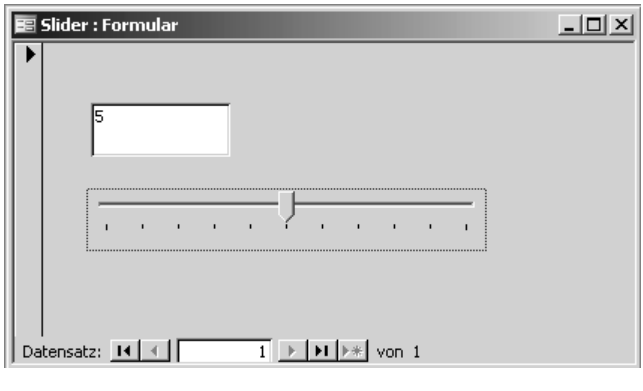


Abbildung 252: Slider und Text

327 Das Spinbutton-Steuerelement

Über das SPINBUTTON-Steuerelement können Sie elegant über zwei Drehknöpfe Werte in einem bestimmten Wertebereich in einem verknüpften Textfeld erstellen. Dabei kann die Schrittweite selbst festgelegt werden.

Legen Sie ein neues Formular mit dem Namen SPINBUTTON an und integrieren Sie ein Textfeld sowie das Steuerelement MICROSOFT FORMS 2.0 SPINBUTTON. Beim Öffnen des Formulars definieren Sie in Listing 398 den kleinsten sowie größten Wert, den Sie über den SPINBUTTON einstellen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Spinbutton
'=====================================================

Private Sub Form_Load()
   With Me
      .SpinButton3.Max = 100
      .SpinButton3.Min = 1
   End With
End Sub
```

Listing 398: Die Grenzwerte des Spinbutton festlegen

Über die Eigenschaft `Max` legen Sie den größtmöglichen Wert des SPINBUTTON fest. Die Eigenschaft `Min` liefert den kleinsten Wert.

Um den augenblicklichen Wert des SPINBUTTON ins Textfeld zu übertragen, erfassen Sie das Makro aus Listing 399.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap06
' Dateiname  Formulare.mdb
' Klasse     Form_Spinbutton
'=====================================================

Private Sub SpinButton3_Change()
   'Übertragen Spinbutton-Status ins Textfeld
   With Me
      .Text4.Value = .SpinButton3.Value
   End With
End Sub
```

Listing 399: Übertragen des Spinbutton-Werts ins Textfeld

Über die Methode `Value` fragen Sie den aktuellen Status des SPINBUTTON ab und übertragen ihn in das Textfeld.

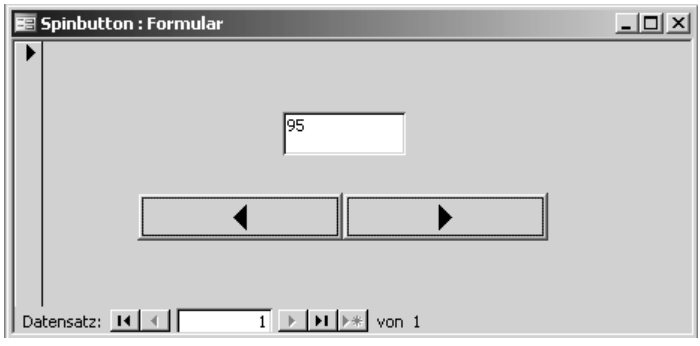


Abbildung 253: Der Spinbutton kennt Werte von 1 bis 100.

Berichte und Datenzugriffsseiten

Mithilfe von Berichten haben Sie die Möglichkeit, Daten wirkungsvoll in gedruckter Form auszugeben. Beim Gestalten der Berichte stehen Ihnen viele Wege offen, da Sie die Größe und Darstellung aller Bestandteile eines Berichts selbst bestimmen können.

Sie können Datenbanken selbstverständlich nicht nur lokal betreiben, sondern auch im Internet so genannte Datenzugriffsseiten ablegen und diese dann einsetzen, um Ihren lokalen Datenbestand abzufragen bzw. zu aktualisieren.

328 Berichtstypen

Access unterscheidet diverse Berichte, die Sie alle mit dem Berichts-Assistenten erstellen können. Genau diese Vorgehensweise empfiehlt sich auch vor der Programmierung von Berichten.

Bevor Sie einen neuen Bericht über den Berichts-Assistenten erstellen, müssen Sie sich Gedanken machen, welche Art von Bericht Sie einsetzen möchten. Access bietet Ihnen dazu folgende Berichtsarten an:

1. DETAILBERICHTE: Bei einem Detailbericht gibt es zu jedem einzelnen Datensatz auch einen Satz im Bericht. Diese Sätze werden dann nach bestimmten Gesichtspunkten gruppiert. Damit ist ein lückenloser Nachweis möglich. Bei Detailberichten werden Fragen nach Lagerbewegungen, Verkäufen, Bestellungen usw. beispielsweise in einem bestimmten Zeitraum beantwortet.

2. BERICHTE MIT ZUSAMMENFASSUNG: Bei diesen Berichten werden nicht mehr alle Daten ausgegeben. Die Daten werden im Vorfeld gruppiert und zusammengefasst. Als Ergebnis erhalten Sie nur noch wenige Zeilen, also eine verdichtete Darstellung.

3. BERICHTE MIT DIAGRAMMEN: Bei dieser Berichtsform werden die Daten über ein Diagramm zusätzlich aufbereitet. Auch diese Aufgabe können Sie ohne eine einzige Zeile VBA standardmäßig sehr schnell ausführen.

4. FORMULARBERICHTE: Der Berichts-Assistent unterstützt ebenso Berichte, die wie Formulare aussehen können.

5. ETIKETTENBERICHTE: Mithilfe des Etiketten-Assistenten können Sie Etiketten erstellen und dabei die Etikettengröße sowie die Seitenmaße festlegen.

329 Der Aufbau eines Berichts

Wenn Sie einen Bericht erstellen, dann hat dieser standardmäßig einen vorgegebenen Aufbau:

- Im SEITENKOPF können Sie Informationen ablegen, wie Überschriften, Logos, Datum usw., die auf jeder Druckseite wiederholt werden.
- Der DETAILBEREICH enthält die wirklichen Daten des Berichts. Diese Daten werden aus einer Tabelle bzw. einem Bericht geholt.
- Im SEITENFUSS geben Sie Daten wie die Seitennummerierung, den Namen des Berichts und seinen Speicherort aus.

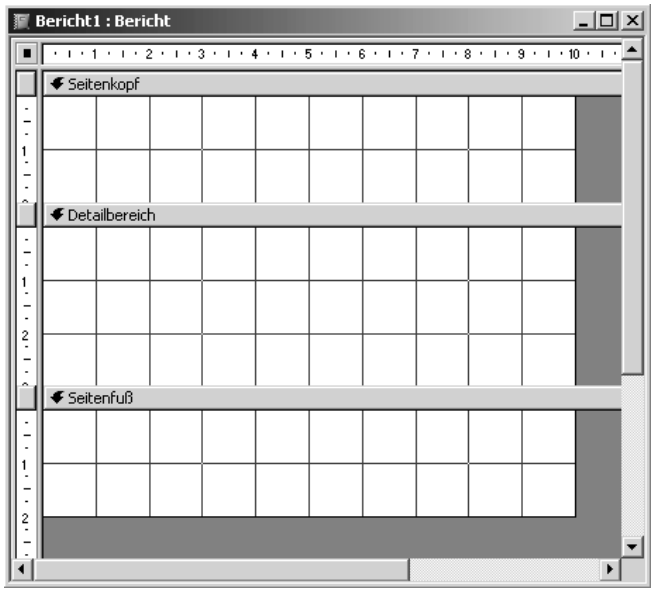


Abbildung 254: Die Grundbestandteile eines Berichts

Es gibt noch weitere Bestandteile eines Berichts, die bei Bedarf verwendet werden können. So können Sie zu Beginn eines Berichts ein Titelblatt definieren, welches als Berichtskopf bezeichnet wird. Dieser Berichtskopf wird dann nur einmalig gedruckt. Genauso gibt es auch einen Berichtsfuß, der am Ende eines Berichts als Abschluss gedruckt werden kann.

Innerhalb des Detailbereichs können Sie mehrere Gruppenköpfe und Gruppenfüße einrichten. Mithilfe dieser Elemente können Sie die Daten gruppieren und übersichtlich anordnen.

330 Berichte öffnen

Für einfachere Aktionen wie das Öffnen, Suchen, Drucken und Schließen von Berichten können Sie mit dem Objekt `DoCmd` arbeiten. Die Methoden des `DoCmd`-Objekts können Sie verwenden, um Microsoft Access-Aktionen aus Visual Basic heraus auszuführen.

Mithilfe der Methode `OpenReport` öffnen Sie einen Bericht in Access. Die Syntax dieser Methode lautet:

```
OpenReport(ReportName, View, FilterName, WhereCondition, WindowMode, OpenArgs)
```

Argument	Beschreibung
`ReportName`	Erforderlich. Gibt den Namen des Berichts an, den Sie öffnen möchten.
`View`	Optional. Bestimmt die Ansicht des Berichts. Es stehen Ihnen dabei folgende Konstanten zur Verfügung: `acViewDesign`: öffnet den Bericht in der Entwurfsansicht. `acViewNormal`: erstellt den Bericht und druckt ihn direkt aus. Sie bekommen den Bericht nicht angezeigt (Standardeinstellung). `acViewPreview`: zeigt den Bericht in der Seitenansicht an.
`FilterName` und `WhereCondition`	Optional. Stellt einen Filter ein.
`WindowMode`	Optional. Legt fest, wie das Formular angezeigt werden soll. `acWindowNormal`: zeigt den Bericht in der Standardansicht an. `acHidden`: zeigt den Bericht im ausgeblendeten Zustand an `acIcon`: blendet den Bericht unten am Bildschirm in der Titelleiste als kleines Symbol ein. `acDialog`: zeigt den Bericht als Dialog an. Dabei können Sie mit einer anderen Aufgabe erst wieder weiterarbeiten, wenn Sie den Bericht wieder geschlossen haben.
`OpenArgs`	Optional. Beim Öffnen des Berichts können bestimmte Werte bzw. Einstellungen mitgegeben werden.

Tabelle 123: Die Argumente der Methode OpenReport

Im folgenden Beispiel aus Listing 400 wird der Bericht UMSÄTZE NACH KATEGORIE in der Vorschau angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtÖffnen()
   On Error GoTo fehler
   DoCmd.OpenReport "Umsätze nach Kategorie", acViewPreview
   DoCmd.Maximize
   Exit Sub
```

Listing 400: Bericht in der Vorschau öffnen

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 400: Bericht in der Vorschau öffnen (Forts.)

Wenden Sie die Methode `OpenReport` an, um einen Bericht zu öffnen. Da jeder Bericht standardmäßig gleich gedruckt wird, verwenden Sie hier die Konstante `acPreview`. Über die Methode `Maximize` zeigen Sie den Bericht in der maximierten Ansicht an.

331 Berichtsfilter setzen

Im nächsten Beispiel aus Listing 401 wird der Bericht ALPHABETISCHE ARTIKELLISTE aufgerufen und dabei ein Filter eingestellt. Es sollen nur Artikel ausgegeben werden, die vom Lieferanten mit der Nummer 4 stammen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtÖffnenUndVoreinstellen()
   On Error GoTo fehler
   DoCmd.OpenReport "Alphabetische Artikelliste", _
   acViewPreview, , "[Lieferanten-Nr] = 4"
   DoCmd.RunCommand acCmdZoom100
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 401: Bericht aufrufen und Filter setzen

Stellen Sie als Filterkriterium eine Lieferantenauswahl ein. Da der Feldnamen einen Bindestrich aufweist, muss der Ausdruck im Filter in eckige Klammern gesetzt werden. Über die Methode `RunCommand`, der Sie die Konstante `acCmdZoom100` zuweisen, wird der Bericht 1:1 am Bildschirm ausgegeben.

332 Bericht schließen

Über die Methode `Close` können Sie einen Bericht schließen. Die Syntax dieser Methode lautet:

```
Close(Objekttyp, Objektname, Speichern)
```

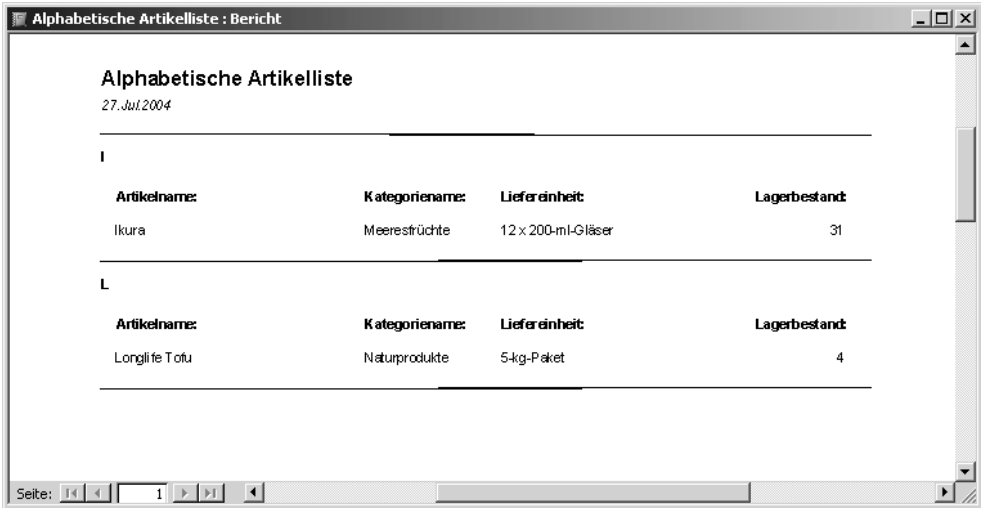


Abbildung 255: Nur bestimmte Datensätze anzeigen

Argument	Beschreibung
`ObjectType`	Optional. Legt den Objekttyp fest: `acDataAccessPage`, `acDefault` (Standard), `acDiagram`, `acForm`, `acFunction`, `acMacro`, `acModule`, `acQuery`, `acReport`, `acServerView`, `acStoredProcedure`, `acTable`
`Objektname`	Optional. Gibt den Namen des Objekts vom Typ `ObjektType` an.
`Save`	Optional. Dabei gelten folgende Konstanten: `acSaveNo`: keine Speicherung `acSavePrompt` (Standard): Dieser Wert wird ignoriert, falls Sie ein Visual Basic-Modul schließen. Das Modul wird geschlossen, Änderungen an dem Modul werden jedoch nicht gespeichert. `acSaveYes`: Wenn Sie dieses Argument nicht angeben, wird der Standardwert (`acSavePrompt`) verwendet.

Tabelle 124: Die Argumente der Methode Close

333 Berichte drucken

Standardmäßig werden Berichte sofort gedruckt, wenn Sie diese aufrufen. Der Bericht wird dann einmal komplett ausgedruckt. Sie haben vorher keine Möglichkeit, Einschränkungen der Datensätze bzw. die Anzahl der Kopien einzustellen. Mithilfe der Methode `PrintOut` gelingt Ihnen dieses Vorhaben jedoch. Die Syntax dieser Methode lautet:

```
PrintOut (Druckbereich, Von, Bis, Druckqualität, Exemplare, ExemplareSortieren)
```

Argument	Beschreibung
Druckbereich	Optional. Der Druckbereich wird angegeben. Dabei stehen folgende Konstanten zur Verfügung: acPages: Bei dieser Konstante können Sie die Seitenzahlen angeben, die Sie ausdrucken möchten. Dabei müssen Sie im nächsten Argument die genauen Werte angeben. acPrintAll: Diese Konstante bewirkt, dass der komplette Bericht ausgedruckt wird. Hier handelt es sich um eine Standardeinstellung von Access. acSelection: Bei dieser Konstanten werden nur die markierten Datensätze ausgedruckt.
Von und Bis	Optional. Gibt an, welche Seiten gedruckt werden sollen. Dieses Argument brauchen Sie natürlich nur dann anzugeben, wenn Sie im Argument Druckbereich die Konstante acPages angegeben haben.
Druckqualität	Optional. Bestimmt die Qualität und nicht zuletzt auch die Geschwindigkeit des Druckvorgangs. Hierfür stehen folgende Konstanten zur Verfügung: acDraft: Mit dieser Konstanten drucken Sie den Bericht in Entwurfsqualität aus. acHigh: Diese Druckqualität ist sehr hoch und hat damit die längste Druckdauer (Standard). acLow: niedrige Druckqualität mit deutlichen Vorteilen bei der Ausdruckgeschwindigkeit. acMedium: mittlere Druckqualität als Kompromiss zwischen Druckgeschwindigkeit und Druckqualität.
Exemplare	Optional. Bestimmt die Anzahl der Kopien, die Sie von dem Bericht wünschen. Der Standardwert liegt hier bei einem Exemplar und kann daher auch weggelassen werden.
ExemplareSortieren	Optional. Gibt an, wenn Sie mehrere Kopien des Berichts drucken und den Ausdruck danach nicht von Hand sortieren möchten. Geben Sie hier den Wert True an, um die Berichtsseiten während des Druckvorgangs zu sortieren.

Tabelle 125: Die Argumente der Methode PrintOut

Im folgenden Beispiel aus Listing 402 werden die ersten beiden Seiten des Berichts UMSÄTZE NACH KATEGORIE direkt auf den Drucker geschickt und mit fünf Kopien in mittlerer Qualität gedruckt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtDrucken()
   DoCmd.OpenReport "Umsätze nach Kategorie", acViewPreview
   DoCmd.PrintOut acPages, 1, 2, acMedium, 5, True
End Sub
```

Listing 402: Bericht benutzerdefiniert ausdrucken

Vor dem Drucken steht Ihnen die Eigenschaft `FastLaserPrinting` zur Verfügung. Mit dieser Eigenschaft können Sie angeben, ob Linien und Rechtecke durch Linien aus Textzeichen (vergleichbar mit den Zeichen Unterstrich (_) und vertikaler Strich (|)) ersetzt werden, wenn Sie ein Formular oder einen Bericht mit einem der gängigen Laserdrucker drucken. Das Drucken kann erheblich schneller erfolgen, wenn Linien und Rechtecke durch Linien aus Textzeichen ersetzt werden.

Im folgenden Beispiel aus Listing 403 werden die ersten beiden Seiten des Berichts KATALOG in der schnelleren Laservariante ausgedruckt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub NochSchnellerDrucken()
   DoCmd.OpenReport "Katalog", acDesign
   Reports!Katalog.FastLaserPrinting = True
   DoCmd.PrintOut acPages, 1, 2, acMedium, , True
End Sub
```

Listing 403: Den noch schnelleren Laserdruck einstellen

334 Berichte kopieren

Möchten Sie einen Bericht zur Sicherheit in eine andere Datenbank transportieren, können Sie hierfür die Methode `CopyObject` einsetzen. Die Syntax dieser Methode lautet:

```
CopyObject(Zieldatenbank, NeuerName, Quellobjekttyp, Quellobjektname)
```

Argument	Beschreibung
Zieldatenbank	Die Kopieraktion führen Sie von der aktuell geöffneten Datenbank aus durch. Im Argument Zieldatenbank geben Sie den Pfad sowie den Namen der Datenbank an, in welche Sie den Bericht kopieren möchten. Lassen Sie dieses Argument leer, wenn Sie den Bericht innerhalb der aktuellen Datenbank kopieren möchten.
NeuerName	Optional. Geben Sie an, wie der kopierte Bericht heißen soll.
Quellobjekttyp	Optional. Geben Sie in einer Konstanten an, ob Sie einen Bericht (acReport), eine Tabelle (acTable), eine Abfrage (acQuery), ein Modul (acModule) oder sonstige Objekte kopieren möchten.
Quellobjektname	Optional. Geben Sie den Namen des Berichts an, den Sie kopieren möchten.

Tabelle 126: Die Argumente der Methode CopyObject

Im folgenden Beispiel aus Listing 404 wird der Bericht KATALOG kopiert und unter dem Namen KATALOGNEU gespeichert

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=================================================

Sub BerichtKopieren()
   On Error GoTo fehler
   DoCmd.CopyObject , "KatalogNeu", acReport, "Katalog"
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 404: Einen Bericht in der aktuellen Datenbank kopieren

Soll der Bericht in eine andere Datenbank kopiert werden, dann sehen Sie sich das Makro aus Listing 405 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtKopierenErw()
   On Error GoTo fehler
   DoCmd.CopyObject  _
      "C:\Eigene Dateien\Nordwind.mdb", "Artikelkatalog", acReport, "Katalog"
   MsgBox "Kopieraktion durchgeführt!"
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 405: Bericht in eine andere Datenbank kopieren

335 Berichte umbenennen

Um einem Bericht einen anderen Namen zu geben, setzen Sie die Methode `Rename` ein. Die Syntax dieser Methode lautet:

```
Ausdruck.Rename(NewName, ObjectType, OldName)
```

Argument	Beispiel
`NewName`	Erforderlich. Gibt an, wie das Objekt nach der Umbenennung heißen soll.
`ObjectType`	Optional. Gibt an, ob Sie einen Bericht (`acReport`), eine Tabelle (`acTable`), eine Abfrage (`acQuery`) oder ein sonstiges Objekt umbenennen möchten. Es gelten hierbei dieselben Konstanten wie schon vorher beschrieben bei der Methode `CopyObject`.
`OldName`	Optional. Beinhaltet den alten Namen des Objekts.

Tabelle 127: Die Argumente der Methode Rename

Im folgenden Beispiel aus Listing 406 wird der Bericht KATALOGNEU umbenannt. Der Name des neuen Berichts lautet dann ARTIKELANGEBOT.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=================================================

Sub BerichtUmbenennen()
   On Error GoTo fehler
   DoCmd.Rename "Artikelangebot", acReport, "KatalogNeu"
   Exit Sub


  fehler:
     MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 406: Bericht umbenennen

336 Berichte exportieren

Mit der Methode `OutputTo` können Sie die Daten in einem bestimmten Microsoft Access-Datenbankobjekt (einem Datenblatt, einem Formular, einem Bericht, einem Modul oder einer Datenzugriffsseite) in verschiedenen Formaten ausgeben. Dabei lautet die Syntax dieser Methode wie folgt:

```
OutputTo(ObjectType, ObjectName, OutputFormat, OutputFile, AutoStart, TemplateFile)
```

Argument	Beschreibung
`ObjectType`	Erforderlich. Legt die Art des Access-Objekts fest, dessen Daten exportiert werden sollen. Dabei haben Sie folgende Möglichkeiten: `acOutputForm`: Export der Daten eines Formulars `acOutputFunction`: Export einer Funktion zur Sicherung `acOutputModule`: Export eines kompletten Moduls inklusive aller Funktionen und Makros `acOutputQuery`: Export der Ergebnisse einer Abfrage `acOutputReport`: Export eines Berichts `acOutputServerView`: Export einer Serveransicht `acOutputStoredProcedure`: Export einer gespeicherten Prozedur `acOutputTable`: Export einer Tabelle
`ObjectName`	Optional. Gibt den Namen des Objekts an, das exportiert werden soll.

Tabelle 128: Die Argumente der Methode OutPutTo

Argument	Beschreibung
OutPutFormat	Optional. Legt fest, in welchem Datenformat die Daten transferiert werden sollen. Die bekanntesten Formate heißen dabei wie folgt: acFormatHTML: konvertiert die Daten in das HTML-Format. acFormatRTF: konvertiert die Daten in das Rich-Text-Format. Dieses Format kann beispielsweise problemlos in Microsoft Word eingelesen werden. acFormatTXT: Mit diesem Format ist das Textformat gemeint. acFormatXLS: konvertiert die Daten in das Microsoft Excel-Format.
OutputFile	Optional. Gibt den Pfad sowie den Dateinamen der Datei an, in welche die Daten transferiert werden sollen. Dabei muss die Datei noch nicht vorhanden sein. Access legt diese bei Bedarf selber an.
AutoStart	Optional. Bietet die Möglichkeit, die so erstellte Exportdatei gleich zu öffnen. Verwenden Sie den Wert True, um die entsprechende auf Windows basierende Anwendung sofort zu starten. Setzen Sie das Argument auf den Wert False oder lassen Sie es weg, wenn Sie die Exportdatei nicht öffnen möchten.
TemplateFile	Optional. Ist von Interesse, wenn Sie eine Vorlage beispielsweise für die HTML-Datei verwenden möchten. In diesem Fall ist dann der komplette Pfad dieser Vorlagendatei anzugeben.

Tabelle 128: Die Argumente der Methode OutPutTo (Forts.)

Im folgenden Beispiel aus Listing 407 wird der Bericht UMSÄTZE NACH JAHR in eine Excel-Tabelle transferiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtExportieren()
   On Error GoTo fehler
   DoCmd.OutputTo acOutputReport, _
   "Umsätze nach Jahr", acFormatXLS, "C:\Eigene Dateien\Umsätze.xls", True
   Exit Sub
```

Listing 407: Bericht exportieren

```
    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 407: Bericht exportieren (Forts.)

337 Berichte formatieren

Für die Formatierung von Berichten stehen Ihnen eine ganze Reihe von Eigenschaften zur Verfügung – je nachdem, welches Berichtsobjekt Sie dabei formatieren möchten.

Im folgenden Beispiel aus Listing 408 wird dem Bericht UMSÄTZE PRO JAHR der zusätzliche Text `Kopie` hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Klasse     Report_Umsätze nach Jahr
'=====================================================

Private Sub Berichtskopf_Format(Cancel As Integer, FormatCount As Integer)
   Dim rpt As Report
   Dim strText As String

   Set rpt = Me
   strText = "Kopie"
   With rpt
      .ScaleMode = 3
      .FontName = "Courier"
      .FontSize = 12
      .FontBold = True
      .ForeColor = RGB(256, 0, 0)
      .CurrentX = rpt.ScaleWidth / 2
   End With
   rpt.Print strText
End Sub
```

Listing 408: Ein zusätzliches Label einfügen – Variante 1

Über die Eigenschaft `ScaleMode` wird die Maßeinheit festgelegt. Dabei bedeutet die Nummer 3, dass die Maßeinheit in Pixel angegeben wird. Mithilfe der Eigenschaft `FontName` können Sie die Schriftart bestimmen, in der das zusätzliche Label eingefügt werden soll. `FontSize` bestimmt die Größe der Schrift. Den Schriftschnitt Fett legen Sie über die Eigenschaft `FontBold` fest, die Sie auf den Wert `True` setzen. Die Schriftfarbe lässt sich über die Eigenschaft `ForeColor` einstellen. Die eigentliche Farbe mischen Sie sich über die Funktion `RGB` (Red-Green-Blue) zusammen. Den genauen Ort im Bericht für das zusätzliche Label legen Sie über die Eigenschaft `CurrentX` fest. Dabei legen Sie fest, dass das Label etwa in der Mitte des Berichts eingefügt werden soll.

Weitere Eigenschaften für Formatierungen können Sie der Tabelle 129 entnehmen.

Abbildung 256: Das zusätzliche Label wurde eingefügt.

Eigenschaft	Beschreibung
`BackColor`	Legt die Hintergrundfarbe fest.
`FontBold`	Formatiert die Schrift fett sowie die Rahmenfarbe für Rechtecke und Kreise.
`FillColor`	Stellt die Füllfarbe für Rechtecke und Kreise dar.
`FillStyle`	Legt das Füllmuster für Rechtecke und Kreise fest.
`ForeColor`	Legt die Farbe des Textes fest.
`FontItalic`	Formatiert die Schrift kursiv.
`FontName`	Legt den Namen der Schriftart fest.
`FontSize`	Legt die Schriftgröße fest.
`FontUnderline`	Unterstreicht die einzelnen Zeichen.

Tabelle 129: Die wichtigsten Eigenschaften rund um die Formatierung von Berichten

Im Makro aus Listing 408 wurde die Eigenschaft `RGB` verwendet, um die Schriftfarbe eines Labels zu bestimmen. Alternativ gibt es eine weitere Funktion, um Farben zu erzeugen. Diese Funktion heißt `QBColor`. Über diese Funktion können 15 Standardfarben abgerufen werden, die Sie in der Tabelle 130 sehen können.

Farbnummer	Farbe
0	Schwarz
1	Blau
2	Grün
3	Cyan
4	Rot
5	Magenta
6	Gelb
7	Weiß

Tabelle 130: Die Standardfarben über die Funktion BQColor

Farbnummer	Farbe
8	Grau
9	Hellblau
10	Hellgrün
11	Hellcyan
12	Hellrot
13	Hellmagenta
14	Hellgelb
15	Leuchtendes Weiß

Tabelle 130: Die Standardfarben über die Funktion BQColor (Forts.)

Im folgenden Beispiel aus Listing 408 wird das Beispiel aus Listing 409 über den Einsatz der Funktion `QBColor` gelöst.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Klasse     Report_Umsätze nach Jahr
'=====================================================

Private Sub Berichtskopf_Format(Cancel As Integer, FormatCount As Integer)
   Dim rpt As Report
   Dim strText As String

   Set rpt = Me
   strText = "Kopie"
   With rpt
      .ScaleMode = 3
      .FontName = "Courier"
      .FontSize = 12
      .FontBold = True
      .ForeColor = QBColor(4)
      .CurrentX = rpt.ScaleWidth / 2
   End With
   rpt.Print strText
End Sub
```

Listing 409: Ein zusätzliches Label einfügen – Variante 2

Beim nächsten Beispiel wird ein Bericht farblich etwas angepasst. Dabei soll jeweils eine Zeile des Berichts mit der Farbe WEISS und die folgende Zeile mit der Farben GRÜN formatiert werden. Dazu setzen Sie das Ereignis `Format` des Detailbereichs wie in Listing 410 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Klasse     Report_Katalog Unterformular
'=====================================================

Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)
   Static i As Integer
   If i Mod 2 = 0 Then
      Me.Detailbereich.BackColor = QBColor(2)
   Else
      Me.Detailbereich.BackColor = QBColor(7)
   End If
End Sub
```

Listing 410: Bericht im Wechsel einfärben

Die statische Variable zu Beginn des Ereignisses wird jeweils um den Wert 1 erhöht, wenn das Ereignis ausgeführt wird, also eine Formatierung vorgenommen wurde. Innerhalb der Schleife wird über die Funktion `Mod` geprüft, ob die Division ein gerades bzw. ungerades Ergebnis liefert. Je nach Ergebnis wird mithilfe der Eigenschaft `BackColor` der Hintergrund des Berichts angesprochen. Über den Einsatz der Funktion `RGB` wird dann jeweils die gewünschte Farbe festgelegt.

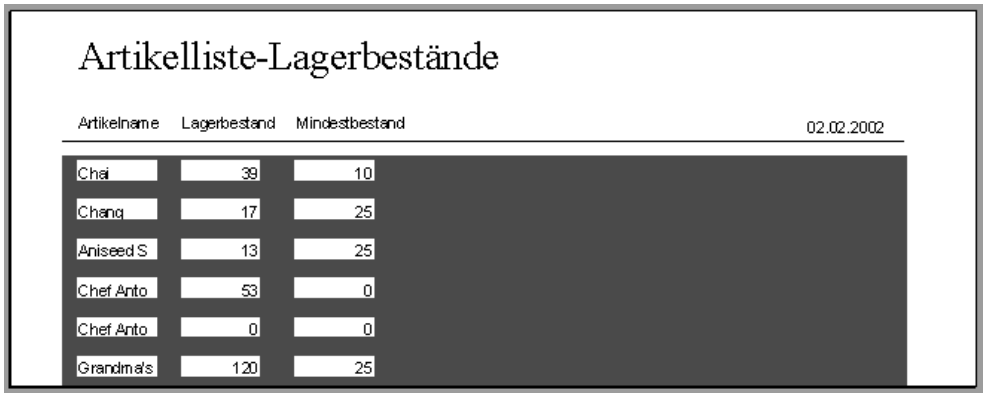


Abbildung 257: Der etwas andere »Look« im Bericht

338 Berichte auflisten

Mit der Eigenschaft `AllReports` können Sie auf die `AllReports`-Auflistung und deren dazugehörige Eigenschaften zugreifen.

Im folgenden Beispiel aus Listing 411 werden die Namen aller Berichte der Datenbank im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub AlleBerichteAuflisten()
   Dim obj As AccessObject
   Dim Dbs As Object

   Set Dbs = Application.CurrentProject
   For Each obj In Dbs.AllReports
      Debug.Print obj.Name
   Next obj
End Sub
```

Listing 411: Alle Berichte auflisten

Durchlaufen Sie mit einer Schleife alle in der Datenbank befindlichen Berichte, die im Auflistungsobjekt `AllReports` verzeichnet sind. In der Schleife geben Sie die Namen der einzelnen Berichte mithilfe der Eigenschaft `Name` aus.

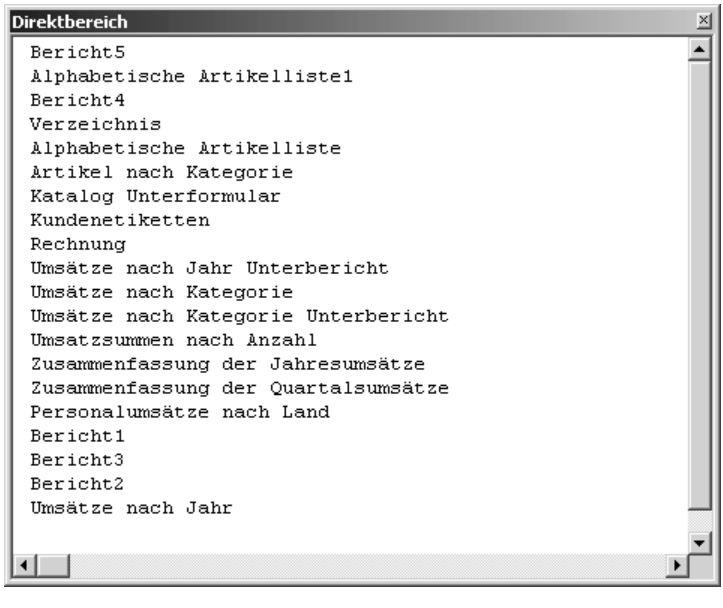


Abbildung 258: Die Berichte der aktuellen Datenbank wurden dokumentiert.

339 Berichtselemente identifizieren

Gehen Sie nun noch einen Schritt weiter und identifizieren Sie alle Elemente eines bestimmten Berichts. Im Makro aus Listing 412 öffnen Sie den Bericht UMSÄTZE NACH KATEGORIE in

der Entwurfsansicht, lesen danach alle verwendeten Steuerelemente aus und geben diese im Direktbereich aus. Schließen Sie danach den Bericht mithilfe der Methode `Close`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub AlleElementeVonBerichtAuflisten()
   Dim obj As Object
   Dim Bericht As Report

   DoCmd.OpenReport "Umsätze nach Kategorie", acViewDesign
   Set Bericht = Reports("Umsätze nach Kategorie")
   For Each obj In Bericht
      Debug.Print obj.Name
   Next obj
   DoCmd.Close
End Sub
```

Listing 412: Alle Berichtselemente auflisten

Geben Sie in der Objektvariablen vom Typ `Report` den Namen Ihres Berichts ein, den Sie näher unter die Lupe nehmen möchten. Setzen Sie danach eine Schleife auf, die die einzelnen Elemente nacheinander über die Eigenschaft `Name` ermittelt und über die Anweisung `Debug.Print` im Direktfenster von Access ausgibt.

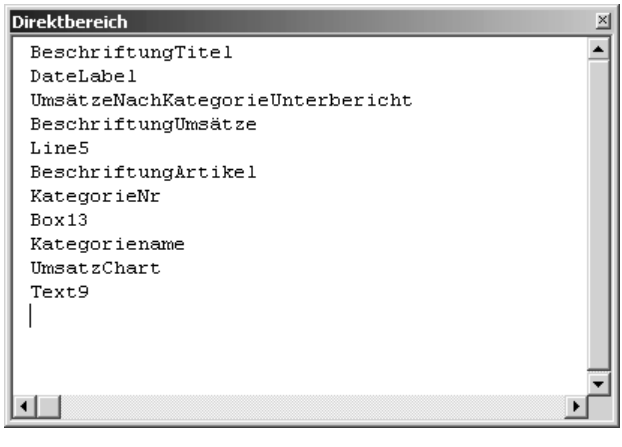


Abbildung 259: Die Berichtselemente wurden aufgelistet.

Mithilfe der `ControlType`-Eigenschaft in Visual Basic können Sie den Typ eines Steuerelements in einem Formular oder einem Bericht bestimmen.

Die Typen der einzelnen Steuerelemente können Sie aber auch über Konstanten ermitteln. Sehen Sie sich dazu die folgende Tabelle an.

Konstante	Steuerelement
acBoundObjectFrame	Gebundenes Objektfeld
acCheckBox	Kontrollkästchen
acComboBox	Kombinationsfeld
acCommandButton	Befehlsschaltfläche
acCustomControl	ActiveX-Steuerelement
acImage	Bild
acLabel	Bezeichnungsfeld
acLine	Linie
acListBox	Listenfeld
acObjectFrame	Ungebundenes Objektfeld oder Diagramm
acOptionButton	Optionsschaltfläche
acOptionGroup	Optionsgruppe
acPage	Seitenwechsel
acPageBreak	Seitenumbruch
acRectangle	Rechteck
acSubform	Unterformular/-bericht
acTabCtl	Registersteuerelement
acTextBox	Textfeld
acToggleButton	Umschaltfläche

Tabelle 131: Die Steuerelementskonstanten der Eigenschaft ControlType

Im folgenden Beispiel aus Listing 413 werden die Steuerelemente des Berichts UMSÄTZE NACH KATEGORIE identifiziert.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'======================================================

Sub FormularSteuerelementNamen()
   Dim obj As Object
   Dim strE As String

   DoCmd.OpenReport "Umsätze nach Kategorie", acViewDesign
   Set Bericht = Reports("Umsätze nach Kategorie")
```

Listing 413: Berichtselemente identifizieren

```
    For Each obj In Bericht.Controls
       Select Case obj.ControlType
          Case 100
             strE = "Beschriftung"
          Case 101
             strE = "Rechteck"
          Case 102
             strE = "Linie"
          Case 103
             strE = "Bild"
          Case 104
             strE = "Schaltfläche"
          Case 105
             strE = "Optionsschaltfläche"
          Case 106
             strE = "Kontrollkästchen"
          Case 107
             strE = "Gruppenfeld"
          Case 109
             strE = "Textfeld"
          Case 110
             strE = "Listenfeld"
          Case 111
             strE = "Kombinationsfeld"
           Case 112
             strE = "Unterbericht"
          Case 114
             strE = "OLE-Objekt - ungebunden"
          Case 118
             strE = "Seitenumbruch"
          Case 122
             strE = "Umschaltfläche"
          Case 123
             strE = "Registersteuerelement"
          Case Else
            strE = ""
       End Select
       Debug.Print "Typ: " & obj.ControlType & _
          "=" & strE & vbLf & "Name: " & obj.Name & vbLf
    Next obj
    DoCmd.Close
 End Sub
```

Listing 413: Berichtselemente identifizieren (Forts.)

Sammeln Sie in einer `Select Case`-Anweisung die verschiedenen Indizes der Steuerelemente und weisen Sie über die Variable `strE` einen Text zu. Diesen geben Sie dann im Direktbereich aus.

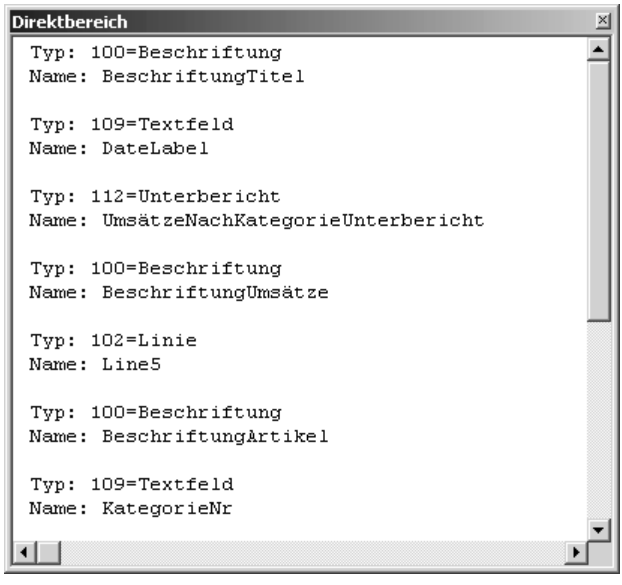


Abbildung 260: Die einzelnen Berichtselemente wurden identifiziert.

340 Berichtsteil ein- und ausblenden

Über den Einsatz der Eigenschaft `Visible` können Sie einzelne Teile eines Berichts bzw. auch Steuerelemente ein- und ausblenden, indem Sie diese Eigenschaft auf den Wert `True` bzw. `False` setzen.

Wie kann man nun einen bestimmten Teil eines Berichts ausblenden?

Ein Bericht ist standardmäßig in verschiedene Sektionen unterteilt. Mithilfe der Eigenschaft `Section` können Sie daher ganz gezielt bestimmte Teile des Berichts ansprechen. Die Berichtsteile können entweder über einen Index oder über eine Konstante angesprochen werden. Entnehmen Sie aus der folgenden Tabelle die dabei möglichen Werte.

Index	Konstante	Beschreibung
0	`acDetail`	Detailbereich
1	`acHeader`	Berichtskopf
2	`acFooter`	Berichtsfuß
3	`acPageHeader`	Seitenkopfbereich
4	`acPageFooter`	Seitenfußbereich
5	`acGroupLevel1Header`	Kopfbereich der Gruppenebene 1
6	`acGroupLevel1Footer`	Fußbereich der Gruppenebene 1
7	`acGroupLevel2Header`	Kopfbereich der Gruppenebene 2
8	`acGroupLevel2Footer`	Fußbereich der Gruppenebene 2

Tabelle 132: Die Berichtssegmente

Das folgende Beispiel aus Listing 414 blendet beispielsweise den Detailbereich beim Öffnen des Formulars aus. Dabei soll nur die Struktur eines Berichts, ohne Daten, angezeigt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Klassen    Report_Alphabetische Artikelliste
'=====================================================

Private Sub Report_Open(Cancel As Integer)
   Me.Section(acDetail).Visible = False
End Sub
```

Listing 414: Bestimmten Teil des Berichts ausblenden

Das Ereignis `Report_Open` tritt automatisch dann ein, wenn der Bericht geöffnet wird. Dies ist der ideale Zeitpunkt, um einzelne Sektionen ein- bzw. auszublenden. Eine Sektion können Sie ausblenden, indem Sie die Eigenschaft `Visible` auf den Wert `False` setzen.

341 Eigene Berichte erstellen

Um einen Bericht mithilfe von VBA anzulegen, setzen Sie die Methode `CreateReport` ein. Die Syntax dieser Methode lautet:

```
Ausdruck.CreateReport(Datenbank, Berichtsvorlage)
```

Argument	Beschreibung
`Ausdruck`	Erforderlich. Gibt ein `Application`-Objekt zurück.
`Datenbank`	Optional. Gibt an, aus welcher Datenbank die Berichtsvorlage für den neuen Bericht verwendet werden soll. Lassen Sie dieses Argument komplett weg, wenn Sie einen leeren Bericht ohne Vorlage erstellen möchten.
`Berichtsvorlage`	Optional. Enthält den Namen des Berichts, auf dessen Basis Sie den neuen Bericht erstellen möchten. Wenn Sie dieses Argument weglassen, legt Microsoft Access dem neuen Bericht die Vorlage zugrunde, die im Dialogfeld OPTIONEN auf der Registerkarte FORMULARE/BERICHTE angegeben ist.

Tabelle 133: Die Argumente der Methode CreateReport

Im folgenden Beispiel aus Listing 415 wird ein leerer Bericht auf Basis der Tabelle KATEGORIEN erzeugt.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'====================================================

Sub NeuenBerichtEinfügen()
   Dim Bericht As Report
   Dim strName As String

   Set Bericht = CreateReport
   Bericht.RecordSource = "Kategorien"
   strName = Bericht.Name
   DoCmd.Close acReport, strName, acSaveYes
End Sub
```

Listing 415: Einen neuen Bericht erzeugen

Definieren Sie im ersten Schritt eine Objektvariable vom Typ `Report`. Damit haben Sie Zugriff auf alle berichtsbezogenen Methoden und Eigenschaften. Um einen neuen Bericht anzulegen, setzen Sie die Methode `CreateReport` ein. Mithilfe der Eigenschaft `RecordSource` geben Sie die Tabelle an, auf die sich der Bericht beziehen soll.

Die Eigenschaft `Name` gibt den Namen des Berichts aus, den Sie verwenden, um den Bericht zu speichern. Schließen Sie den noch geöffneten Bericht mithilfe der Methode `Close`.

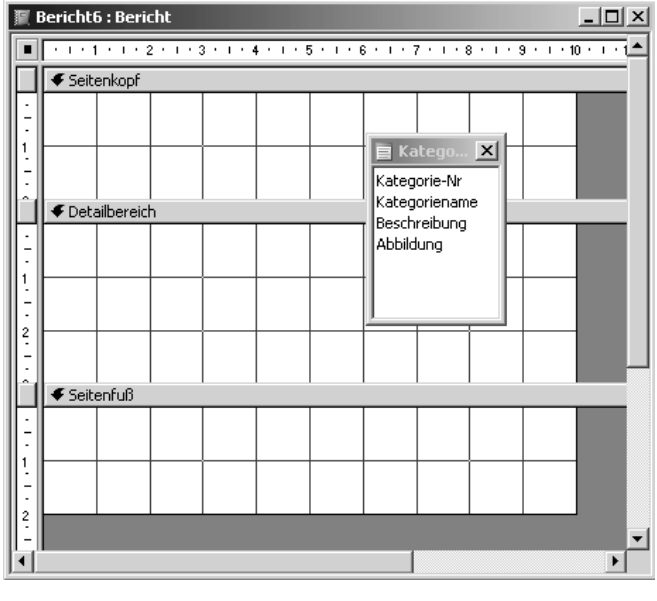


Abbildung 261: Einen neuen, noch leeren Bericht erzeugen

342 Berichtselemente einfügen

Über die Methode `CreateReportControl` können Sie Steuerelemente in einen Bericht einfügen. Die Syntax dieser Methode lautet:

```
CreateReportControl(Berichtsname, Steuerelementtyp, Bereich, Übergeordnet,
Spaltenname, Links, Oben, Breite, Höhe)
```

Argument	Beschreibung
`Berichtsname`	Erforderlich. Gibt den Namen des Berichts an, in den Sie die Steuerelemente einfügen möchten.
`Steuerelementtyp`	Erforderlich. Bestimmt die Art des Steuerelements. `acBoundObjectFrame`: gebundenes Objektfeld `acCheckBox`: Kontrollkästchen `acComboBox`: Kombinationsfeldliste `acCommandButton`: Befehlsschaltfläche `acCustomControl`: Zusatzsteuerelement `acImage`: Bild `acLabel`: Bezeichnungsfeld `acLine`: Linie `acListBox`: Listenfeld `acObjectFrame`: ungebundenes Objektfeld `acOptionButton`: Optionsschaltfläche `acOptionGroup`: Gruppenfeld `acPage`: Seitenwechsel `acPageBreak`: Seitenumbruch `acRectangle`: Rechteck `acSubform`: Unterbericht `acTabCtl`: Registersteuerelement `acTextBox`: Textfeld `acToggleButton`: Umschaltfläche

Tabelle 134: Die Argumente der Methode CreateReportControl

Argument	Beschreibung
Bereich	Optional. Gibt an, wo das Steuerelement genau eingefügt werden soll. Auch hierfür gibt es vorgefertigte Konstanten: acDetail: Bei dieser Standardeinstellung wird das Steuerelement im Detailbereich des Berichts eingefügt. acFooter: Das Steuerelement wird im Berichtsfuß eingefügt. acGroupLevel1Footer: Gruppenfuß 1 acGroupLevel1Header: Gruppenkopf 1 acGroupLevel2Footer: Gruppenfuß 2 acGroupLevel2Header: Gruppenkopf 2 acHeader: Berichtskopf acPageFooter: Seitenfuß acPageHeader: Seitenkopf
Übergeordnet	Optional. Gibt den Namen des übergeordneten Steuerelements eines zugeordneten Steuerelements an. Bei Steuerelementen ohne übergeordnetes Steuerelement verwenden Sie eine leere Zeichenfolge für dieses Argument oder lassen es weg.
Spaltennamen	Optional. Gibt den Feldnamen des verknüpften Felds der Tabelle an, sofern es sich um ein gebundenes Steuerelement handelt. Möchten Sie keine Verknüpfung zu einem Tabellenfeld herstellen, dann geben Sie dieses Argument nicht an.
Links, Oben, Breite und Höhe	Optional. Damit können Sie die genaue Position sowie dessen Abmessung einstellen. Die Einheit für diese Positionierung wird in Twips ausgegeben. Dabei entsprechen 567 Twips einem Zentimeter.

Tabelle 134: Die Argumente der Methode CreateReportControl (Forts.)

Im folgenden Beispiel aus Listing 416 wird ein neuer Bericht auf Basis der Tabelle Kategorien erstellt und mit einem Text sowie einem Bezeichnungsfeld angelegt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub BerichtselementeEinfügen()
   Dim Bericht As Report
   Dim ctrlBez As Control
   Dim ctrlText As Control
```

Listing 416: Einen Bericht mit Feldern füllen

```
    Set Bericht = CreateReport
    Bericht.RecordSource = "Kategorien"

    Set ctrlText = CreateReportControl(Bericht.Name, acTextBox, _
    acDetail, "", "Kategoriename", 1500, 50, 1200, 200)
    Set ctrlBez = CreateReportControl _
       (Bericht.Name, acLabel, , Text.Name, "Kategoriename", 150, 50, 800, 200)
    DoCmd.Close acReport, Bericht.Name, acSaveYes
End Sub
```

Listing 416: Einen Bericht mit Feldern füllen (Forts.)

Möchten Sie einen Bericht noch schneller und komfortabler erstellen, dann können Sie dies auch über den Einsatz von ADO machen. Im folgenden Beispiel aus Listing 417 wird ein Bericht auf Basis der Tabelle `Kunden` erstellt und alle dazu benötigten Steuerelemente werden automatisch in den Bericht eingefügt und verknüpft.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub BerichtErstellen()
   Dim Conn As New ADODB.Connection
   Dim Dbs As ADODB.Recordset
   Dim Datenfeld As ADODB.Field
   Dim ctrlTextfeld As Control
   Dim ctrlBezfeld As Control
   Dim lngPosOben As Long
   Dim lngPosLinks As Long

   Set Bericht = CreateReport
   lngPosLinks = 0
   lngPosOben = 0
   Bericht.Report.RecordSource = "Kunden"
   Set Conn = CurrentProject.Connection
   Set Dbs = New ADODB.Recordset
   Dbs.Open "SELECT * FROM Kunden where (Land = 'Deutschland')", _
      Conn, adOpenKeyset, adLockOptimistic

   For Each Datenfeld In Dbs.Fields
      Set ctrlTextfeld = CreateReportControl(Bericht.Report.Name, acTextBox, _
         acDetail, , Datenfeld.Name, lngPosLinks + 1500, lngPosOben)
      ctrlTextfeld.SizeToFit
      Set ctrlBezfeld = CreateReportControl(Bericht.Report.Name, acLabel, _
         acDetail, ctrlTextfeld.Name, Datenfeld.Name, _
         lngPosLinks, lngPosOben, 1400, ctrlTextfeld.Height)
      ctrlBezfeld.SizeToFit
```

Listing 417: Die vollautomatische Berichtserstellung

```
        lngPosOben = lngPosOben + ctrlTextfeld.Height + 50
    Next Datenfeld

    Dbs.Close
    Set Dbs = Nothing
End Sub
```

Listing 417: Die vollautomatische Berichtserstellung (Forts.)

Erstellen Sie im ersten Schritt über die Methode `CreateReport` einen neuen, noch leeren Bericht. Legen Sie danach die Koordinaten für das erste Bezeichnungsfeld durch die Variablen `lngPosOben` und `lngPosLinks` fest. Diese Festlegung der Positionen führen Sie ganz bewusst über Variablen schon am Beginn des Makros durch, damit Sie später diese nur hoch addieren müssen, um zur jeweils nächsten Einfügeposition zu gelangen.

Mithilfe der Eigenschaft `RecordSource` geben Sie bekannt, dass der Bericht auf die Tabelle `Kunden` zugreifen soll. Stellen Sie danach die Verbindung zur Datenquelle her und erstellen Sie ein neues `Recordset`-Objekt.

Führen Sie nach dem Öffnen der Tabelle `Kunden` eine SQL-Abfrage durch, die alle Kunden aus Deutschland im `Recordset`-Objekt `DBS` speichert. Über das Auflistungsobjekt `Fields` können Sie nun alle Datenfelder in einer Schleife abarbeiten. Für jedes Datenfeld wird sowohl ein Textfeld als auch ein Bezeichnungsfeld angelegt. Das Bezeichnungsfeld erhält dabei automatisch die Beschriftung des Datenfelds.

Mithilfe der Methode `SizeTofit` können Sie einen weiteren Automatismus nützen. Diese Eigenschaft passt das angegebene Steuerelement automatisch an den Text an, den das Steuerelement zugewiesen bekommt.

Damit die Einfügeposition der neuen Steuerelemente am Ende der Schleife verschoben wird, addieren Sie zur Variablen `lngPosOben` den Wert 1500. Beim nächsten Schleifendurchlauf wird dadurch die neue Einfügeposition festgelegt.

343 Datenzugriffsseiten

Die Datenzugriffsseiten erstellen Sie wie ganz normale Formulare, die Sie über den Assistenten erzeugen.

Befolgen Sie nun die nächsten Arbeitsschritte, um eine Datenzugriffsseite zu erstellen.

1. Klicken Sie in der Datenbankleiste auf den Eintrag `Seiten`.
2. Führen Sie danach einen Doppelklick auf den Befehl ERSTELLT EINE DATENZUGRIFFSSEITE UNTER VERWENDUNG DES ASSISTENTEN aus.
3. Wählen Sie im Drop-down-Feld TABELLEN/ABFRAGEN die Quelle für Ihr Online-Formular aus.

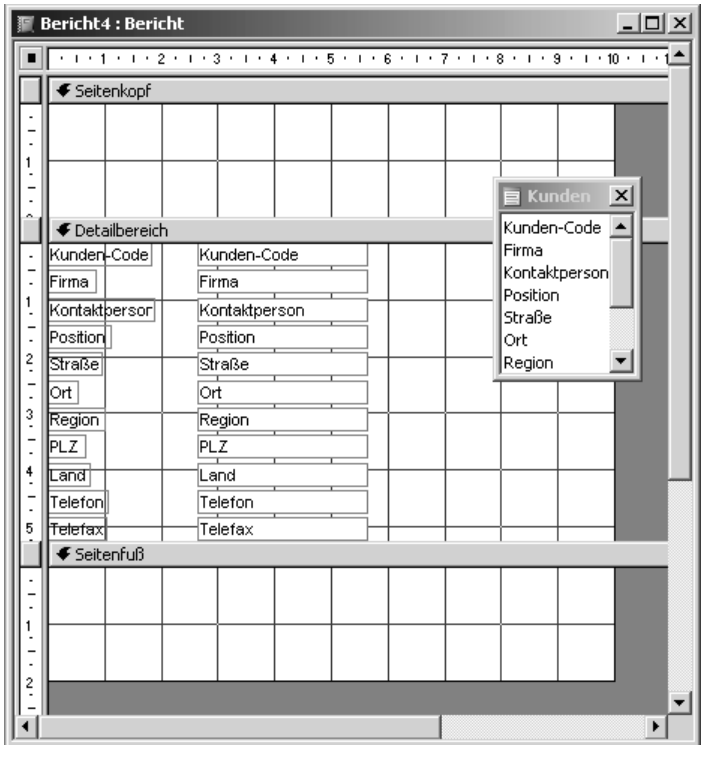


Abbildung 262: Der Bericht wurde angelegt.

4. Entscheiden Sie sich für die Felder, die in der Datenzugriffsseite angezeigt werden sollen. Übertragen Sie die gewünschten Felder vom linken in das rechte Listenfeld, indem Sie die Schaltflächen einsetzen.
5. Klicken Sie danach auf die Schaltfläche WEITER.
6. Wählen Sie bei Bedarf eine Gruppierungsebene aus. Betrachten Sie die Auswirkung der Gruppierung im Vorschaufenster.
7. Klicken Sie auf die Schaltfläche WEITER, um zum nächsten Schritt des Assistenten zu gelangen.
8. Bestimmen Sie die Sortierreihenfolge der Datenzugriffsseite, indem Sie das gewünschte Sortierfeld aus den Drop-down-Listen auswählen und mithilfe der Umschaltfläche AUFSTEIGEND bzw. ABSTEIGEND die Sortierung festlegen.
9. Klicken Sie auf WEITER.
10. Geben Sie der Zugriffsseite einen Namen.
11. Aktivieren Sie die Option SEITE ÖFFNEN, damit das Resultat sofort angezeigt wird.
12. Klicken Sie auf die Schaltfläche FERTIG STELLEN, um die Datenzugriffsseite zu erzeugen.

Abbildung 263: Datenzugriffsseite erstellen

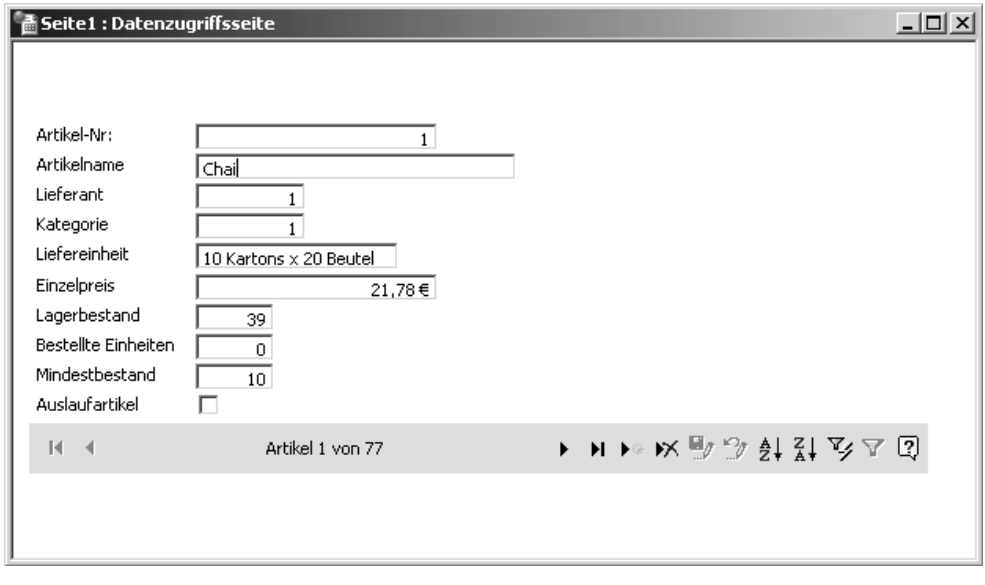


Abbildung 264: Die fertige Datenzugriffsseite

Die Datenzugriffsseite wird standardmäßig als HTML-Datei im selben Verzeichnis gespeichert wie die aktuelle Datenbank.

Das Erstellen einer Datenzugriffsseite kann auch über das Makro aus Listing 418 erledigt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseiteErzeugen()
   Dim dapSeite As DataAccessPage

   Set dapSeite = Application.CreateDataAccessPage("Artikelbestand", True)
   dapSeite.ApplyTheme "citrus"
   DoCmd.Close acDataAccessPage, dapSeite.Name, acSaveYes
End Sub
```

Listing 418: Datenzugriffsseite erstellen

Mithilfe der Methode `CreateDataAccessPage` erstellen Sie eine Datenzugriffsseite. Mit der Methode `ApplyTheme` können Sie das Microsoft Office-Design für eine angegebene Datenzugriffsseite festlegen. Dazu müssen Sie die standardmäßig angebotenen Designs natürlich kennen. Ein paar davon können Sie der folgenden Tabelle entnehmen.

Design	Beschreibung
artsy	Ein Design mit einem leichten schwarzen Hintergrund.
blank	Dieses Design hat einen leeren Hintergrund.
blends	Bei diesem Design wird am linken Rand ein bläulicher Balken angezeigt.
BluePrnt	Dieses Design beinhaltet einen weißen Hintergrund und kleine hellblaue Quadrate.
BoldStri	Bei diesem Design ist die Datenzugriffsseite vertikal liniert.
CapSules	Dieses Design gibt die Datenzugriffsseite mit weißem Hintergrund und formschönen grünlichen Elementen aus.
Citrus	Bei diesem Design wird ein frühlingshaftes Muster verwendet.
expeditn	Dieses Design gibt die Datenzugriffsseite in einem sandfarbenen Muster aus.
indust	Dieses Design belegt die Datenzugriffsseite mit einem gleichmäßigen grauen Muster auf weißem Hintergrund.
ricepapr	Dieses Design weist der Datenzugriffsseite eine reisförmige Musterung auf olivgrünem Hintergrund zu.
rmnsque	Bei diesem Design wird der Datenzugriffsseite ein gepunktetes Muster auf hellgrauem Hintergrund zugewiesen.
strtedge	Dieses Design erzeugt eine ganz feine horizontale Linierung auf weißem Grund.
sumipntg	Bei diesem Design wird ein marmorähnlicher Effekt auf der Datenzugriffsseite erzeugt.

Tabelle 135: Mögliche Designkonstanten für Datenzugriffsseiten

344 Datenzugriffsseiten auslesen

Um zu ermitteln, welche Datenzugriffsseiten in Ihrer aktuellen Datenbank vorhanden sind, können Sie das Makro aus Listing 419 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseitenAusgeben()
   Dim dap As AccessObject

   Set Conn = Application.CurrentProject
   For Each dap In Conn.AllDataAccessPages
      Debug.Print dap.Name
   Next dap
End Sub
```

Listing 419: Alle Datenzugriffsseiten dokumentieren

Über das Objekt `CurrentProject` kommen Sie an alle Access-Objekte der aktuellen Datenbank heran. So haben Sie auch Zugriff auf das Auflistungsobjekt `AllDataAccessPages`, in dem alle Datenzugriffsseiten der aktuellen Datenbank verzeichnet sind. Diese geben Sie über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

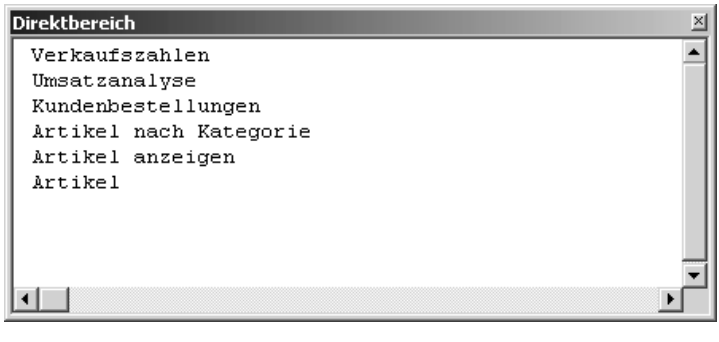


Abbildung 265: Alle Datenzugriffsseiten wurden ermittelt.

345 Speicherort der Datenzugriffsseiten ermitteln

Um herauszufinden, wo genau sich die Datenzugriffsseiten auf Ihrer Festplatte befinden, können Sie das Makro aus Listing 420 einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseitenQuellenAusgeben()
   Dim dap As AccessObject

   Set Conn = Application.CurrentProject

   For Each dap In Conn.AllDataAccessPages
      Debug.Print "Die Datenzugriffsseite:'" & _
         dap.Name & vbLf & "' ist gespeichert unter: " & _
         vbLf & dap.FullName & vbLf
   Next dap
End Sub
```

Listing 420: Die Speicherorte von Datenzugriffsseiten finden

Mit der Eigenschaft `FullName` können Sie den vollständigen Pfad (einschließlich des Dateinamens) der Datenzugriffsseite ermitteln.

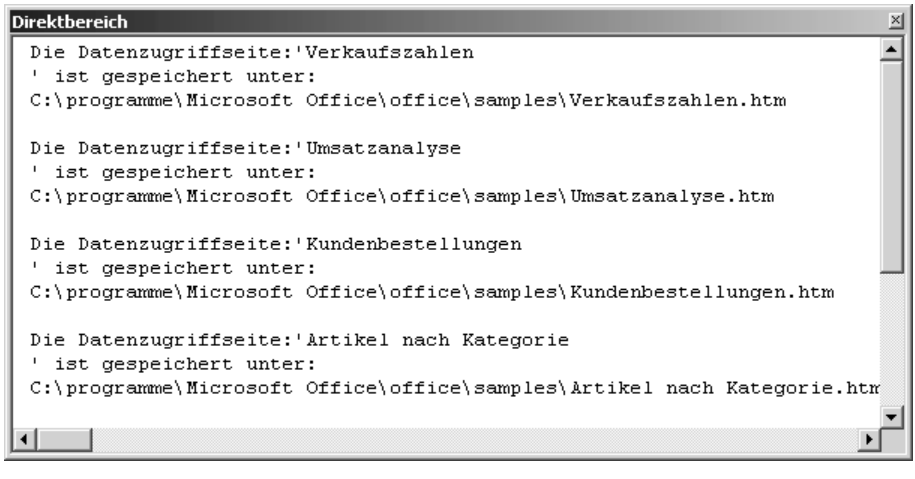


Abbildung 266: Die Speicherpfade der Datenzugriffsseiten werden ausgelesen.

346 Datenzugriffsseiten anzeigen

Um eine Datenzugriffsseite anzuzeigen können Sie die Methode `OpenDataAccessPage` einsetzen. Diese Methode hat folgende Syntax:

```
OpenDataAccessPage(DatenzugriffsseitenName, Ansicht)
```

Im Argument `DatenzugriffsseitenName` geben Sie den Namen der Seite an, die Sie öffnen möchten.

Im Argument `Ansicht` können Sie mithilfe einer Konstanten entscheiden, wie Sie diese Datenzugriffsseite öffnen möchten. Es stehen dabei folgende Konstanten zur Verfügung:

- `acDataAccessPageBrowse`: Bei dieser Standardeinstellung wird die Datenzugriffsseite in der Seitenansicht von Access geöffnet.
- `acDataAccessPageDesign`: Die Datenzugriffsseite wird in der Entwurfsansicht geöffnet.

Im folgenden Beispiel aus Listing 421 wird die Datenzugriffsseite ARTIKEL geöffnet.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=================================================

Sub DatenzugriffsseiteÖffnen()
   DoCmd.OpenDataAccessPage "Artikel", acDataAccessPageBrowse
End Sub
```

Listing 421: Datenzugriffsseite öffnen

347 Datenzugriffsseite anpassen

Über VBA-Makros können Sie jederzeit bestehende Datenzugriffsseiten anpassen und wieder speichern. Im folgenden Makro aus Listing 422 wird eine Datenzugriffsseite geöffnet, ein bestimmtes Steuerelement angesprungen und verändert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap07
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=================================================

Sub DatenzugriffsseiteÖffnenUndÄndern()
   Const Dap = "Alphabetische Artikelliste"

   With DoCmd
      .Echo False
      .OpenDataAccessPage Dap, acDataAccessPageDesign

      With DataAccessPages(Dap)
         .ApplyTheme "expeditn"
         .Document.all("BezeichnungsfeldO").InnerText = _
            "Heute ist " & Format(Date, "dd.mm.yyyy")
      End With

      .Close acDataAccessPage, Dap, acSaveYes
      .OpenDataAccessPage Dap, acDataAccessPageBrowse
```

Listing 422: Datenzugriffsseite anpassen

```
        .Echo True
    End With
End Sub
```

Listing 422: Datenzugriffsseite anpassen (Forts.)

Beim Ausführen eines Makros in Microsoft Access werden durch Bildschirmaktualisierungen oft Informationen angezeigt, die für die Funktionalität des Makros ohne Bedeutung sind. Wenn Sie das Argument `Echo` auf den Wert `False` setzen, wird das Makro ohne Bildschirmaktualisierung ausgeführt. Beim Beenden des Makros schaltet Microsoft Access automatisch `Echo` wieder ein und aktualisiert das Fenster.

Öffnen Sie die Datenzugriffsseite im Entwurfsmodus mithilfe der Methode `OpenDataAccessPage`, indem Sie die Konstante `acDataAcessPageDesign` einsetzen. Führen Sie danach die Methode `ApplyTheme` aus, um der Datenzugriffsseite ein neues Design zuzuweisen. Danach greifen Sie auf das Bezeichnungsfeld zu und legen über die Eigenschaft `InnerText` den Text des Felds fest. Dabei geben Sie das aktuelle Datum aus. Schließen Sie die Datenzugriffsseite und speichern Sie dabei die Änderungen, indem Sie der Methode `Close` den Namen der Seite sowie die Konstante `acSaveYes` übergeben. Öffnen Sie danach gleich im Anschluss die Datenzugriffsseite in der Seitenansicht.

Die Ereignisprogrammierung

In Access gibt es jede Menge Ereignisse, die Sie für die Programmierung einsetzen können. Diesen Ereignissen lassen sich dann zusätzliche Befehle zuweisen. So können Sie beispielsweise beim Starten eines Formulars weitere Aktionen wie die Vorbelegung bestimmter Felder vornehmen, die im Dialog angeboten werden sollen.

In diesem Kapitel können Sie die wichtigsten Ereignisse nachschlagen.

348 Zugangsverwaltung in Access (Formular)

Das Ereignis `Open` tritt dann ein, wenn Sie ein Formular oder einen Bericht aufrufen. Damit ist genau der Zeitpunkt gemeint, bevor der erste Datensatz angezeigt wird. Dieses Ereignis können Sie einsetzen, um Initialisierungsarbeiten im Formular/Bericht bzw. Vorbelegungen einiger Steuerelemente vorzunehmen.

Im folgenden Beispiel richten Sie über das Ereignis `Open` eine Art Zugangsschutz für ein Formular ein. Dabei können Sie beim Öffnen eines Formulars über eine API-Funktion den Namen des Anwenders abfragen und in Abhängigkeit davon entweder den Zugang zulassen oder das Formular wieder schließen. Um diese Aufgabe umzusetzen, legen Sie ein neues Makro auf Modulebene an, das Sie in Listing 423 sehen können.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Modul      mdlAPI
'=================================================

Declare Function GetUserName Lib "advapi32.dll" Alias _
   "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

Listing 423: Den Anwendernamen auslesen

Stellen Sie jetzt das Ereignis `Open` im Formular `Personal` ein, indem Sie folgende Arbeitsschritte befolgen:

1. Gehen Sie in die Entwurfsansicht des Formulars, in dem Sie das Ereignis einstellen möchten.
2. Klicken Sie mit der rechten Maustaste auf eine freie Stelle des Formulars und wählen Sie den Befehl EIGENSCHAFTEN aus dem Kontextmenü.
3. Wechseln Sie auf die Registerkarte EREIGNIS.

4. Wählen Sie aus dem Drop-down-Menü den Befehl FORMULAR.

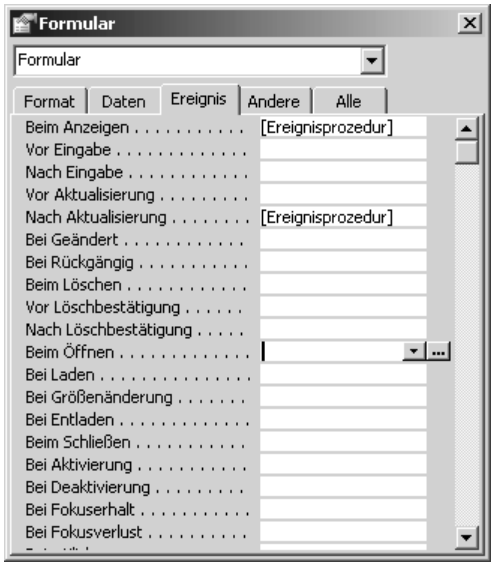


Abbildung 267: Ein Ereignis einstellen

5. Setzen Sie den Mauszeiger in die Zeile BEIM ÖFFNEN und klicken Sie auf das Symbol mit den drei Punkten ganz rechts in der Zeile.

6. Im Dialogfeld GENERATOR AUSWÄHLEN wählen Sie den Eintrag CODE-GENERATOR und bestätigen mit OK.

7. Erfassen Sie dann das folgende Ereignismakro aus Listing 424.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Form_Open(Cancel As Integer)
   Dim strID As String
   Dim lngSize As Long
   Dim lngAns As Long

   lngSize = 8
   strID = "       "
   lngAns = GetUserName(strID, lngSize)
   strID = Trim(strID)
   If strID = "Held" Or strID = "admin" Then
```

Listing 424: Die Benutzerverwaltung über ein Ereignismakro einstellen

```
        Me.TabCtl0.Pages(1).Visible = True
    Else
        Me.TabCtl0.Pages(1).Visible = False
    End If
End Sub
```

Listing 424: Die Benutzerverwaltung über ein Ereignismakro einstellen (Forts.)

Über die API-Funktion `GetUser` können Sie den angemeldeten Benutzer ermitteln. In einer anschließenden Abfrage stellen Sie fest, ob der berechtigte Anwender `Held` bzw. `admin` versucht, das Formular `Personal` zu öffnen. Wenn es sich dabei um einen anderen Anwender handelt, dann darf nur das erste Registerblatt des Formulars angezeigt werden. Das erste Registerblatt wird über die Nummer 0, das zweite durch die Nummer 1 repräsentiert. Blenden Sie daher das zweite Registerblatt aus, indem Sie die Eigenschaft `Visible` auf den Wert `False` setzen.

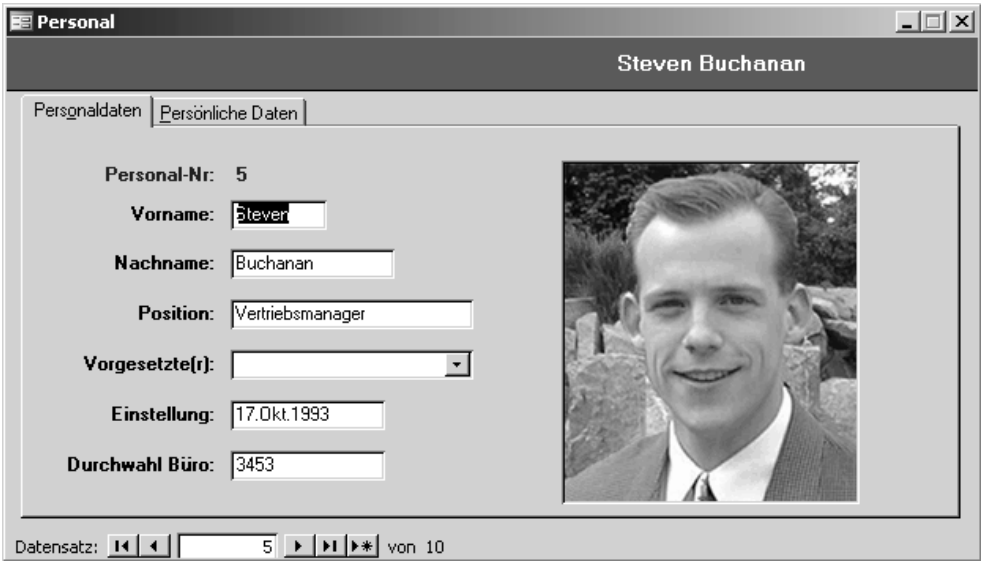


Abbildung 268: Als zugelassener Anwender dürfen Sie beide Registerblätter einsehen.

349 Zugangsverwaltung in Access (Bericht)

Auch für einen Bericht können Sie das Ereignis `Open` einsetzen. Dabei wird im folgenden Beispiel aus Listing 425 das Ereignis `Report_Open` im `Bericht5` eingestellt und dabei geprüft, ob der Anwender den Bericht überhaupt anzeigen darf. Die Überprüfung des Anwenders wird in diesem Beispiel alternativ zur letzten Aufgabe nicht über eine API-Funktion `GetUserName`, sondern über die Funktion `Environ` überprüft.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Report_Bericht5
'=================================================

Private Sub Report_Open(Cancel As Integer)
   Select Case Environ("Username")
      Case "Held", "Mueller", "Schmitt"
         'Zugriff erlaubt
      Case Else
         'Zugriff verweigert
         MsgBox "Keine Berechtigung diesen Bericht auszuführen!"
         Cancel = True
   End Select
End Sub
```

Listing 425: Zugangsberechtigung bei Berichten prüfen

Über eine `Select Case`-Anweisung überprüfen Sie mittels der Funktion `Environ`, der Sie als Konstante `Username` übergeben, welcher Anwender gerade versucht, den Bericht zu öffnen. Im Beispiel wird das Öffnen des Berichts nur den drei genannten Personen gestattet. Allen anderen wird der Zugang zum Bericht verweigert, indem das Argument `Cancel` auf den Wert `True` gesetzt wird. Damit wird das Ereignis automatisch abgebrochen, noch bevor es zum Einsatz kommt.

350 Bestimmtes Feld fokussieren

Im folgenden Beispiel setzen Sie das Ereignis `Open` ein, um nach dem Anzeigen eines Formulars den Mauszeiger in ein bestimmtes Datenfeld zu setzen. So setzen Sie im Listing 426 den Mauszeiger im Formular LIEFERANTEN nach dem Öffnen in das Feld TELEFON.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=================================================

Private Sub Form_Open(Cancel As Integer)
   Me.Telefon.SetFocus
End Sub
```

Listing 426: Den Fokus in ein bestimmtes Feld setzen

Mithilfe des Schlüsselworts `Me` haben Sie Zugriff auf alle Steuerelemente des aktiven Formulars. So können Sie auch das Steuerelement TELEFON ansprechen und über die Methode `SetFocus` den Mauszeiger in das Feld hineinsetzen.

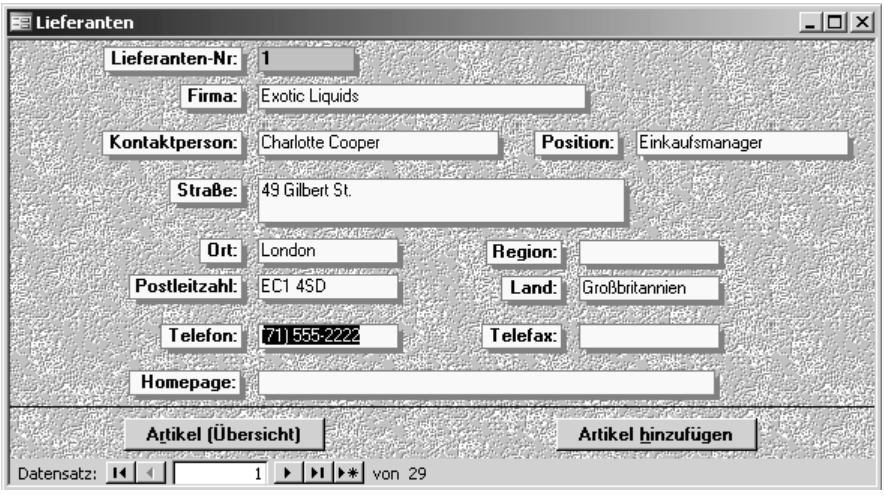


Abbildung 269: Der Fokus befindet sich nach dem Öffnen des Formulars im Feld Telefon.

351 Sicherheitscheck durchführen

Haben Sie ein Formular (`Artikel3`) mit einer Tabelle (`Artikel3`) verknüpft und möchten Sie beim Öffnen prüfen, ob sich überhaupt Datensätze in der verknüpften Tabelle befinden, setzen Sie das Makro aus Listing 427 für das Formular `Artikel3` ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel3
'=====================================================

Private Sub Form_Open(Cancel As Integer)
   Dim conn As Database
   Dim dbs As Recordset

   Set conn = CurrentDb
   Set dbs = conn.OpenRecordset("Artikel3", dbOpenDynaset)

   If dbs.RecordCount = 0 Then
      MsgBox "Es sind keine Daten in der Tabelle " & dbs.Name
     Cancel = True
   End If

   dbs.Close
   Set conn = Nothing
End Sub
```

Listing 427: Vor dem Öffnen prüfen, ob Daten bereitstehen

Mithilfe der Methode `CurrentDB` ermitteln Sie den Namen der momentan geöffneten Datenbank. Öffnen Sie danach die verknüpfte Tabelle und überprüfen Sie, ob in der Tabelle überhaupt Datensätze vorhanden sind. Ist dies der Fall, meldet die Eigenschaft `RecordCount` einen Wert > 0. In diesem Fall können Sie das Ereignis weiter fortführen. Im anderen Fall geben Sie eine Meldung auf dem Bildschirm aus und setzen das Argument `Cancel` auf den Wert `False`, um den Öffnen-Vorgang des Formulars abzubrechen.

352 Alle beteiligten Elemente schließen

Sicherheitshalber können Sie beim Schließen eines Formulars eventuell geöffnete Tabellen und Formulare automatisch schließen, indem Sie das Ereignis `Form_Close` einsetzen. Das folgende Beispiel aus Listing 428 schließt nach dem Beenden des Formulars `Lieferanten` die Formulare `Artikel` und `Artikelliste`, sofern diese noch geöffnet sind.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Modul      Dienstprogramme
'=====================================================

Function IsLoaded(ByVal strFormName As String) As Boolean
   Const conObjStateClosed = 0
   Const conDesignView = 0

   If SysCmd(acSysCmdGetObjectState, _
      acForm, strFormName) <> conObjStateClosed Then
      If Forms(strFormName).CurrentView <> conDesignView Then
         IsLoaded = True
      End If
   End If
End Function


'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Form_Close()
   If IsLoaded("Artikelliste") Then DoCmd.Close acForm, "Artikelliste"
   If IsLoaded("Artikel") Then DoCmd.Close acForm, "Artikel"
End Sub
```

Listing 428: Geöffnete Tabellen werden mit geschlossen.

Mithilfe der Eigenschaft `IsLoaded` (ursprünglich aus der Microsoft-Beispieldatenbank *Nordwind.mdb*) können Sie überprüfen, ob ein bestimmtes Access-Objekt noch geöffnet ist.

Wenn ja, dann wird der Wert `True` gemeldet. In diesem Fall wenden Sie die Methode `Close` an, um ein Formular zu schließen.

353 Nach fünf Sekunden automatisch schließen

Stellen Sie sich vor, Sie haben die Möglichkeit, mithilfe von Ereignissen einen schönen Countdown zu programmieren. So programmieren Sie im nächsten Beispiel das Formular `Lieferanten` so, dass es nach ca. fünf Sekunden automatisch das dazugehörige Formular `Artikel` schließt.

Diesen Countdown stellen Sie ein, indem Sie beim Schließen des Formulars `Lieferanten` »den Zeitzünder scharf machen«. Dabei geben Sie folgendes Ereignis aus Listing 429 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Sub Form_Close()
   Form_Artikel.TimerInterval = 5000
End Sub
```

Listing 429: Den Schließ-Verzögerer setzen

Mit der Eigenschaft `TimerInterval` können Sie in Millisekunden die Länge des Intervalls angeben, das zwischen zwei Timer-Ereignissen eines Formulars liegen soll.

Stellen Sie nun auf der anderen Seite im Formular `Artikel` das Ereignis aus Listing 430 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Sub Form_Timer()
   Me.TimerInterval = 0
   MsgBox "Zeit abgelaufen!"
   DoCmd.Close acForm, "Artikel"
End Sub
```

Listing 430: Das Formular Artikel wird nach fünf Sekunden geschlossen.

Setzen Sie die Eigenschaft `TimerInterval` auf den Wert 0. Rufen Sie danach als Test beide Formulare auf und beenden Sie dann das Formular `Lieferanten`. Gedulden Sie sich fünf Sekunden und sehen Sie dabei zu, wie auch das Formular `Artikel` automatisch geschlossen wird.

354 Zugriff dokumentieren

In der folgenden Lösung wird der Zugriff auf den `Bericht5` in der Tabelle `Zugriffe` festgehalten. Dabei soll beim Schließen des Berichts ein Datensatz in die Tabelle `Zugriffe` geschrieben werden, aus dem hervorgeht, wer und wann jemand den Bericht verwendet hat. Legen Sie dazu eine Datentabelle gemäß folgender Abbildung 270 an.

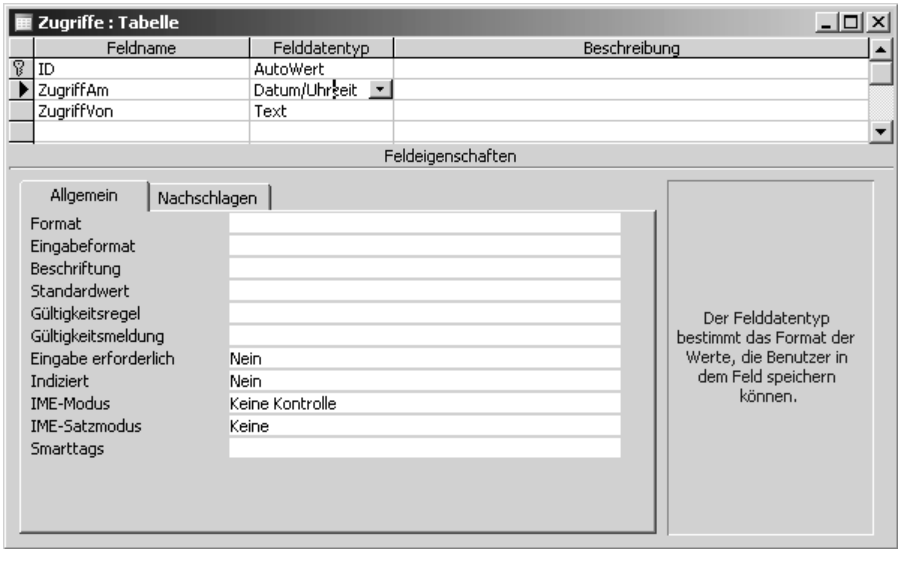


Abbildung 270: Die Tabelle für die Zugriffsüberwachung

Stellen Sie jetzt das Ereignis `Report_Close` ein und ergänzen Sie den Ereignisrahmen wie in Listing 431 angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Report_Bericht5
'=====================================================

Private Sub Report_Close()
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset

   With DBS
      .Open "Zugriffe", Conn, adOpenKeyset, adLockOptimistic
```

Listing 431: Jeder Zugriff auf den Bericht wird festgehalten.

```
        .AddNew
        .Fields("ZugriffAm") = Now
        .Fields("ZugriffVon") = Environ("Username")
        .Update
        .Close
    End With

    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 431: Jeder Zugriff auf den Bericht wird festgehalten. (Forts.)

Öffnen Sie zuerst die Tabelle über die Methode `Open`. Über die Methode `AddNew` wird ein neuer, noch leerer Datensatz in die Tabelle geschrieben. Danach werden die Felder über die Auflistung `Fields` gefüllt. Über den Einsatz der Methode `Update` wird der Datensatz gesichert. Die Methode `Close` schließt die Tabelle wieder.

Abbildung 271: Alle Zugriffe werden im Hintergrund dokumentiert.

355 Listenfeld beim Laden des Formulars füllen

Das Ereignis `Load` tritt ein, wenn ein Formular geöffnet und dessen Datensätze angezeigt werden. Dabei wird dieses Ereignis in der zeitlichen Reihenfolge vor dem Ereignis `Open` abgewickelt. Wenn Sie auf Werte bestimmter Datenfelder reagieren möchten, die nach dem Ereignis zur Verfügung stehen, dann ist dieses Ereignis das richtige.

Im folgenden Beispiel aus Listing 432 wird im Listenfeld aus `ListboxFüllen` ein Listenfeld mit den in der Datenbank enthaltenen Tabellen gefüllt und der letzte Eintrag im Listenfeld markiert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_ListboxFüllem
'=================================================

Private Sub Form_Load()
   Dim Katalog As ADOX.Catalog
   Dim TabInfo As ADOX.Table
   Dim intZ As Integer

   Set Katalog = New ADOX.Catalog
   Katalog.ActiveConnection = CurrentProject.Connection

   For Each TabInfo In Katalog.Tables
      With TabInfo
         If TabInfo.Type = "TABLE" Then
            Me!Liste0.AddItem .Name
         End If
      End With
   Next

   intZ = Me.Liste0.ListCount
   Me.Liste0.Selected(intZ - 1) = True
   Set Katalog = Nothing
End Sub
```

Listing 432: Listenfeld füllen und letzten Eintrag markieren

Mithilfe der Eigenschaft `ListCount` ermitteln Sie die Anzahl der Einträge im Listenfeld `Liste0`. Da die Zählung bei Null anfängt, also der erste Eintrag im Listenfeld den Index 0 hat, müssen Sie vom letzten Eintrag des Listenfelds den Wert 1 subtrahieren. Über die Eigenschaft `Selected` markieren Sie einen Eintrag im Listenfeld.

356 Schließen verhindern

In der folgenden Lösung soll verhindert werden, dass ein Formular geschlossen werden kann. Genauer gesagt, es soll eine Rückfrage angezeigt werden, ob das Formular geschlossen werden soll oder nicht.

Im Beispiel aus Listing 433 wird beim Schließen des Formulars `Kunden2` nachgefragt, ob der Vorgang wirklich durchgeführt werden soll.

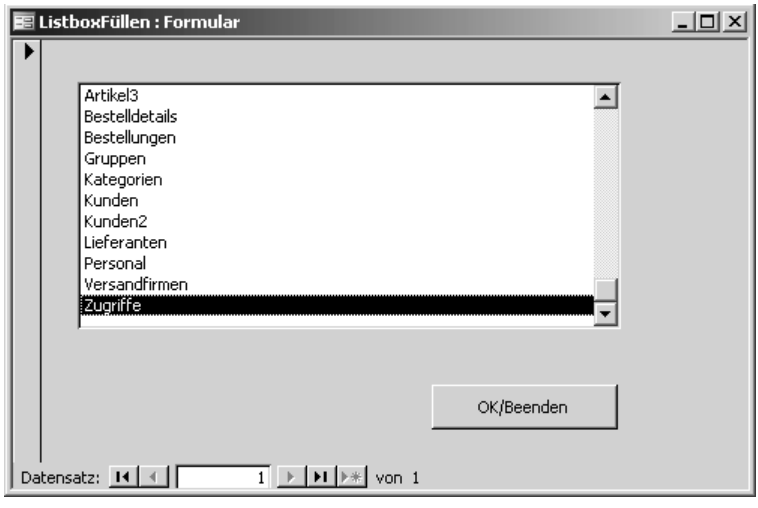


Abbildung 272: Der letzte Eintrag im Listenfeld wurde markiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden2
'=====================================================

Private Sub Form_Unload(Cancel As Integer)
   Dim strRück As String

   strRück = MsgBox("Möchten Sie beenden?", vbYesNo + vbQuestion)

   If strRück <> "" Then
      If strRück = 6 Then
         Cancel = False
      Else
         Cancel = True
      End If
   End If
End Sub
```

Listing 433: Abfrage beim Schließen des Formulars anzeigen

Über die Konstante `vbYesNo` definieren Sie, dass das Meldungsfenster mit den beiden Schaltflächen JA und NEIN angezeigt wird. Die Konstante `vbQuestion` sorgt dafür, dass ein zusätzliches Fragezeichensymbol im Fenster angezeigt wird. Für den Fall, dass die Schaltfläche JA angeklickt wird, wird der Rückgabewert 6 geliefert. In diesem Fall setzen Sie das Argument `Cancel` auf den Wert `True`, wenn der Anwender die Schaltfläche NEIN anklickt. Das Formular bleibt dann geöffnet (siehe Abbildung 273).

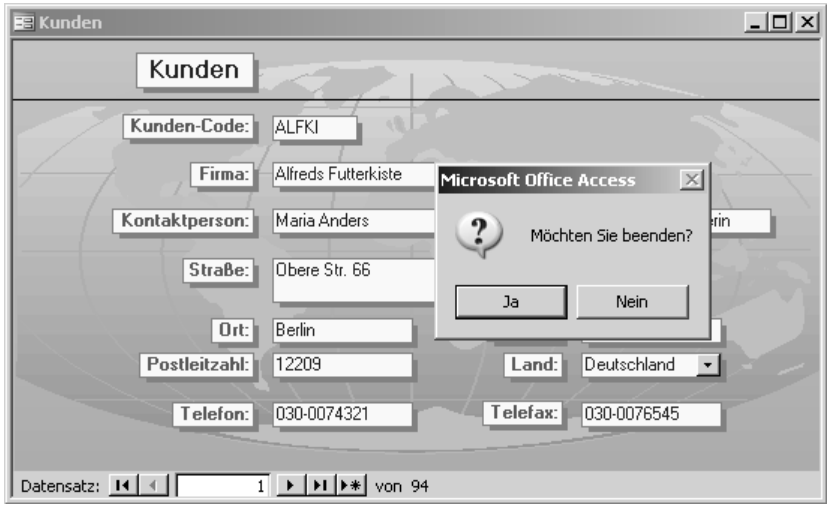


Abbildung 273: Beim Beenden der Form erscheint die Rückfrage.

357 Rechnungsfälligkeiten überwachen

Gegeben ist eine Tabelle mit dem Namen `Rechnungsliste` aus der Abbildung 274 und ein Formular `Rechnungsliste`, über das Sie Ihre Rechnungen erfassen und überwachen. Wenn Sie durch das Formular blättern, dann wollen Sie auf Rechnungen, deren Fälligkeitstermin bereits überschritten ist, sofort aufmerksam gemacht werden (siehe Abbildung 274).

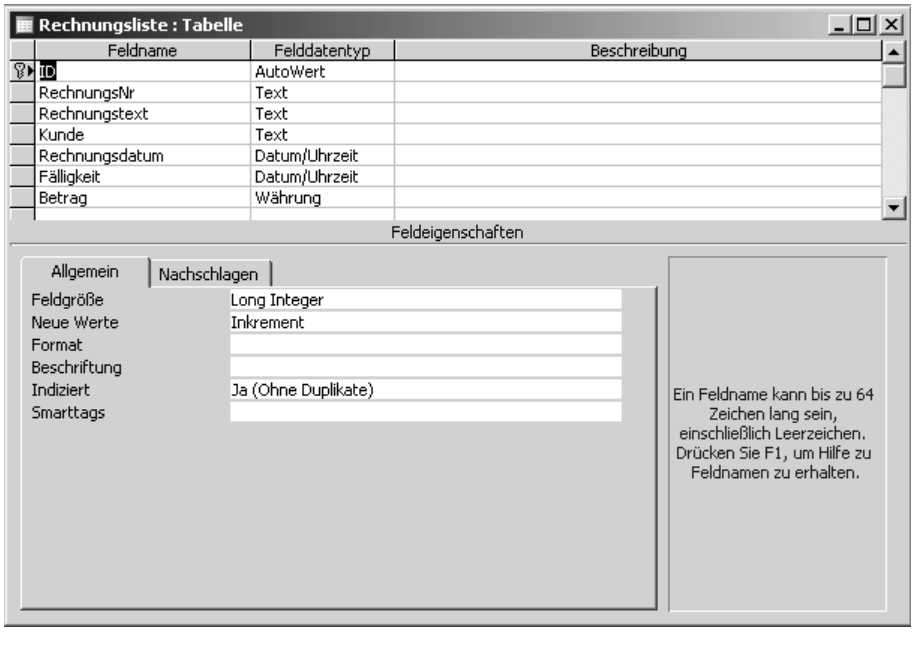


Abbildung 274: Der Aufbau der Rechnungsliste

Erstellen Sie im nächsten Schritt ein Formular auf Basis der Tabelle `Rechnungsliste`. Nutzen Sie den Formular-Assistenten, um auf schnellstem Wege ein dazugehöriges Formular zu erzeugen. Hinterlegen Sie danach das Ereignismakro `Form_Current` aus Listing 434, das automatisch beim Blättern durch das Formular ausgelöst wird.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Rechnungsliste
'=================================================

Private Sub Form_Current()
   If Me.Fälligkeit.Value <= Date Then
      Me.Fälligkeit.ForeColor = RGB(255, 0, 0)
      Me.Fälligkeit.FontBold = True
   Else
      Me.Fälligkeit.ForeColor = RGB(0, 0, 0)
      Me.Fälligkeit.FontBold = True
   End If
End Sub
```

Listing 434: Rechnungsfälligkeiten überwachen

Überprüfen Sie, ob im Feld `Fälligkeit` ein Datum steht, das kleiner bzw. gleich dem heutigen Datum ist. Ist dies der Fall, dann setzen Sie die Eigenschaft `ForeColor` ein, um die Schriftfarbe des Felds zu formatieren. Die gewünschte Farbe mischen Sie über die Funktion `RGB`. Über die Eigenschaft `FontBold` weisen Sie der Schriftart die Formatierung `FETT` zu, indem Sie diese mit dem Wert `True` versehen.

Abbildung 275: Die Rechnungsfälligkeit wurde überschritten.

358 Information beim letzten Satz

Das Ereignis `Current` tritt ein, wenn der Fokus auf einen Datensatz verschoben wird, der somit zum aktuellen Datensatz wird. Es tritt außerdem ein, wenn das Formular aktualisiert bzw. erneut abgefragt wird.

Im nächsten Beispiel aus Listing 435 soll eine Meldung im Formular `Artikel` ausgegeben werden, wenn Sie durch die einzelnen Datensätze blättern und am letzten Satz angekommen sind.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=================================================

Private Sub Form_Current()
   If Me.CurrentRecord <> 1 Then
      If Me.RecordsetClone.RecordCount = Me.CurrentRecord Then
         MsgBox "Letzter Satz erreicht!", vbInformation
         DoCmd.GoToRecord acDataForm, "Artikel", acFirst
      End If
   End If
End Sub
```

Listing 435: Bei Erreichen des letzten Satzes eine Meldung anzeigen

Mit der Eigenschaft `RecordsetClone` stellen Sie eine Kopie der Abfrage oder Tabelle her, die dem Formular zugrunde liegt und von der Einstellung der Eigenschaft `RecordSource` des Formulars festgelegt wird. Mit dieser Kopie können Sie jetzt machen, was Sie wollen, ohne dass Sie damit Schaden anrichten können. Wenden Sie Eigenschaft `RecordCount` an, um die Anzahl der hinterlegten Datensätze zu erfragen. Diese Anzahl vergleichen Sie dann mit dem momentan eingestellten Datensatz im Formular. Für den Fall, dass der letzte Datensatz erreicht ist, geben Sie eine Meldung aus. Damit beim Öffnen des Formulars nicht gleich die Meldung angezeigt wird, fragen Sie ab, ob der aktuell eingestellte Satz einen Wert ungleich 1 aufweist. Danach stellen Sie über den Einsatz der Methode `GoToRecord` den ersten Datensatz im Formular wieder ein.

359 Dynamisches Ein- und Ausblenden von Feldern

In Abhängigkeit bestimmter Datenfelder im Formular können Sie andere Formularfelder ein- und ausblenden. Im folgenden Beispiel wurde im Formular `Personal` das Feld `Position` überwacht. Dabei soll die Telefonnummer dann ausgeblendet werden, wenn im Feld `Position` der Eintrag `Geschäftsführer` steht. Die Lösung für diese Aufgabe sehen Sie in Listing 436.

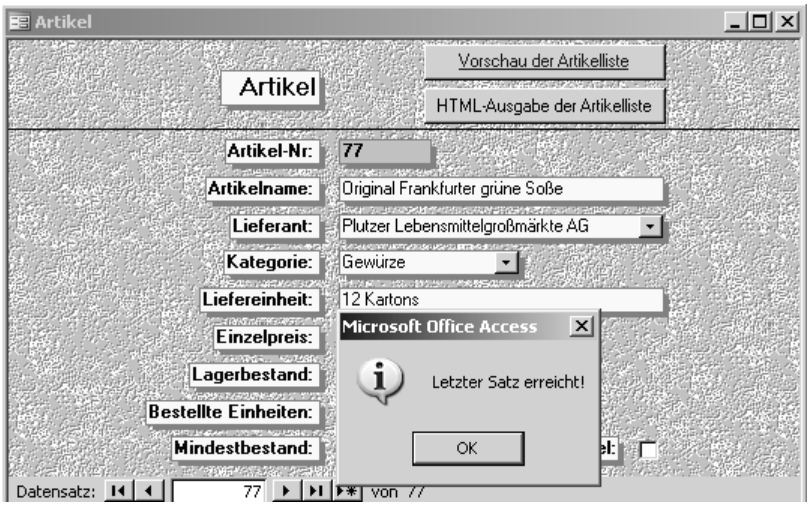


Abbildung 276: Der letzte Satz im Formular wurde erreicht.

```
'======================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'======================================================

Private Sub Form_Current()
   If Me.Position = "Geschäftsführer" Then
      Me.[Durchwahl Büro].Visible = False
   Else
      Me.[Durchwahl Büro].Visible = True
   End If
End Sub
```

Listing 436: Bestimmte Felder ein- und ausblenden

Mithilfe der Eigenschaft `Visible` können Sie ein Feld ausblenden, indem Sie dieser Eigenschaft den Wert `False` zuweisen.

Achten Sie bei Datenfeldern, die Leerzeichen im Feldnamen enthalten, darauf, dass Sie den Feldnamen in eckige Klammern setzen.

360 Variablen Titel im Formular einstellen

Das Ereignis `Form_Current` können Sie auch einsetzen, um Informationen in der Titelleiste des Formulars anzuzeigen. Standardmäßig wird hierbei der Name des Formulars angezeigt.

Im Formular `Kategorien` wird in Listing 437 der Name der jeweiligen Kategorie in Großbuchstaben angezeigt. Wenn eine neue Kategorie hinzugefügt wird, erfolgt die Ausgabe des aktuellen Datums in der Titelleiste.

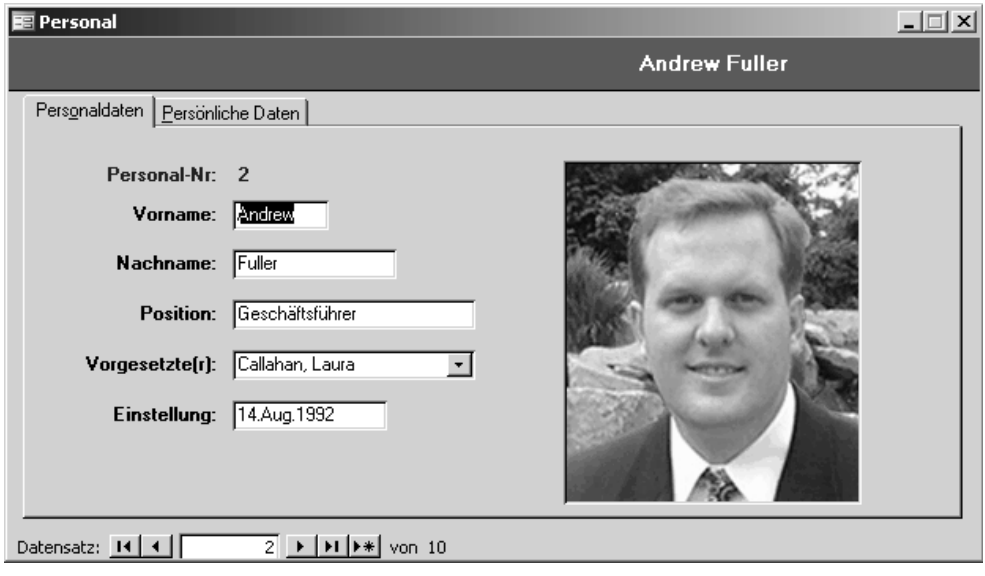


Abbildung 277: Beim Geschäftsführer wurde die Telefonnummer ausgeblendet.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kategorien
'=================================================

Private Sub Form_Current()
   If Me.NewRecord Then
      Me.Caption = "Neue Kategorie  " & Date
   Else
      Me.Caption = UCase(Me.Kategoriename)
   End If
End Sub
```

Listing 437: Einen variablen Titel im Formular einstellen

Mit der Eigenschaft `NewRecord` können Sie überprüfen, ob der aktuell ausgewählte Datensatz ein neuer Datensatz ist. Über die Eigenschaft `Caption` können Sie die Titelleiste im Formular füllen. Dabei kann auch die Funktion `UCase` zum Einsatz kommen, um den Titel in Großbuchstaben umzuwandeln.

361 QuickInfos definieren

Im nächste Beispiel aus Listing 438 setzen Sie das Ereignis `Form_Current` dazu ein, um für jedes Textfeld im Formular `Artikel2` eine QuickInfo zu definieren. Diese QuickInfo wird angezeigt, wenn Sie ein Textfeld aktivieren. Als Text für die QuickInfo definieren Sie in diesem Beispiel den augenblicklichen Inhalt des Textfelds.

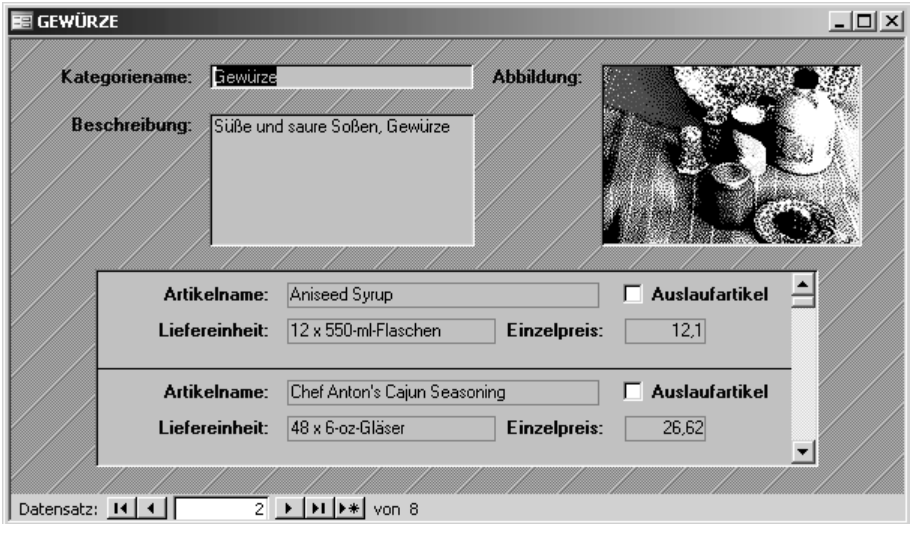


Abbildung 278: Der Kategoriename wird als Fensterüberschrift definiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Form_Current()
   Dim ctrl As Control

   For Each ctrl In Me.Controls
      If ctrl.ControlType = acTextBox Then
         ctrl.ControlTipText = ctrl.Value
      End If
   Next ctrl
End Sub
```

Listing 438: Den Feldwert zusätzlich als QuickInfo angeben

In einer `For each next`-Schleife durchlaufen Sie alle Steuerelemente in dem Formular. Innerhalb der Schleife prüfen Sie über die Eigenschaft `ControlType`, ob das jeweilige Steuerelement ein Textfeld ist. Wenn ja, dann wenden Sie die Eigenschaft `ControlTipText` an, um eine QuickInfo festzulegen. Über die Eigenschaft `Value` übertragen Sie dabei den Wert des Textfelds.

362 Eingaben limitieren

Stellen Sie sich vor, Sie haben ein Formular, in dem Sie die Anlage neuer Sätze beschränken möchten. Dabei sollen in der verknüpften Tabelle, die hinter dem Formular hinterlegt ist, nur 100 Sätze maximal eingegeben werden können.

Abbildung 279: Die QuickInfo beim Feld Liefereinheit

Im Beispiel aus Listing 439 wird im Formular `Kunden2` die Eingabe von höchstens 100 Sätzen erlaubt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden2
'=================================================

Private Sub Form_Current()
   Dim DBS As Recordset
   Dim intz As Integer

   Set DBS = RecordsetClone
   intz = DBS.RecordCount

   If Me.NewRecord Then
      Me.AllowAdditions = (intz < 100)
      MsgBox "Die max. Anzahl der Sätze ist erreicht!"
   Else
      Me.AllowEdits = True
   End If
End Sub
```

Listing 439: Die Eingabe von Datensätzen einschränken

Mit der Eigenschaft `RecordsetClone` erstellen Sie eine Kopie der Tabelle, die dem Formular zugrunde liegt. Über die Methode `RecordCount` zählen Sie die bereits erfassten Datensätze der mit dem Formular verknüpften Tabelle. Fragen Sie über die Eigenschaft `NewRecord` nach, ob es sich bei dem aktuell eingestellten Satz im Formular um einen neuen Satz handelt oder nicht. Da Sie alle Änderungen an bereits bestehenden Datensätzen zulassen möchten, setzen Sie diese Eigenschaft `AllowEdits` im `Else`-Zweig der `If`-Anweisung auf den Wert `True`. Im `Then`-Zweig der `If`-Anweisung definieren Sie die Bedingung `intzl < 100`.

363 Neue Mitarbeiter dokumentieren

Im folgenden Beispiel werden über das Formular PERSONAL neue Mitarbeiter erfasst. Um nun eine lückenlose Dokumentation der Neueinstellungen zu erhalten, erstellen Sie zunächst eine Tabelle wie in Abbildung 280 gezeigt.

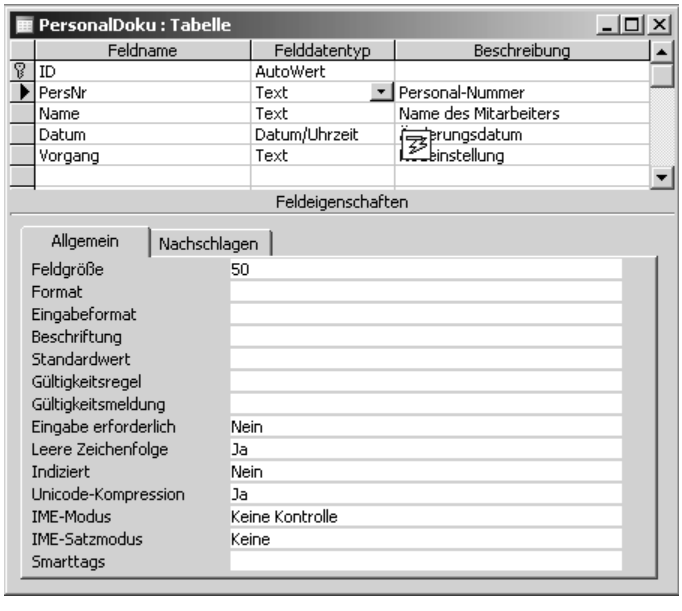


Abbildung 280: Die Tabelle zur Dokumentierung des Personals

Erfassen Sie das Ereignis `AfterInsert` und ergänzen Sie es wie in Listing 440 gezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Form_AfterInsert()
   Dim conn As New ADODB.Connection
   Dim dbs As ADODB.Recordset

   If Me.NewRecord = False Then
      MsgBox "Neuanlage"
      Set conn = CurrentProject.Connection
      Set dbs = New ADODB.Recordset
      dbs.Open "PersonalDoku", conn, adOpenKeyset, adLockOptimistic
      dbs.AddNew
      dbs.Fields("PersNR") = Me.[Personal-Nr]
```

Listing 440: Neue Einstellungen werden in einer Tabelle dokumentiert.

```
        dbs.Fields("Name") = Me.Nachname & ", " & Me.Vorname
        dbs.Fields("Datum") = Date
        dbs.Fields("Vorgang") = "Neueinstellung"
        dbs.Update
        dbs.Close
        Set dbs = Nothing
        Set conn = Nothing
    End If
End Sub
```

Listing 440: Neue Einstellungen werden in einer Tabelle dokumentiert. (Forts.)

Nachdem Sie Ihre Eingaben vorgenommen haben, überprüfen Sie mithilfe der Eigenschaft `NewRecord`, ob der erfasste Datensatz ein Neuzugang ist. Dabei ist zu anzumerken, dass hier nicht wie erwartet und in der Online-Hilfe beschrieben die Abfrage nach `True` zum Erfolg führt, sondern die Abfrage mit `False`.

Wenn es sich um einen neuen Datensatz handelt, dann stellen Sie die Verbindung zur Tabelle `PersonalDoku` her. Legen Sie über die Methode `AddNew` einen neuen Satz in der Tabelle an und übertragen Sie die Felder aus dem Formular `Personal` in die Tabelle. Vergessen Sie dabei nicht, die Methode `Update` am Ende des Übertragungsvorgangs anzugeben. Damit wird der Satz endgültig in der Tabelle gespeichert. Schließen Sie danach die Tabelle über die Methode `Close` und heben Sie die Objektverknüpfung auf.

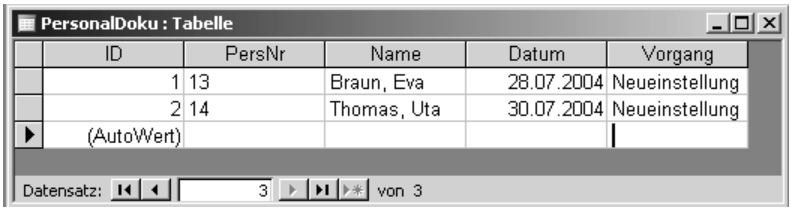


Abbildung 281: Alle Neueinstellungen werden lückenlos festgehalten.

364 Felder automatisch vorbelegen

Das Ereignis `BeforeInsert` tritt auf, wenn der Benutzer das *erste Zeichen* in einen *neuen Datensatz* eingibt, jedoch bevor der entsprechende Datensatz tatsächlich erstellt wird. Dieses Ereignis können Sie beispielsweise einsetzen, um bestimmte Felder im Formular automatisch vorzubelegen.

Im nächsten Beispiel aus Listing 441 werden Sie im Formular `Kunden` während der Anlage eines neuen Kunden die Felder `Region`, `Land` und `Ort` automatisch füllen. Gehen Sie bei diesem Beispiel davon aus, dass diese drei Felder nahezu immer dem zuletzt angelegten Datensatz entsprechen, da Sie gewöhnlich immer alle Kunden aus einem Land, einem Ort bzw. einer Region nacheinander eingeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=====================================================

Sub Form_BeforeInsert(Cancel As Integer)
   With Me
      .Ort = DLast("Ort", "Kunden")
      .Region = DLast("Region", "Kunden")
      .Land = DLast("Land", "Kunden")
   End With
End Sub
```

Listing 441: Die Vorbelegung einiger Felder vornehmen

Mithilfe der Funktion `DLast` können Sie den zuletzt erfassten Datensatz in einer Tabelle ermitteln und beispielsweise in ein Formular übertragen.

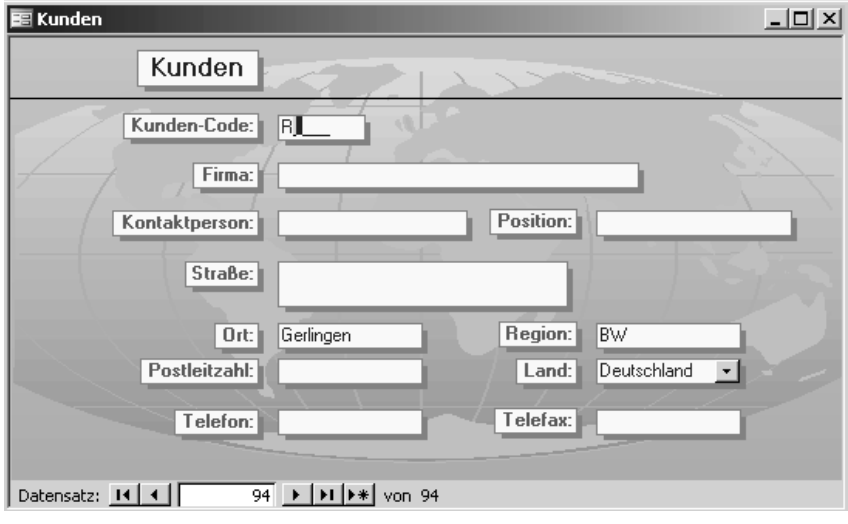


Abbildung 282: Die zuletzt vorgenommenen Einträge wurden automatisch eingestellt.

Wenn Sie lieber feste Werte als Vorbelegungstexte haben möchten, dann geben Sie diese wie in Listing 442 gezeigt in doppelten Anführungszeichen an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden (auskommentiert)
'=====================================================
```

Listing 442: Feste Texte als Vorgabe verwenden

```
Sub Form_BeforeInsert(Cancel As Integer)
   With Me
      .Ort = "Gerlingen"
      .Region = "BW"
      .Land = "Deutschland"
   End With
End Sub
```

Listing 442: Feste Texte als Vorgabe verwenden (Forts.)

Als weitere Variante können Sie sich einen Dummysatz in einer Tabelle definieren, z. B. den ersten Datensatz, den Sie dann immer standardmäßig übertragen, wenn Sie das erste Zeichen in einem neuen Datensatz im Formular eingeben. Im folgenden Beispiel aus Listing 443 wird in `Artikel2` jeweils der erste Satz der Tabelle `Artikel` eingesetzt, wenn ein neuer Satz angegangen wird.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Form_BeforeInsert(Cancel As Integer)
   Dim DBS As Recordset

   Set DBS = Me.RecordsetClone
   With Me
      If DBS.RecordCount > 0 Then
         DBS.MoveFirst
         .Artikelname = DBS!Artikelname
         .[LieferantenNr] = DBS![Lieferanten-Nr]
         .[KategorieNr] = DBS![Kategorie-Nr]
         .Liefereinheit = DBS!Liefereinheit
         .Einzelpreis = DBS!Einzelpreis
         .Lagerbestand = DBS!Lagerbestand
         .BestellteEinheiten = DBS!BestellteEinheiten
         .Mindestbestand = DBS!Mindestbestand
         .Auslaufartikel = DBS!Auslaufartikel
      End If
   End With
End Sub
```

Listing 443: Den ersten Satz im Formular einstellen

Mit der Eigenschaft `RecordsetClone` erstellen Sie eine Kopie des Datensatzes, der dem Formular zugrunde liegt und von der Einstellung der Eigenschaft `RecordSource` des Formulars festgelegt wird. Danach aktivieren Sie den ersten Satz der Tabelle über die Methode `MoveFirst`. Jetzt übertragen Sie die Inhalte dieses Satzes Feld für Feld in die Formularfelder.

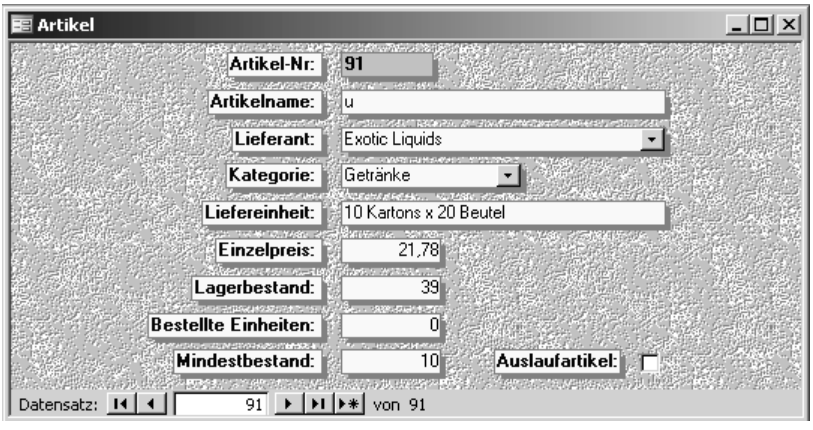


Abbildung 283: Der erste Datensatz wurde als Vorlage verwendet.

365 Rückfrage vor dem Speichern einholen

Das Ereignis `BeforeUpdate` können Sie einsetzen, um vor dem Speichern eines neuen Datensatzes bzw. einer Änderung eines bereits angelegten Datensatzes eine Rückfrage einzuholen. Im Makro aus Listing 444 wurde für das Formular `Lieferanten` diese Rückfrage eingebaut.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lierferanten
'=================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
   If MsgBox("Möchten Sie wirklich speichern?", _
      vbYesNo + vbQuestion, "Speichern bestätigen") <> vbYes Then
      Cancel = True
      Me.Undo
   End If
End Sub
```

Listing 444: Eine Rückfrage vor dem Speichern einholen

Fragen Sie über eine `MsgBox` ab, ob der veränderte Wert auch tatsächlich zurückgeschrieben werden soll. Ist dies nicht der Fall, dann wenden Sie die Methode `Undo` an, um den ursprünglichen Wert wieder einzustellen, und setzen Sie das Argument `Cancel` auf den Wert `True`, um das Ereignis abzubrechen.

366 Eingabeprüfung vornehmen

Das Ereignis `Form_BeforeUpdate` können Sie unter anderem gut einsetzen, wenn Sie Eingaben in Formularfeldern kontrollieren möchten.

Im folgenden Beispiel aus Listing 445 wird im Formular `Personal` geprüft, ob die Eingabe einer Postleitzahl richtig durchgeführt wurde. Für die Länder Frankreich, Italien, Spanien und Deutschland gilt eine einheitliche Länge von fünf Zeichen. In Australien und Singapur wird mit einer vierstelligen Postleitzahl gearbeitet. In Kanada enthält die Postleitzahl sowohl numerische als auch alphanumerische Zeichen in zwei Dreierblöcken, z.B. H1J 1C3 für Quebec.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
   Select Case Me!Land
      Case IsNull(Me![Land])
         Exit Sub
      Case "Frankreich", "Italien", "Spanien", "Deutschland"
         If Len(Me![PLZ]) <> 5 Then
            MsgBox "Postleitzahl muss 5 Zeichen lang sein"
            Cancel = True
            Me![PLZ].SetFocus
         End If
      Case "Australien", "Singapur"
         If Len(Me![PLZ]) <> 4 Then
            MsgBox "Postleitzahl muss 4 Zeichen lang sein"
            Cancel = True
            Me![PLZ].SetFocus
         End If
      Case "Kanada"
         If Not Me![PLZ] Like "[A-Z][0-9][A-Z] [0-9][A-Z][0-9]" Then
            MsgBox "Postleitzahl ist ungültig. "
            Cancel = True
            Me![PLZ].SetFocus
         End If
      Case Else
         'keine Prüfung
   End Select
End Sub
```

Listing 445: Die PLZ-Prüfung vor dem Speichern des Satzes durchführen

Über eine `Select Case`-Anweisung wird das Feld `Land` ausgewertet. Ist kein Land erfasst, kann auch die PLZ-Prüfung unterbleiben. Daher wird in diesem Zweig das Makro über den Befehl `Exit Sub` sofort verlassen. Über die Funktion `Len` prüfen Sie die Textlänge der verschiedenen Postleitzahlen. Entspricht dabei die Vorgabe nicht der Realität, setzen Sie das Argument `Cancel` auf den Wert `True`, um die Speicherung des Datensatzes abzubrechen. Mithilfe der Methode `SetFocus` setzen Sie den Mauszeiger danach direkt in das Feld `PLZ`, um eine schnelle Korrektur zu ermöglichen.

Die Postleitzahl von Kanada erfordert einen Abgleich eines Musters, bestehend aus zwei Dreierblöcken. Dabei entspricht die erste Ziffer im ersten Block einem Buchstaben zwischen A und Z, die zweite Ziffer setzt sich aus einer Zahl zwischen 0 und 9 zusammen, die dritte Ziffer muss dann wieder im Buchstabenbereich A bis Z liegen. Der zweite Block enthält als erstes und drittes Zeichen eine Zahl zwischen 0 und 9. In der Mitte muss wieder ein Buchstabe stehen. Einen solchen Vergleich führen Sie über `Like` durch.

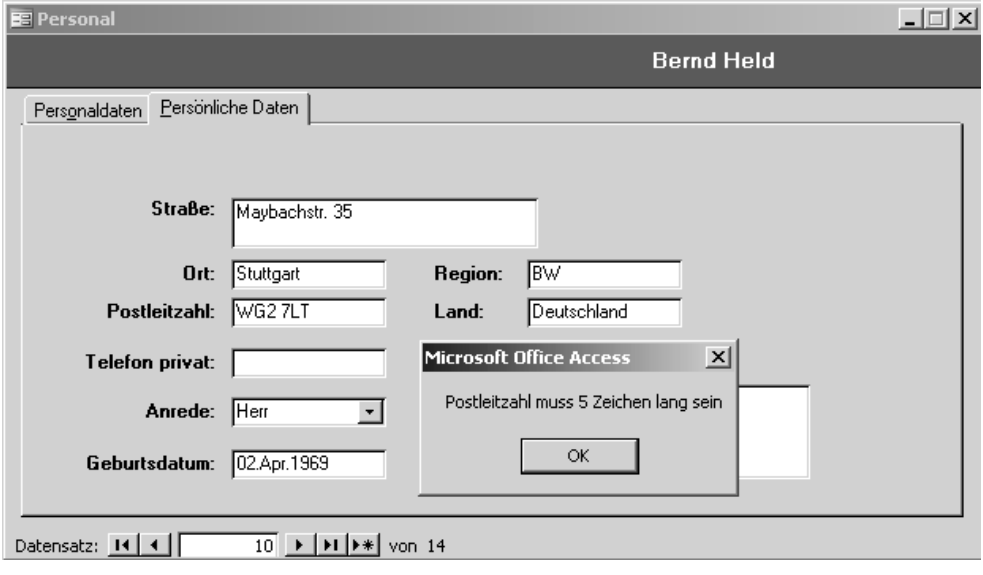


Abbildung 284: Eine ungültige Postleitzahl für Deutschland wurde erfasst.

367 Muss-Felder definieren

Auf eine ganz ähnliche Weise können Sie Muss-Felder definieren, d.h., in diesen Feldern muss ein Eintrag erfolgen, damit der Datensatz gespeichert werden kann.

Im folgenden Beispiel aus Listing 446 werden die beide Felder `Lagerbestand` sowie `Mindestbestand` im Formular `Artikel` als Muss-Felder definiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
   If IsNull(Me.Lagerbestand) Then
      MsgBox "Geben Sie eine Lagerbestandsmenge ein!"
      Cancel = True
```

Listing 446: Muss-Felder definieren

```
    End If
    If IsNull(Me.Mindestbestand) Then
       MsgBox "Geben Sie einen Mindestbestand ein!"
       Cancel = True
    End If
 End Sub
```

Listing 446: Muss-Felder definieren (Forts.)

Mithilfe der Funktion `IsNull` überprüfen Sie, ob in einem Feld eine Eingabe vorgenommen wurde. Wenn nicht, dann machen Sie den Anwender darauf mit einer Bildschirmmeldung aufmerksam.

368 Änderungen in Textdatei festhalten

Im nächsten Beispiel aus Listing 447 werden alle Änderungen im Formular `Kunden` bzw. in der dahinter liegenden Tabelle `Kunden` in eine Textdatei geschrieben.

```
 '=====================================================
 ' Auf CD     Buchdaten\Beispiele\Kap08
 ' Dateiname  Ereignisse.mdb
 ' Klasse     Form_Kunden
 '=====================================================

 Private Sub Form_BeforeUpdate(Cancel As Integer)
    Dim ctrl As Control

    Open "C:\Eigene Dateien\Änderungen.txt" For Append As #1
    Print #1, "Letzte Änderung: " & Now & " von " & Environ("username")

    For Each ctrl In Me.Controls
       With ctrl
          If .ControlType = acTextBox Then
             Print #1, .OldValue
          End If
       End With
    Next ctrl
    Print #1, "-*-*-*-"
    Close #1
 End Sub
```

Listing 447: Änderungen am Kundenstamm zusätzlich festhalten

Sobald Sie eine Änderung in der Tabelle `Kunden` über das Formular `Kunden` vornehmen, wird im Hintergrund eine Textdatei geöffnet. Dabei geben Sie bei der Methode `Open` das Argument `For Append` an. Damit werden alle Änderungen in der Textdatei jeweils am Ende der Textdatei angehängt. Schreiben Sie zu Beginn der jeweiligen Änderung das Änderungsdatum sowie den Namen des Anwenders mithilfe der Anweisung `Print`. Danach durchlaufen Sie mittels einer Schleife alle Textfelder des Formulars. Innerhalb der Schleife setzen Sie die Eigenschaft

OldValue ein, um die ursprünglichen Werte vor der Änderung zu ermitteln. Schließen Sie nach dem Schleifenaustritt die Textdatei über die Methode Close.

Abbildung 285: Alle Änderungen werden in einer separaten Textdatei festgehalten.

369 Neuanlage von Datensätzen verhindern

Das Ereignis Form_BeforeUpdate können Sie ebenso einsetzen, wenn Sie die Neuanlage eines Datensatzes über ein Formular verhindern möchten.

Im Beispiel aus Listing 448 wird im Formular KATEGORIEN für bestimmte Anwender die Neuanlage von Datensätzen verweigert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kategorien
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
   If Me.NewRecord And Environ("username") <> "admin" Then
      MsgBox "Es ist nicht möglich, neue Sätze anzuhängen!"
      Me.Undo
      Cancel = True
   End If
End Sub
```

Listing 448: Die Neuanlage von Datensätzen wird verweigert.

Mithilfe der Eigenschaft `NewRecord` können Sie überprüfen, ob ein Satz neu angelegt wurde. In diesem Fall wird der Wert `True` zurückgegeben. Die Methode `Undo` setzen Sie ein, um die ursprünglichen Werte im Formular wieder herzustellen. Indem Sie das Argument `Cancel` auf den Wert `True` setzen, wird der Speichervorgang abgebrochen.

370 Überhaupt keine Änderungen zulassen

Gehen Sie einen Schritt weiter und lassen Sie im nächsten Beispiel in Listing 449 überhaupt keine Änderung zu. Auch hier können Sie das Ereignis `Form_BeforeUpdate` einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kategorien (auskommentiert)
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
   If Me.Dirty Then
      MsgBox "Es ist nicht möglich, Sätze zu ändern!"
      Me.Undo
      Cancel = True
   End If
End Sub
```

Listing 449: Das Ereignis BeforeUpdate einsetzen, um Änderungen an Sätzen rückgängig zu machen

Mit der Eigenschaft `Dirty` ermitteln Sie, ob der aktuelle Datensatz seit dem letzten Speichern verändert wurde. Wenn ja, wenden Sie die Methode `Undo` an, um diese Änderung rückgängig zu machen. Setzen Sie das Argument `Cancel` auf den Wert `True`, um den Speichervorgang abzubrechen.

371 Letzte Änderung festhalten

Im Makro aus Listing 450 wurde in der Tabelle `Artikel` ein zusätzliches Feld mit dem Namen `LetzterUpdate` angelegt. Nachdem eine Änderung an einem Datensatz im Formular `Artikel` ausgeführt wird, wird in der Tabelle beim betreffenden Satz das Änderungsdatum automatisch geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================
```

Listing 450: Die letzte Änderung an einem Datensatz in der Tabelle zurückschreiben

```
Private Sub Form_AfterUpdate()
   Dim DBS As ADODB.Recordset
   Dim varx As Variant

   varx = DLookup("[Artikel-Nr]", "Artikel", "[Artikel-Nr] = " _
      & Forms!Artikel!ArtikelNr)
   varx = "[Artikel-Nr]='" & varx & "'"

   On Error GoTo fehler
   Set DBS = New ADODB.Recordset

   With DBS
      .Open Source:="Artikel", ActiveConnection:=CurrentProject.Connection, _
          CursorType:=adOpenKeyset, LockType:=adLockOptimistic

      .Find Criteria:=varx, SearchDirection:=adSearchForward

      If Not .EOF Then
         .Fields("LetzterUpdate") = Now
      End If
   End With
   DBS.Update
   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 450: Die letzte Änderung an einem Datensatz in der Tabelle zurückschreiben (Forts.)

Mithilfe der Funktion `DLookup` suchen Sie in der Tabelle `Artikel` nach dem gerade geänderten Datensatz im Formular. Das eindeutige Suchkriterium ist hierfür die Artikelnummer. Bilden Sie danach den Suchbegriff, der sich aus dem Feldnamen in der Tabelle sowie der eigentlichen Artikelnummer zusammensetzt, und übergeben Sie diesen Suchbegriff nach dem Öffnen der Tabelle der Methode `Find`. Wurde der Datensatz der Tabelle gefunden, dann übertragen Sie die aktuelle Zeit, inklusive des Datums, über die Funktion `Now` in das Feld `LetzterUpdate`. Wenden Sie danach die Methode `Update` an, um den Datensatz zu speichern. Im Anschluss daran schließen Sie die Tabelle über die Methode `Close` und heben den Objektverweis auf.

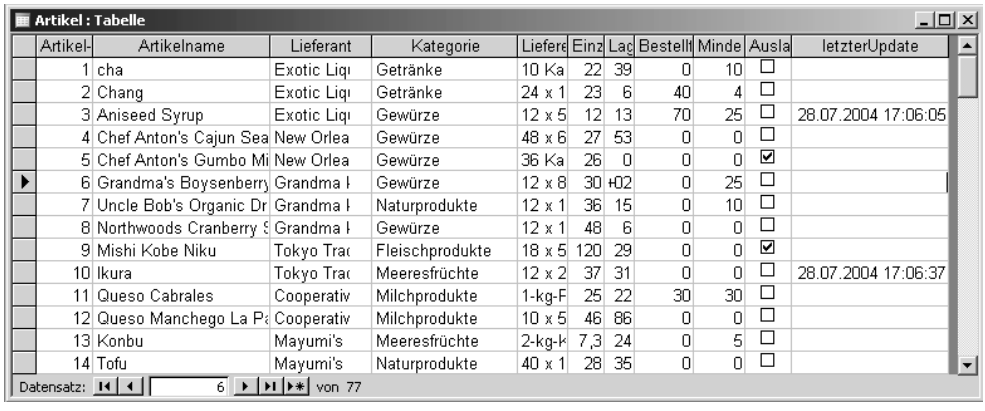


Abbildung 286: Die letzten Änderungen sind jetzt sofort ersichtlich.

372 Löschen von Datensätzen verhindern

Um einen Datensatz aus einem Formular zu löschen, können Sie standardmäßig aus dem Menü BEARBEITEN den Befehl DATENSATZ LÖSCHEN durchführen. Haben Sie wichtige Datensätze in Ihrem Datenbestand, die auf keinen Fall gelöscht werden dürfen, können Sie das Argument `Cancel` des Ereignisses `Form_Delete` einsetzen, um eine Löschung zu verhindern.

Im folgenden Beispiel aus Listing 451 wird im Formular `Artikel` jegliche Löschung von Artikeln verhindert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Form_Delete(Cancel As Integer)
   MsgBox "Es dürfen keine Datensätze gelöscht werden!", vbCritical
   Cancel = True
End Sub
```

Listing 451: Das Löschen von Datensätzen wird verhindert.

Indem Sie das Argument `Cancel` auf den Wert `True` setzen, brechen Sie den Löschvorgang ab. Wenn Sie diese Lösung so einsetzen, dann sollten Sie den Anwender dabei aber auf diese Funktion mittels einer Bildschirmmeldung aufmerksam machen.

In der Praxis wird man hin und wieder zusätzliche Kriterien definieren, die generell eine Löschung von Datensätzen erlauben, jedoch nicht in speziellen Fällen.

Im folgenden Beispiel aus Listing 452 dürfen Kunden aus `Deutschland` im Formular `Kunden` nicht gelöscht werden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=================================================

Private Sub Form_Delete(Cancel As Integer)
   If Me.Land = "Deutschland" Then
      MsgBox "Datensätze mit Kunden aus " & Me.Land & _
         " können nicht gelöscht werden!"
      Cancel = True
   End If
End Sub
```

Listing 452: Das Löschen von Datensätzen gilt nur für bestimmte Datensätze.

373 Eine eigene Löschabfrage einsetzen

Das Ereignis `BeforeDelConfirm` tritt auf, nachdem ein Anwender einen oder mehrere Datensätze gelöscht hat, jedoch bevor Microsoft Access ein Dialogfeld anzeigt, in dem der Benutzer zur Bestätigung des Löschvorgangs aufgefordert wird.

Wenn Sie einen Datensatz aus einem Formular über das Menü BEARBEITEN und den Befehl DATENSATZ LÖSCHEN entfernen möchten, wird von Access ein Standarddialogfeld angezeigt, in dem Sie die Löschaktion bestätigen müssen. Dieses Standarddialogfeld können Sie durch ein eigenes Dialogfeld im Formular `Artikel2` wie in Listing 453 gezeigt ersetzen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=================================================

Private Sub Form_BeforeDelConfirm(Cancel As Integer, Response As Integer)
   If MsgBox("Möchten Sie wirklich löschen?", _
      vbOKCancel + vbQuestion) = vbCancel Then
      Cancel = True
   End If
End Sub
```

Listing 453: Eine eigene Löschabfrage einsetzen

Das Dialogfeld soll die Schaltflächen OK und ABBRECHEN enthalten. Wenn die Schaltfläche ABBRECHEN angeklickt wurde, wird der Wert 2 zurückgemeldet, was der Konstanten `vbCancel` entspricht. In diesem Fall setzen Sie das Argument `Cancel` auf den Wert `True`, um die Standardlöschabfrage von Access abzubrechen.

374 Den letzten Datensatz einstellen

Im Beispiel aus Listing 454 wird im Formular `Artikel2` standardmäßig der letzte Datensatz der damit verknüpften Tabelle `Artikel` angezeigt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=================================================

Private Sub Form_Activate()
   DoCmd.GoToRecord , , acLast
End Sub
```

Listing 454: Den letzten Datensatz der Tabelle im Formular einstellen

Setzen Sie die Methode `GoToRecord` ein und übergeben Sie dieser Methode die Konstante `acLast`, um das Formular mit dem letzten Datensatz der verknüpften Tabelle einzustellen.

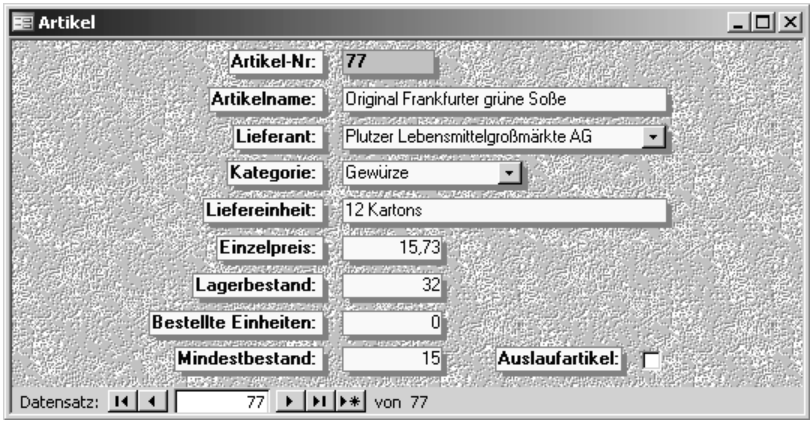


Abbildung 287: Der letzte Satz wird angezeigt.

375 Den zuletzt geänderten Satz einstellen

Erinnern Sie sich noch, wie Sie vorher die letzte Änderung eines Datensatzes in der Tabelle `Artikel` im Feld `LetzterUpdate` über das Ereignis `Form_AfterUpdate` vorgenommen haben?

Jetzt können Sie genau auf diese letzte Änderung zugreifen und beim Aktivieren des Formulars genau diesen Satz einstellen, indem Sie das Ereignis aus Listing 455 einsetzen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=================================================

Private Sub Form_Activate()
   Dim DBS As ADODB.Recordset
   Dim varx As Variant
   Dim lngPos As Integer

   varx = DMax("LetzterUpdate", "Artikel")
   varx = "LetzterUpdate='" & varx & "'"

   On Error GoTo fehler
   Set DBS = New ADODB.Recordset

   With DBS
      .Open Source:="Artikel", ActiveConnection:=CurrentProject.Connection, _
      CursorType:=adOpenKeyset, LockType:=adLockOptimistic

      .Find Criteria:=varx, SearchDirection:=adSearchForward
      lngPos = .AbsolutePosition

      If Not .EOF Then
         DoCmd.GoToRecord acDataForm, "Artikel", acGoTo, lngPos
      End If
   End With
   DBS.Close
   Set DBS = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 455: Den zuletzt geänderten Datensatz im Formular voreinstellen

Mithilfe der Funktion `DMax` suchen Sie in der Tabelle `Artikel` nach dem zuletzt geänderten Datensatz. Dieser Datensatz hat das »größte« Datum. Das eindeutige Suchkriterium ist hierfür das Datum der letzten Änderung aus dem Feld `LetzterUpdate`. Bilden Sie danach den Suchbegriff, der sich aus dem Feldnamen in der Tabelle sowie dem ermittelten höchsten Datum zusammensetzt, und übergeben Sie diesen Suchbegriff nach dem Öffnen der Tabelle der Methode `Find`. Wurde der Datensatz der Tabelle gefunden, dann speichern Sie die aktuelle Position des `Recordset`-Objekts über die Eigenschaft `AbsolutePosition` in der Variablen `lngPos`. Übergeben Sie diese Position dann an die Methode `GoToRecord`. Damit wird der zuletzt geänderte Datensatz im Formular angezeigt. Wenden Sie danach die Methode `Close` an, um die Tabelle zu schließen, und heben Sie den Objektverweis auf.

376 Variable Feldanpassung vornehmen

Bei Eintreten des `Resize`-Ereignisses können Sie ein Steuerelement verschieben oder dessen Größe ändern, wenn sich die Größe des Formulars ändert, in dem sich dieses Steuerelement befindet.

Im nächsten Beispiel aus Listing 456 werden Sie die Länge der beiden Textfelder `Firma` und `Straße` im Formular `Kunden` variabel gestalten.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=================================================

Private Sub Form_Resize()
   Dim intBreite As Integer

   intBreite = Me.WindowWidth
   Me!Firma.Width = intBreite / 2
   Me!Straße.Width = intBreite / 2
End Sub
```

Listing 456: Textfelder werden dynamisch angepasst.

Über die Eigenschaft `WindowWidth` ermitteln Sie die aktuelle Breite eines Formulars. Die so ermittelte Breite dividiert durch 2 weisen Sie den Textfeldern `Firma` und `Straße` als neue Breite zu. Sie erhalten somit einen Bewegungseffekt in Ihrem Formular.

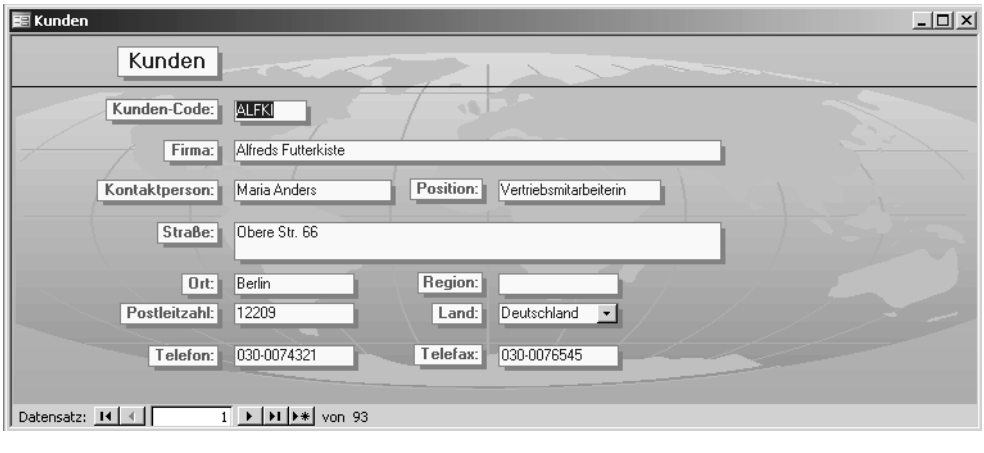


Abbildung 288: Beim Verbreitern des Formulars werden die Textfelder Firma und Straße angepasst.

377 Mit Doppelklick Felder ein- und ausblenden

Bei Formularen tritt das Ereignis `DblClick` ein, wenn der Benutzer auf eine leere Fläche oder auf einen Datensatzmarkierer des Formulars doppelt klickt. Im folgenden Beispiel aus Listing 457 werden mit einem Doppelklick auf das Formular `Artikel` bestimmte Felder je nach augenblicklichem Status entweder ein- oder ausgeblendet.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=================================================

Private Sub Detail_DblClick(Cancel As Integer)
   With Me
      If .Lagerbestand.Visible = True Then
         .Lagerbestand.Visible = False
         .BeschriftungLagerbestand.Visible = False
         .BestellteEinheiten.Visible = False
         .BeschriftungBestellteEinheiten.Visible = False
         .Mindestbestand.Visible = False
         .BeschriftungMindestbestand.Visible = False
         .Auslaufartikel.Visible = False
         .BeschriftungAuslaufartikel.Visible = False
      Else
         .Lagerbestand.Visible = True
         .BeschriftungLagerbestand.Visible = True
         .BestellteEinheiten.Visible = True
         .BeschriftungBestellteEinheiten.Visible = True
         .Mindestbestand.Visible = True
         .BeschriftungMindestbestand.Visible = True
         .Auslaufartikel.Visible = True
         .BeschriftungAuslaufartikel.Visible = True
      End If
   End With
End Sub
```

Listing 457: Felder dynamisch ein- und ausblenden

Prüfen Sie zuerst am Feld `Lagerbestand`, ob das Feld gerade sichtbar ist oder nicht. Je nachdem blenden Sie die restlichen Felder ein, indem Sie die Eigenschaft `Visible` der Steuerelemente auf den Wert `True` setzen, bzw. blenden Sie die Felder aus, indem Sie der Eigenschaft `Visible` den Wert `False` zuweisen.

Klicken Sie doppelt auf eine freie Fläche im Formular, um die Felder auszublenden.

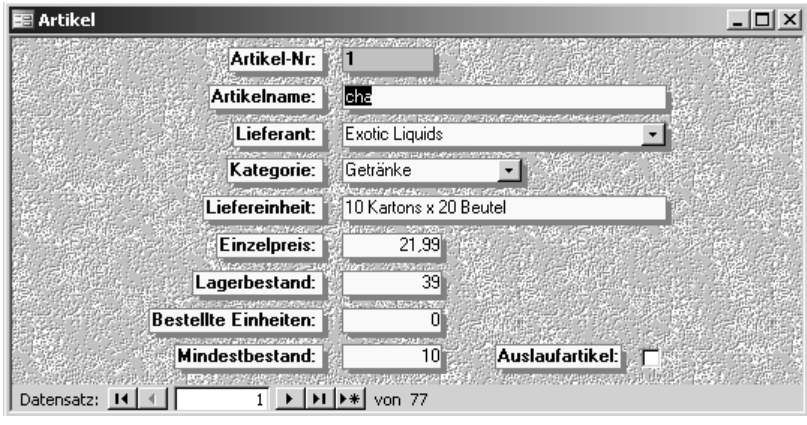


Abbildung 289: Alle Felder sind sichtbar.

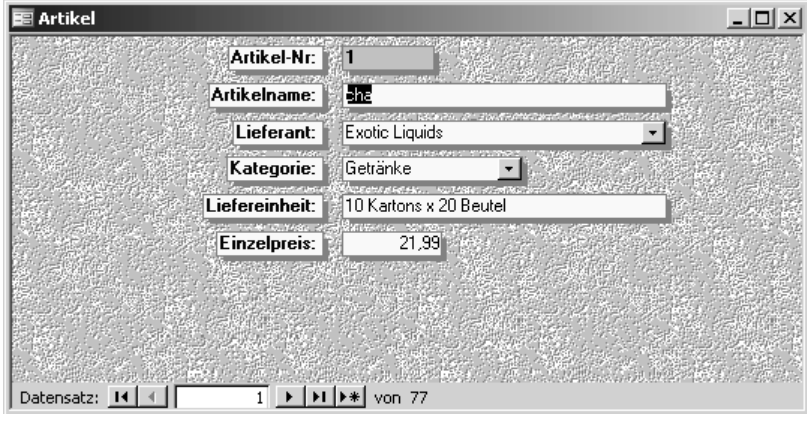


Abbildung 290: Einige Felder sind nun ausgeblendet.

378 Lagerbestand per Mausklick erhöhen

Das Ereignis `Click` tritt auf, wenn der Benutzer die linke Maustaste klickt, während sich der Mauszeiger auf einem freien Bereich oder auf Datensatzmarkierern in der Datenblattansicht des Formulars befindet. Dieses Ereignis kann man selbstverständlich auf jedes andere Steuerelement im Formular anwenden.

Eine kleine Erleichterung können Sie beispielsweise im Formular `Artikel` integrieren. Dabei soll durch einen Klick auf das Textfeld `Lagerbestand` derselbe jeweils um den Wert 1 erhöht werden. Stellen Sie dazu das `Click`-Ereignis für das Textfeld `Lagerbestand` wie in Listing 458 gezeigt ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Lagerbestand_Click()
   Me.Lagerbestand.Value = Me.Lagerbestand.Value + 1
End Sub
```

Listing 458: Lagerzugänge per Mausklick einpflegen

379 Welche Taste wurde gedrückt?

Listing 459 zeigt, wie man feststellen kann, welche Maustaste im Detailbereich des Formulars `Kunden` gedrückt wurde. Dabei kann es vorkommen, dass vorher zusätzlich zu den gedrückt gehaltenen Maustasten eine der Tasten Alt, ⇧ oder Strg gedrückt wurde.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=====================================================

Private Sub Detail_MouseDown(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   If Button = acLeftButton Then
      MsgBox "Sie haben die linke Taste gedrückt."
   End If

   If Button = acRightButton Then
      MsgBox "Sie haben die rechte Taste gedrückt."
   End If

   If Button = acMiddleButton Then
      MsgBox "Sie haben die mittlere Taste gedrückt."
   End If

   If Shift = 1 Then MsgBox "Umschalt-Taste gedrückt"
   If Shift = 2 Then MsgBox "Strg-Taste gedrückt"
   If Shift = 4 Then MsgBox "Alt-Taste gedrückt"
End Sub
```

Listing 459: Feststellen, welche Maustaste gedrückt wurde

Das Argument `Button` hat drei Konstanten, über die Sie ermitteln können, welche Taste auf der Maus gedrückt wurde. Die dazugehörigen Konstanten bzw. Zahlenwerte sind:

- acLeftButton: 1
- acRightButton: 2
- acMiddleButton: 4

Mithilfe des Arguments Shift können Sie feststellen, welche Tasten zusätzlich zu den Maustasten gedrückt wurden.

Hinter den Argumenten X und Y steht eine Zahl, die die aktuelle Position des Mauszeigers anzeigt. Die Werte X und Y bilden dabei die linke obere Ecke des Objekts.

380 Schaltfläche beim Klicken verändern

Im folgenden Beispiel aus Listing 460 wird im Formular ListboxFüllen die Schaltfläche beim Klicken verändert. Dabei wird der Schriftschnitt FETT sowie die Schriftfarbe ROT eingestellt. Beim erneuten Freigeben der linken Maustaste wird die Standardformatierung wieder hergestellt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_ListboxFüllen
'=================================================

Private Sub Befehl2_MouseDown(Button As Integer, Shift As Integer, _
   X As Single, Y As Single)
   Dim lngRot As Long

   lngRot = QBColor(4)
   With Me.Befehl2
      .FontBold = True
      .ForeColor = lngRot
   End With
End Sub


Private Sub Befehl2_MouseUp(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   Dim lngBlack As Long

   lngBlack = QBColor(0)
   With Me.Befehl2
      .FontBold = False
      .ForeColor = lngBlack
   End With
End Sub
```

Listing 460: Schaltflächen beim Klicken verändern

Über die Funktion QBColor bestimmen Sie die Farbe, die Sie über die Eigenschaft ForeColor der Beschriftung der Schaltfläche zuweisen. Die genaue Farbbelegung der Funktion QBColor können Sie im Anhang nachschlagen. Indem Sie die Eigenschaft FontBold auf den Wert True setzen, erreichen Sie, dass der Beschriftungstext der Schaltfläche im Fettdruck ausgegeben wird.

381 Kontextmenü deaktivieren

Wenn Sie in einem Formular die rechte Maustaste drücken, wird Ihnen standardmäßig ein Kontextmenü angeboten, über das Sie beispielsweise in eine andere Formularansicht springen können. Dieses Kontextmenü können Sie ausschalten, indem Sie das Ereignis MouseDown einsetzen. Dazu legen Sie fest, in welchem Bereich des Formulars dieses Kontextmenü ausgeschaltet werden soll.

Im folgenden Beispiel aus Listing 461 wird im Detailbereich des Formulars Lieferanten das Kontextmenü deaktiviert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=================================================

Private Sub Detail_MouseDown(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   If Button = acRightButton Then
      Me.ShortcutMenu = False
   End If
End Sub
```

Listing 461: Das Kontextmenü deaktivieren

Über die Abfrage der gedrückten Taste auf Ihrer Maus können Sie über das Argument acRightButton abfragen, ob die rechte Maustaste gedrückt wurde. Ist dies der Fall, schalten Sie das Kontextmenü ab, indem Sie die Eigenschaft ShortcutMenu auf den Wert False setzen.

382 Spezialeffekte einsetzen

Zu nahezu jedem Steuerelement stehen Ihnen Spezialeffekte zur Verfügung. Sie können beispielsweise Textfelder erhöht oder vertieft einstellen. Das folgende Beispiel geht vom Formular SPEZIALEFFEKTE aus und sieht wie in Abbildung 291 gezeigt aus.

Sie können jetzt das Ereignis MouseOver einsetzen, um einen anderen Spezialeffekt für die Felder einzustellen, über die Sie streichen.

Im folgenden Beispiel aus Listing 462 wird beim Überstreichen der Textfelder geprüft, welcher Spezialeffekt eingestellt ist, und dementsprechend ein anderer Spezialeffekt eingestellt.

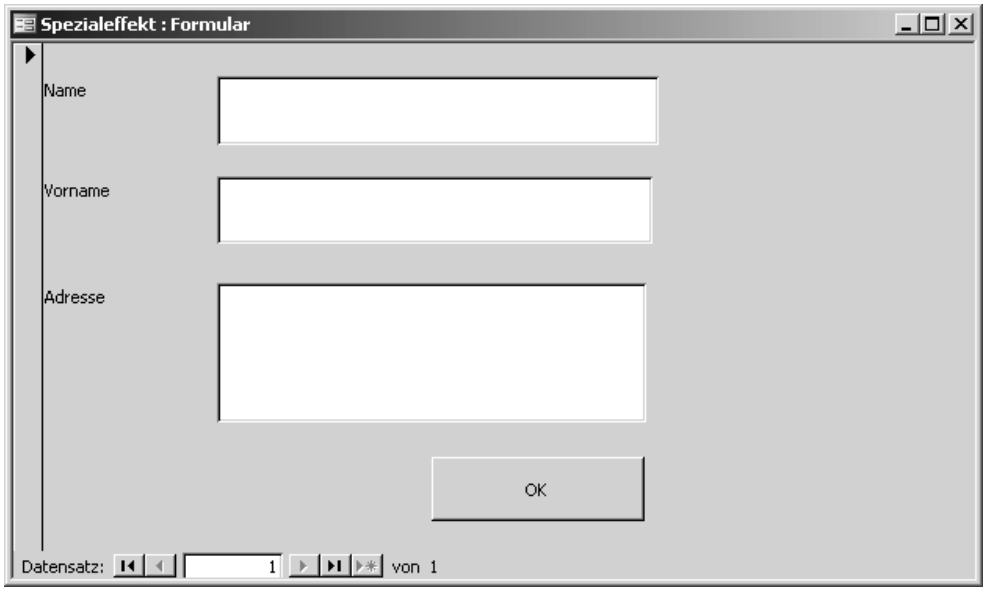


Abbildung 291: Die Textfelder sind alle identisch eingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Spezialeffekt
'=====================================================

Private Sub Text0_MouseMove(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   If Me!Text0.SpecialEffect = 1 Then
      Me!Text0.SpecialEffect = 2
   Else
      Me!Text0.SpecialEffect = 1
   End If
End Sub

Private Sub Text2_MouseMove(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   If Me!Text2.SpecialEffect = 1 Then
      Me!Text2.SpecialEffect = 2
   Else
      Me!Text2.SpecialEffect = 1
   End If
End Sub
```

Listing 462: Spezialeffekte für Textfelder einsetzen

```
Private Sub Text4_MouseMove(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   If Me!Text4.SpecialEffect = 1 Then
      Me!Text4.SpecialEffect = 2
   Else
      Me!Text4.SpecialEffect = 1
   End If
End Sub
```

Listing 462: Spezialeffekte für Textfelder einsetzen (Forts.)

Wenden Sie die Eigenschaft `SpecialEffect` an, um nach einer Mausbewegung über das entsprechende Textfeld ein erhöhtes Textfeld anzuzeigen.

Selbstverständlich stehen Ihnen noch weitere Spezialeffekte zur Verfügung, die Sie der folgenden Tabelle entnehmen können.

Einstellung	Wert	Beschreibung
Flach	0	Das Objekt wird flach und in den Standardfarben des Systems oder in benutzerdefinierten Farben angezeigt, die in der Entwurfsansicht eingestellt wurden.
Erhöht	1	Das Objekt ist oben und links hervorgehoben und hat unten und rechts einen Schatten.
Vertieft	2	Das Objekt hat oben und links einen Schatten und ist unten und rechts hervorgehoben.
Graviert	3	Das Objekt hat eine vertiefte Linie um das Steuerelement.
Schattiert	4	Das Objekt hat einen Schatten unterhalb und rechts des Steuerelements.
Unterstrichen	5	Das Objekt hat eine vertiefte Linie unterhalb des Steuerelements.

Tabelle 136: Die Spezialeffekte in Access

383 Ins Internet verzweigen

In dem Formular `Spezialeffekt` wurde bei der folgenden Aufgabe eine weitere Schaltfläche integriert, deren Beschriftung eine gültige Internetadresse beinhaltet. Wenn man über diese Schaltfläche mit der Maus streicht, wird dabei eine Rückfrage am Bildschirm angezeigt, ob ins Internet verzweigt werden soll oder nicht. Um diese Aufgabe zu lösen, kann die API-Funktion mit dem Namen `ShellExecute` aus Listing 463 gestartet werden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Spezialeffekt
'=================================================

Private Declare Function ShellExecute Lib
   "shell32.dll" Alias "ShellExecuteA" _
   (ByVal hwnd As Long, ByVal lpOperation As String, _
    ByVal lpFile As String, ByVal lpParameters As String, _
    ByVal lpDirectory As String, ByVal bShowCMD As Long) As Long


Private Sub Befehl10_MouseMove(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)
   Dim strR As String

   strR = MsgBox _
      ("Möchten Sie eine Verbindung ins Internet herstellen?", vbYesNo + vbQuestion)
   If strR <> "" Then

      If strR = 6 Then
         ShellExecute Me.hwnd, "open", Befehl10.Caption, 0, 0, 5
      End If
   End If
End Sub
```

Listing 463: Eine Internetseite besuchen

Die Sprungadresse für die Internetseite holen Sie sich direkt aus der Beschriftung der Schaltfläche mithilfe der Eigenschaft `Caption`. Die notwendige Aktion heißt `open`, was nichts anderes bedeutet, als dass die Internetseite geöffnet werden soll.

Eine alternative Lösung, eine Internetseite aufzurufen, ist der Einsatz des Internet Explorers. Dazu wurde im Formular eine zusätzliche Schaltfläche eingefügt und als Beschriftung die Internetseite von n-TV mit der URL `http://www.n-tv.de` angegeben. Danach wurde das `Click`-Ereignis für diese Schaltfläche eingestellt.

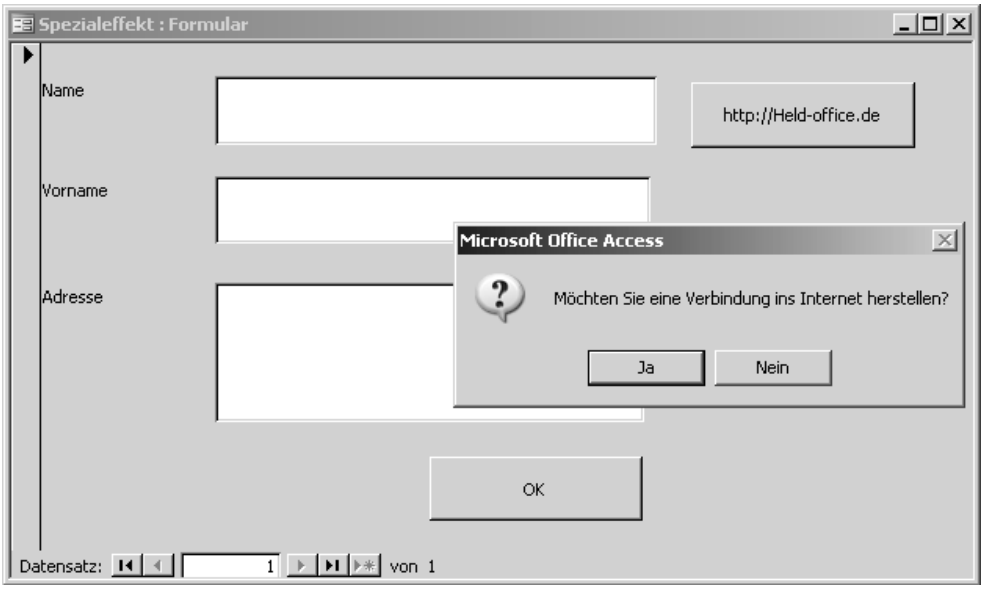


Abbildung 292: Rückfrage vor dem Sprung ins Internet

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Spezialeffket
'=====================================================

Private Sub Befehl11_Click()
   Dim IE As Object

   Set IE = CreateObject("Internetexplorer.application")
   IE.navigate Befehl11.Caption
End Sub
```

Listing 464: Den Internet Explorer starten

Deklarieren Sie zuerst einmal ein Objekt vom Typ `Object`. Zu diesem Zeitpunkt geben Sie also noch nicht bekannt, dass Sie eigentlich mit Objekten des Internet Explorers arbeiten möchten. Diese Vorgehensweise wird als Late-Binding bezeichnet. Bei diesem Late-Binding brauchen Sie keine Objektbibliotheken in der Entwicklungsumgebung unter EXTRAS/VERWEISE einzubinden, sondern das Objekt wird über die Funktion `CreateObject` erstellt. Danach können Sie auf Methoden und Eigenschaften zugreifen, die für den Internet Explorer angeboten werden. Unter anderem ist dies die Methode `Navigate`, die es erlaubt, eine Internetadresse oder eine andere Office-Datei (keine Access-Datenbanken) im Internet Explorer zu öffnen.

384 Textfelder optisch hervorheben

Eine Standardanwendung für die beiden Ereignisse ist das Ein- und Entfärben von Textfeldern beim Fokuserhalt bzw. Fokusverlust.

Im folgenden Beispiel aus Listing 465 wird im Formular `Textfelder` diese Methode angewendet. Dabei wird bei Fokuserhalt des Textfelds die Hintergrundfarbe des Textfelds mit GELB formatiert. Beim Fokusverlust wird die Standardfarbe des jeweiligen Textfelds wieder hergestellt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Textfelder
'=================================================

Private Sub Text0_GotFocus()
   Me.Text0.BackColor = QBColor(6)
End Sub

Private Sub Text0_LostFocus()
   Me.Text0.BackColor = QBColor(15)
End Sub

Private Sub Text2_GotFocus()
   Me.Text2.BackColor = QBColor(6)
End Sub

Private Sub Text2_LostFocus()
   Me.Text2.BackColor = QBColor(15)
End Sub

Private Sub Text4_GotFocus()
   Me.Text4.BackColor = QBColor(6)
End Sub

Private Sub Text4_LostFocus()
   Me.Text4.BackColor = QBColor(15)
End Sub
```

Listing 465: Textfelder bei Fokuserhalt färben

Über die Eigenschaft `BackColor` können Sie den Hintergrund eines Textfelds mit einer Farbe belegen. Die Farbe wird über die Funktion `QBColor` erzeugt. Dabei wird die Farbe Gelb über die Nummer 6 und die blendend weiße Farbe über die Nummer 15 repräsentiert. Eine Tabelle mit den Farbzuordnungen finden Sie im Anhang dieses Buchs.

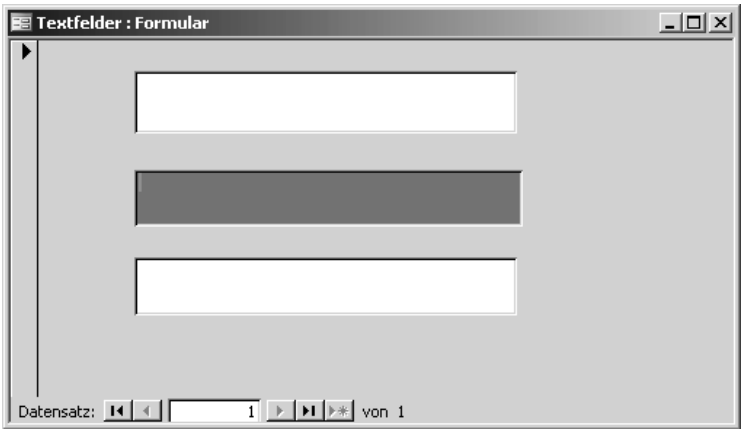


Abbildung 293: Das zweite Feld wurde bei Fokuserhalt mit Gelb hinterlegt.

385 Tasten erkennen

Um überhaupt einmal zu erkennen, wie man Tasten abfangen kann, wurde ein neues Formular mit dem Namen `Tasten` eingefügt, in dem eine Schaltfläche integriert wurde. Nach dem Aufruf dieses Formulars können Sie die Tasten [Pos1], [Ende], [Esc], [Enter], [Entf], [Rücktaste], [⇧] und [Strg] drücken. Alle diese Tasten werden im Makro aus Listing 466 erkannt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Tasten
'=====================================================

Private Sub Befehl0_KeyDown(KeyCode As Integer, Shift As Integer)
   Select Case KeyCode
      Case vbKeyHome
         MsgBox "Sie haben die Pos1-Taste gedrückt!"
      Case vbKeyEnd
         MsgBox "Sie haben die Ende-Taste gedrückt!"
      Case vbKeyEscape
         MsgBox "Sie haben die Esc-Taste gedrückt!"
      Case vbKeyReturn
         MsgBox "Sie haben die Enter-Taste gedrückt!"
      Case vbKeyDelete
         MsgBox "Sie haben die Entf-Taste gedrückt!"
      Case vbKeyBack
         MsgBox "Sie haben die Rücktaste gedrückt!"
      Case vbKeyShift
         MsgBox "Sie haben die Umschalttaste gedrückt!"
      Case vbKeyControl
         MsgBox "Sie haben die Strg-Taste gedrückt!"
```

Listing 466: Tasten auswerten über den KeyCode

```
    End Select
End Sub
```

Listing 466: Tasten auswerten über den KeyCode (Forts.)

Jede Taste kann über eine so genannte `KeyCode`-Konstante angesprochen werden.

Abbildung 294: Die Abfrage von bestimmten Tasten in einem Formular

Hinweis

Weitere Informationen über die Belegung der Tasten finden Sie im Antwort-Assistenten der Online-Hilfe unter dem Stichwort »Tasten-Code-Konstanten«.

386 Tastenkombinationen einrichten

Im folgenden Beispiel wurde ein neues Formular mit drei Textfeldern angelegt und unter dem Namen `Tastenkombinationen` gespeichert. Das Befüllen des Formulars kann entweder Feld für Feld über die Eingabe der Informationen Anwendername, Datum und Uhrzeit erfolgen oder eleganter und schneller über Tastenkombinationen erledigt werden. Für das Füllen der Felder wurden folgende Tastenkombinationen definiert:

- Strg + U: fügt den Namen des Anwenders ins Feld `Text0` ein.
- Strg + D: fügt das aktuelle Datum ins Feld `Text2` ein.
- Strg + T: Fügt die aktuelle Uhrzeit ins Feld `Text4` ein.

Die Tastenkombinationen werden über das Ereignis `KeyDown` der jeweiligen Textfelder wie in Listing 467 gezeigt eingestellt, d.h., die Tastenkombinationen sind nur gültig, wenn der Mauszeiger im jeweiligen Textfeld steht.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Tastenkombinationen
'=================================================

Private Sub Text0_KeyDown(KeyCode As Integer, Shift As Integer)
   'Die Tastenkombination Strg + U wird abgefragt
   If (Shift = acCtrlMask) And (KeyCode = Asc("U")) Then
      Me.Text0 = Environ("username")
   End If
End Sub


Private Sub Text2_KeyDown(KeyCode As Integer, Shift As Integer)
   'Die Tastenkombination Strg + D wird abgefragt
   If (Shift = acCtrlMask) And (KeyCode = Asc("D")) Then
      Me.Text2 = Date
   End If
End Sub


Private Sub Text4_KeyDown(KeyCode As Integer, Shift As Integer)
   'Die Tastenkombination Strg + T wird abgefragt
   If (Shift = acCtrlMask) And KeyCode = Asc("T") Then
      Me.Text4 = Time
   End If
End Sub
```

Listing 467: Die Tastenkombinationen abfragen und Aktionen durchführen

Da das Argument `KeyCode` einen numerischen Wert erwartet, wenden Sie die Funktion `Asc` an und übergeben ihr den gewünschten Buchstaben, zu dem Sie den zugeordneten numerischen Wert erhalten möchten. Im Argument `Shift` können Sie die Taste Strg abfragen, indem Sie die Konstante `acCtrlMask` verwenden. Den Anwendernamen können Sie über die Funktion `Environ` und die Konstante `Username` abfragen. Das Datum fügen Sie über die Funktion `Date`, die Uhrzeit über die Funktion `Time` ein.

387 Aktuellen Datensatz ans Ende kopieren

Im folgenden Beispiel wird die Tastenkombination Strg + Y im Feld `Artikelname` des Formulars `Artikel2` verwendet, um den aktuell aktiven Datensatz 1:1 ans Ende zu kopieren. Die Artikelnummer wird jedoch noch wie gewünscht selbst von Access vergeben. Legen Sie das Makro aus Listing 468 hinter das Feld `Artikelname`.

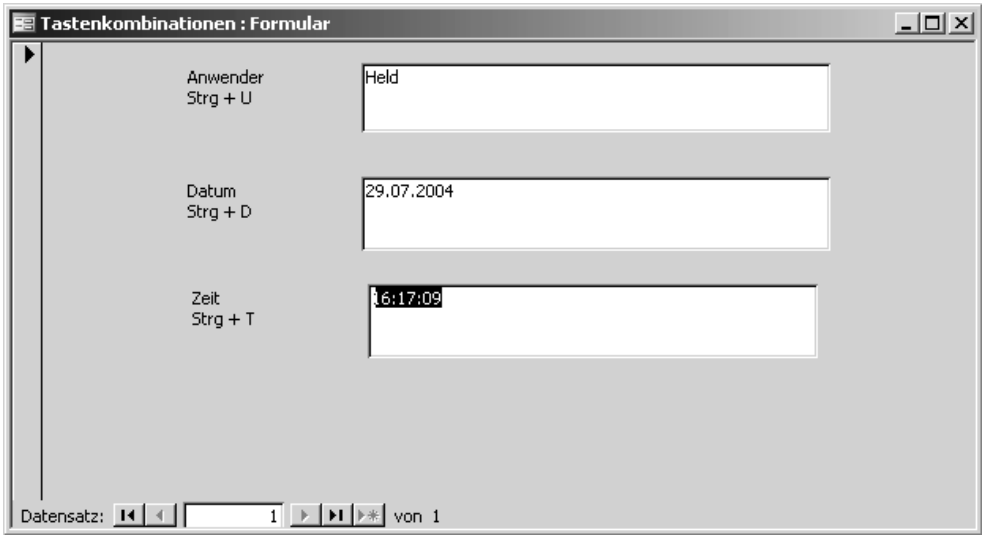


Abbildung 295: Schnellere Eingabe durch Tastenkombinationen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Artikelname_KeyDown(KeyCode As Integer, Shift As Integer)
   Dim DBS As Recordset

   'Aktuellen Satz kopieren
   If (Shift = acCtrlMask) And KeyCode = Asc("Y") Then
      Set DBS = Me.RecordsetClone

      If DBS.RecordCount > 0 Then
         DBS.AddNew
         DBS!Artikelname = Me.Artikelname
         DBS![Lieferanten-Nr] = Me.[Lieferanten-Nr]
         DBS![Kategorie-Nr] = Me.[Kategorie-Nr]
         DBS!Liefereinheit = Me.Liefereinheit
         DBS!Einzelpreis = Me.Einzelpreis
         DBS!Lagerbestand = Me.Lagerbestand
         DBS!BestellteEinheiten = Me.BestellteEinheiten
         DBS!Mindestbestand = Me.Mindestbestand
         DBS!Auslaufartikel = Me.Auslaufartikel
         DBS.Update
      End If
   End If
End Sub
```

Listing 468: Aktuellen Datensatz ans Ende kopieren

Erstellen Sie zunächst mithilfe der Methode `RecordsetClone` einen »Klon« vom aktuell eingestellten Datensatz. Danach fügen Sie einen neuen Datensatz über die Methode `AddNew` im Formular ein und füllen diesen mit den Werten aus dem Klon. Anschließend übertragen Sie die Werte aus dem Formular in den neuen Datensatz. Über die Methode `Update` speichern Sie den neuen Datensatz.

Hinweis

Die Tastenkombination Strg + Y funktioniert nur, wenn der Mauszeiger im Feld `Artikelname` des Formulars `Artikel2` steht.

388 Doppelte Eingaben verhindern

Bei der Erfassung von neuen Kunden im Formular `Kunden` haben Sie beschlossen, nur noch eindeutige Firmennamen einzugeben. Dabei soll Access Sie unterstützen und sofort eine Meldung auf dem Bildschirm erscheinen lassen, wenn Sie im Feld `Firma` einen `Kunden` eingeben, der bereits in der verknüpften Tabelle `Kunden` gespeichert ist.

Setzen Sie für diese Aufgabe das Ereignis `BeforeUpdate` des Felds `Firma` ein, welches Sie in Listing 469 sehen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=====================================================

Private Sub Firma_BeforeUpdate(Cancel As Integer)
   If (Not IsNull(DLookup("[Firma]", "Kunden", "[Firma] ='" _
      & Me!Firma & "'"))) Then
      MsgBox "Diese Firma wurde in der Tabelle schon angelegt!"
      Cancel = True
      Me!Firma.Undo
   End If
End Sub
```

Listing 469: Dubletten-Check bei der Eingabe vornehmen

Prüfen Sie mithilfe der Methode `DLookup`, ob der eingegebene Firmenname bereits in der verknüpften Tabelle `Kunden` erfasst wurde. Diese Überprüfung kann natürlich entfallen, wenn Sie in dem Feld noch keinen Firmennamen erfasst haben. Dafür sorgt die Funktion `IsNull`, die einen Wert vom Typ `Boolean` – nämlich `True` – zurückgibt, wenn das Feld leer ist. Befindet sich der eingegebene Firmenname bereits in der verknüpften Tabelle, dann können Sie das Argument `Cancel` einsetzen, welches Sie auf den Wert `True` setzen, um die Speicherung des Kunden abzubrechen.

389 Eingaben vorwegnehmen

Das Ereignis `BeforeUpdate` lässt sich auch verwenden, um einige Formularfelder automatisch füllen zu lassen. So können Sie beispielsweise bei der Eingabe der Postleitzahl den dazugehörigen Ort, die Region und das Land automatisch in die dafür zur Verfügung stehenden Formularfelder einfügen.

Im folgenden Beispiel aus Listing 470 wird im Formular `Kunden2` das Feld `PLZ` überwacht. Sobald dort eine Eingabe vorgenommen wird, wird im bisherigen Kundenstamm nachgesehen, ob diese Postleitzahl bereits verwendet wurde. Wenn ja, dann werden der dazugehörige Ort, die Region und das Land an den neuen Datensatz vererbt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden2
'=================================================

Private Sub PLZ_BeforeUpdate(Cancel As Integer)
   Me!Ort = (DLookup("[Ort]", "Kunden", "[PLZ] ='" & Me!PLZ & "'"))
   Me!Region = (DLookup("[Region]", "Kunden", "[PLZ] ='" & Me!PLZ & "'"))
   Me!Land = (DLookup("[Land]", "Kunden", "[PLZ] ='" & Me!PLZ & "'"))
End Sub
```

Listing 470: Über die Postleitzahl wird im Kundenstamm nach weiteren Informationen gesucht.

Prüfen Sie mithilfe der Methode `DLookup`, ob die eingegebene Postleitzahl bereits in der verknüpften Tabelle `Kunden` erfasst wurde. Befindet sich die eingegebene Postleitzahl bereits in der verknüpften Tabelle, dann wenden Sie abermals die Methode `DLookup` an, um die Textfelder `Ort`, `Region` sowie das Kombinationsfeld `Land` zu füllen.

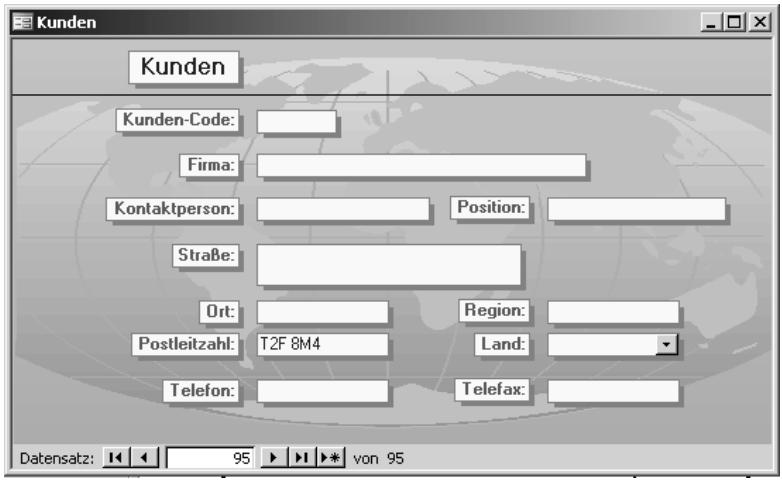


Abbildung 296: Die Postleitzahl wurde erfasst.

Und mit der Taste ⇆ ins nächste Feld positioniert. Dieser Vorgang löst das Ereignis `BeforeUpdate` aus.

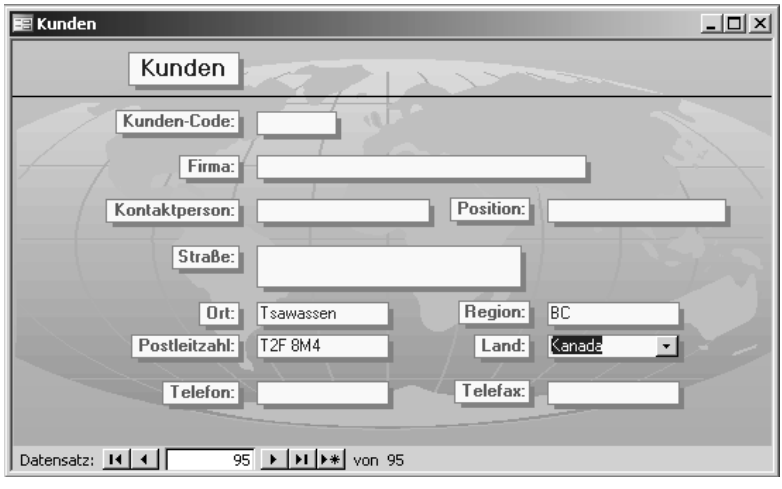


Abbildung 297: Die Felder Ort, Region und Land wurden automatisch gefüllt.

390 Löschen von Feldern verhindern

Mithilfe des Ereignisses `BeforeUpdate` können Sie Ihre Anwender auch davon abhalten, bestimmte Felder im Formular `Lieferanten` zu löschen.

Im folgenden Ereignismakro aus Listing 471 wird verhindert, dass bereits erfasste Werte im Textfeld `Kontaktperson` gelöscht werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Kontaktperson_BeforeUpdate(Cancel As Integer)
   If IsNull(Me.Kontaktperson) Then
      MsgBox "Sie dürfen keine Kontaktperson löschen!", vbCritical
      Me!Kontaktperson.Undo
      Cancel = True
   End If
End Sub
```

Listing 471: Die Löschung von Daten verhindern

Mithilfe der Funktion `IsNull` finden Sie heraus, ob das Textfeld `Kontaktperson` leer ist. Wenn ja, dann wurde die bereits erfasste Information gelöscht oder es wurde bei einer Neuanlage eines Satzes keine Kontaktperson angegeben. Im ersten Fall geben Sie eine Bildschirmmeldung aus und widerrufen die Löschung mithilfe der Methode `Undo`. Setzen Sie das Argument

`Cancel` auf den Wert `True`, damit die Löschung des Textfelds `Kontaktperson` in der verknüpften Tabelle `Lieferanten` nicht durchgeführt wird.

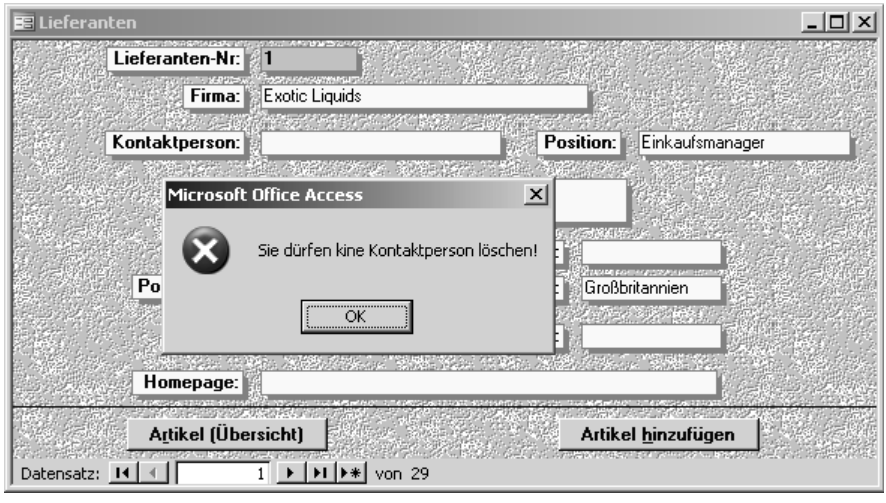


Abbildung 298: Beim Löschen einer Kontaktperson wird eine Meldung angezeigt.

Sobald Sie die Schaltfläche OK anklicken, wird der ursprüngliche Wert wieder im Formular eingefügt. Sie haben keine Möglichkeit, einen Datensatz ohne Kontaktperson zu speichern.

Hinweis

Bei der Neuanlage eines Satzes hilft Ihnen dieses Ereignis nicht wirklich weiter. Dazu müssen Sie das Ereignis `Exit` einsetzen, welches im Anschluss beschrieben wird.

391 QuickInfos anzeigen

Im folgenden Beispiel aus Listing 472 wurde im Formular `Personal` auf der Registerkarte PERSÖNLICHE DATEN das Textfeld `Bemerkungen` mit einer QuickInfo ausgestattet. Diese QuickInfo wird angezeigt, sobald das Feld aktiviert wird. Selbstverständlich muss die unterstützende QuickInfo nicht angezeigt werden, wenn das Feld bereits gefüllt ist.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap08
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=================================================
```

Listing 472: Eine QuickInfo anzeigen

```
Private Sub Bemerkungen_Enter()
   Dim strText As String

   strText = "Bitte eigene Informationen angeben!"

   If Me.Bemerkungen <> "" Then
      Me.Bemerkungen.ControlTipText = strText
   End If
End Sub
```

Listing 472: Eine QuickInfo anzeigen (Forts.)

Mithilfe der Eigenschaft `ControlTipText` können Sie eine QuickInfo für ein Steuerelement anzeigen. Diese Info wird jedoch nur angezeigt, wenn das Feld `Bemerkungen` noch nicht gefüllt ist.

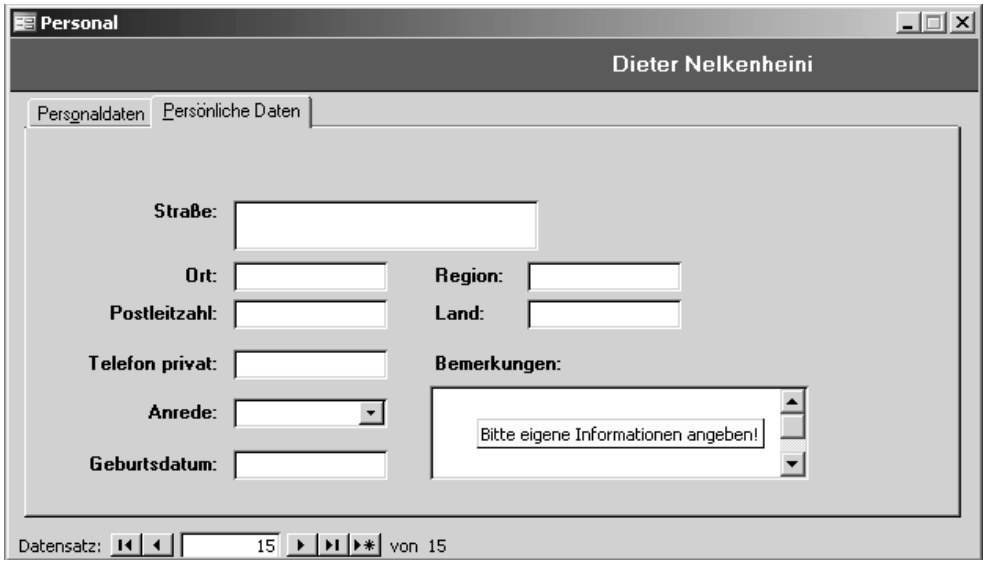


Abbildung 299: QuickInfo bei Aktivieren des Felds anzeigen

392 Reihenfolge beim Öffnen und Schließen eines Formulars

Wenn Sie ein Formular zum ersten Mal öffnen, treten die folgenden Ereignisse in der angegebenen Reihenfolge ein:

`Open`, `Load`, `Resize`, `Activate`, `Current`

Möchten Sie die Ereignisse über das Eigenschaftenfenster einstellen, lauten die einzustellenden Ereignisse wie folgt:

`Beim Öffnen`, `Beim Laden`, `Bei Größenänderung`, `Bei Aktivierung`, `Beim Anzeigen`

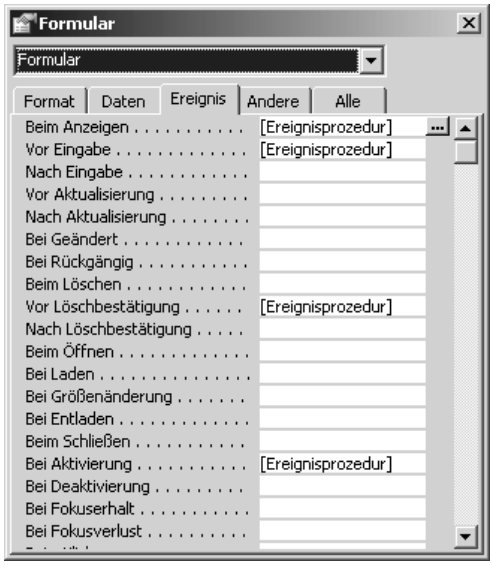


Abbildung 300: Formularereignisse über das Eigenschaftenfenster einstellen

Wenn Sie ein Formular schließen, treten die folgenden Ereignisse in der angegebenen Reihenfolge ein:

```
Unload, Deactivate, Close
```

Um diese Ereignisse im Eigenschaftenfenster auf der Registerkarte EREIGNISSE einzusehen, klicken Sie folgende Ereignisse an:

`Bei Entladen`, `Bei Deaktivierung`, `Beim Schließen`

393 Aktivierreihenfolge bei Steuerelementen

Wenn Sie beispielsweise über die Taste ⇆ von einem Textfeld in ein anderes wechseln, werden folgende Ereignisse nacheinander ausgelöst:

```
Enter, GotFocus, Exit, LostFocus
```

Diese Ereignisse heißen im Eigenschaftenfenster auf der Registerkarte EREIGNISSE wie folgt:

`Beim Hingehen`, `Bei Fokuserhalt`, `Beim Verlassen`, `Bei Fokusverlust` (siehe Abbildung 301)

394 Reihenfolge der Aktualisierungsereignisse

Für die Aktualisierung von Ereignissen stehen zwei Ereignisse in unmittelbarer Reihenfolge untereinander: `BeforeUpdate` und `AfterUpdate`.

Diese Ereignisse heißen im Eigenschaftenfenster `Vor Aktualisierung` und `Nach Aktualisierung`.

Wenn Sie in ein Steuerelement eines Formulars neue oder geänderte Daten eingeben und dann zu einem anderen Datensatz wechseln oder den Datensatz mit dem Befehl DATENSATZ

Abbildung 301: Die Steuerelementereignisse

SPEICHERN aus dem Menü DATENSÄTZE speichern, tritt zunächst das `BeforeUpdate`-Ereignis des Steuerelements und unmittelbar danach das `BeforeUpdate`-Ereignis des Formulars ein.

395 Eine komplette Kette von Ereignissen

Stellen Sie sich jetzt einmal vor, Sie müssten ein Formular öffnen, dann ein Textfeld anspringen, dort einen Wert eingeben und mit der Taste ⇆ zum nächsten Formularfeld wechseln. Die komplette Reihenfolge der Ereignisse sieht so aus:

`Open`, `Load`, `Enter`, `GotFocus`, `KeyDown`, `KeyPress`, `Change`, `KeyUp`, `BeforeUpdate`, `AfterUpdate`, `Exit`, `LostFocus`

Möchten Sie die Ereignisse über das Eigenschaftenfenster einsehen, lauten die Ereignisse wie folgt:

`Beim Öffnen`, `Bei Laden`, `Beim Hingehen`, `Bei Fokuserhalt`, `Bei Maustaste Ab`, `Bei Taste`, `Bei Geändert`, `Bei Taste Auf`, `Vor Aktualisierung`, `Nach Aktualisierung`, `Beim Verlassen`, `Bei Fokusverlust`.

VBE und Security

Über die VBE-Programmierung können Sie direkt auf Module, Makros, Berichte und Formulare in einer Datenbank zugreifen. Dies werden Sie dann tun, wenn Sie beispielsweise Ihren Quellcode in Textdateien sichern oder auch weitergeben möchten. Ebenso können Sie VBE zum Säubern einer Datenbank von Modulen, Ereignissen und Formularen verwenden. Außerdem setzen Sie VBE ein, um Datenbanken gezielt mit bestimmten Makros zu bestükken bzw. bereits vorhandenen Quellcode an einer bestimmten Stelle zu aktualisieren oder zu ersetzen.

Bevor aber auf das Thema VBE eingegangen wird, noch ein paar Worte zum Schutz von Quellcodes und Datenbanken sowie zum Einrichten und Verwalten von Arbeitsgruppen.

Um Ihren Quellcode zu schützen, bieten sich Ihnen mehrere Möglichkeiten:

1. Hinterlegen eines Kennworts, mit dem Sie die Anzeige des Quellcodes verhindern können
2. Umwandeln der Datenbank in ein Format, bei dem der Quellcode nur noch kompiliert vorliegt und weder verändert noch eingesehen werden kann
3. Verschlüsseln Ihrer Datenbank

396 Kennwort für die Anzeige des Quellcodes anlegen

Möchten Sie Ihren Quellcode vor neugierigen Augen schützen, dann vergeben Sie ein Kennwort. Dabei befolgen Sie die folgenden Arbeitsschritte:

1. Wechseln Sie in die Entwicklungsumgebung von Access.
2. Klicken Sie Ihr VBA-Projekt im Projekt-Explorer mit der rechten Maustaste an und wählen Sie aus dem Kontextmenü den Befehl EIGENSCHAFTEN VON.
3. Wechseln Sie auf die Registerkarte SCHUTZ.
4. Aktivieren Sie das Kontrollkästchen PROJEKT FÜR DIE ANZEIGE SPERREN.
5. Erfassen und bestätigen Sie ein Kennwort in den unteren beiden Eingabefeldern.
6. Bestätigen Sie Ihre Einstellung mit OK.

Damit diese Einstellung wirksam wird, schließen Sie Ihre Datenbank und öffnen Sie diese direkt im Anschluss. Wechseln Sie über die Tastenkombination Alt + F11 in die Entwicklungsumgebung und versuchen Sie, Ihr VBA-Projekt zu öffnen. Der Code wird nur angezeigt, wenn Sie das richtige Kennwort eingeben.

Abbildung 302: Kennwort festlegen

397 Datenbank ohne Quellcode speichern

Wenn Sie sich entschließen, eine Datenbank ohne Quellcode zu speichern, oder liegt der Quellcode kompiliert vor, müssen Sie Ihre Datenbank im MDE-Format abspeichern. Wenn Sie eine Datenbank in diesem Format speichern, dann wird eine Kopie Ihrer Datenbank erstellt, die keinen VBA-Quellcode enthält und zudem kleiner ist als die Originaldatenbank. Die Codeanweisungen liegen nun kompiliert vor und können weiterhin und sogar noch schneller ausgeführt werden. Es besteht jedoch keine Möglichkeit, den Quellcode anzuzeigen bzw. zu bearbeiten.

Als kleiner Nebeneffekt haben Sie bei diesem Format auch keine Möglichkeit mehr, die Entwurfsansicht von Formularen und Berichten aufzurufen. Kurz gesagt, es sind auch die Formulare und Berichte vor einer Veränderung sicher. Nebenbei haben Sie auch keine Chance, neue Formulare oder Berichte in die MDE-Datei einzufügen. Selbst das Einfügen von Modulen ist hierbei nicht möglich. Aus einer MDE-Datei können auch keine Tabellen, Abfragen, Formulare oder Berichte importiert bzw. exportiert werden, so dass Sie auch hierbei auf der sicheren Seite sind.

Um eine MDE-Datei aus Ihrer Datenbank zu erzeugen, befolgen Sie die nächsten Arbeitsschritte:

1. Stellen Sie zuvor sicher, dass kein anderer Anwender diese Datenbank im Zugriff hat.
2. Rufen Sie aus dem Menü EXTRAS den Befehl DATENBANK-DIENSTPROGRAMME/MDE-DATEI ERSTELLEN auf.
3. Wählen Sie im Dialog ALS MDE ZU SPEICHERNDE DATENBANK eine Datenbank aus, die Sie umwandeln möchten.
4. Bestätigen Sie diesen Vorgang mit Klick auf MDE ERSTELLEN.

398 Datenbanken verschlüsseln

Möchten Sie eine geschützte Datenbank vor dem unbefugten Zugriff durch andere Anwender bewahren, haben Sie die Möglichkeit, die Datenbank zu verschlüsseln. Durch die Verschlüsselung wird die Datenbank unlesbar und so gegen ein unbefugtes Anzeigen oder Verwenden geschützt.

Damit die Verschlüsselung überhaupt etwas bringt, muss die Datenbank vorher durch ein Kennwort geschützt werden. Befolgen Sie die nächsten Arbeitsschritte, um eine Datenbank zu verschlüsseln:

1. Sorgen Sie dafür, dass die Datenbank, die Sie verschlüsseln möchten, momentan von niemandem geöffnet ist.
2. Wählen Sie aus dem Menü EXTRAS den Befehl SICHERHEIT/DATENBANK VER-/ENTSCHLÜSSELN.
3. Geben Sie im Dialogfeld DATENBANK VERSCHLÜSSELN ALS im Feld DATEINAME den Namen der neuen verschlüsselten Datei an.
4. Klicken Sie auf die Schaltfläche SPEICHERN.

Durch das Verschlüsseln wird die Leistungsfähigkeit einer Datenbank um bis zu 15% reduziert. Des Weiteren kann eine verschlüsselte Datenbank mit Programmen wie DriveSpace, WinZip oder PKZIP nicht weiter komprimiert werden. Wenn Sie eine verschlüsselte Datenbank komprimieren, ändert sich die Größe der Datenbank daher nicht.

399 Access-Lösung mithilfe von Startparametern absichern

Startoptionen ermöglichen es Ihnen, den Zugriff auf bestimmte Standardmenüs und Symbolleisten, auf das Datenbankfenster sowie Spezialtasten weitreichend einzuschränken.

Im folgenden Beispiel möchten Sie das Formular `Artikel` gleich nach dem Start der Datenbank einblenden und bestimmte Aktionen verhindern.

1. Wählen Sie aus dem Menü EXTRAS den Befehl START.
2. Im Drop-down-Menü FORMULAR/SEITE ANZEIGEN stellen Sie das Formular `Artikel` ein.
3. Klicken Sie im Feld MENÜLEISTE auf den Namen der gewünschten Menüleiste.
4. Deaktivieren Sie die Kontrollkästchen UNBESCHRÄNKTE MENÜS ANZEIGEN, STANDARDKONTEXTMENÜS ZULASSEN, DATENBANKFENSTER ANZEIGEN, EINGEBAUTE SYMBOLLEISTEN ZULASSEN, SYMBOLLEISTEN- UND MENÜÄNDERUNGEN ZULASSEN und ACCESS-SPEZIALTASTEN VERWENDEN.
5. Bestätigen Sie Ihre Einstellung mit OK.
6. Schließen Sie jetzt Ihre Datenbank.

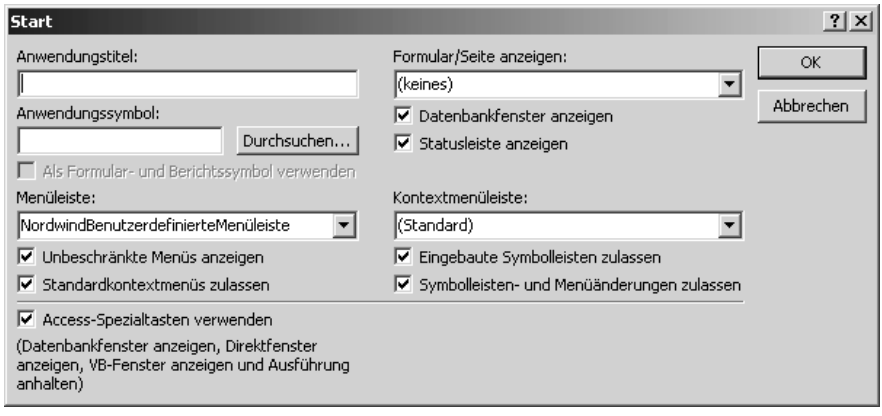


Abbildung 303: Startoptionen einstellen

Nach dem erneuten Starten von Access wird das Formular `Artikel` direkt geöffnet. Sie haben jetzt beispielsweise keine Möglichkeit mehr, über die Tastenkombination Alt + F11 in die Entwicklungsumgebung zu gelangen.

400 Schützen einer Datenbank über ein Kennwort

Eine weitere Möglichkeit, eine Datenbank zu schützen, besteht darin, diese mit einem Zugangskennwort zu belegen. Bevor Sie diese Aufgabe vornehmen, sollten Sie sicherstellen, dass die Datenbank momentan nicht von einem weiteren Anwender geöffnet ist.

Um eine Datenbank mit einem Zugriffskennwort zu schützen, gehen Sie wie folgt vor:

1. Schließen Sie die Datenbank.
2. Starten Sie Access erneut und wählen Sie aus dem Menü DATEI den Befehl ÖFFNEN.
3. Im Dialogfeld ÖFFNEN markieren Sie die Datenbank, die Sie über ein Kennwort absichern möchten, und klicken auf den Pfeil der Schaltfläche ÖFFNEN.
4. Wählen Sie aus dem Kontextmenü der Schaltfläche den Befehl EXKLUSIV ÖFFNEN.
5. Nachdem die Datenbank geöffnet wurde, wählen Sie aus dem Menü EXTRAS den Befehl SICHERHEIT/DATENBANKKENNWORT FESTLEGEN.
6. Geben Sie im Dialogfeld DATENBANKKENNWORT FESTLEGEN ein Kennwort ein und bestätigen Sie dieses.
7. Klicken Sie danach auf die Schaltfläche OK.

Wenn nun in Zukunft versucht wird, die so geschützte Datenbank zu öffnen, muss das Kennwort eingegeben werden. Achten Sie darauf, dass Sie sich das Kennwort aufschreiben und an einer sicheren Stelle deponieren. Es gibt standardmäßig keine Möglichkeit, eine Datenbank ohne das Kennwort zu öffnen.

Geschützte Datenbank per VBA öffnen (DAO)

In Zusammenhang mit dem Öffnen geschützter Datenbanken besteht außerdem die interessante Möglichkeit, diese Aufgabe über ein VBA-Makro auszuführen. Das Makro aus Listing 473 öffnet die geschützte Datenbank *Nordwind.mdb*, die sich im Verzeichnis *C:\Eigene Dateien* befindet.

Als kleine Vorarbeit für diese Aufgabe aktivieren Sie eine zusätzliche Bibliothek, indem Sie wie folgt vorgehen:

1. Wechseln Sie in die Entwicklungsumgebung.
2. Wählen Sie aus dem Menü EXTRAS den Befehl VERWEISE.

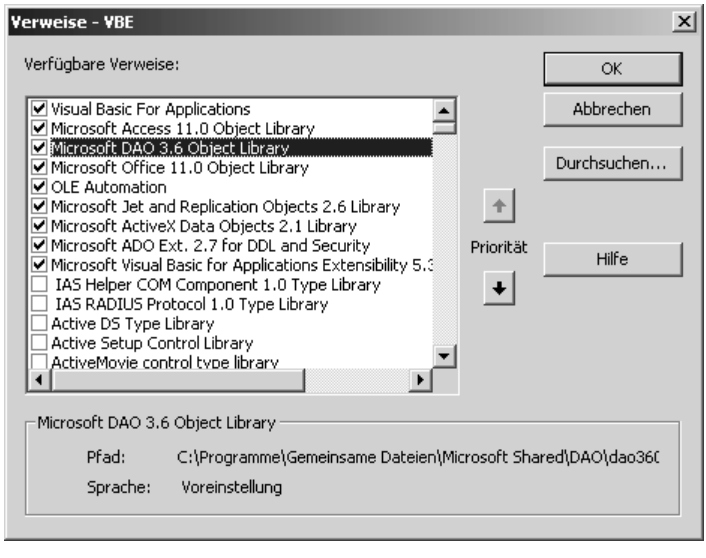


Abbildung 304: Die DAO-Bibliothek aktivieren

3. Aktivieren Sie die Bibliothek MICROSOFT DAO 3.6 OBJECT LIBRARY.
4. Bestätigen Sie Ihre Einstellung mit OK.

Erfassen Sie jetzt das Makro aus Listing 473:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================
```

Listing 473: Geschützte Datenbank öffnen (DAO)

```
Sub DatenbankMitKennwortÖffnen()
   Dim accObj As Object
   Dim strPass As String
   Dim db As Database

   Set accObj = CreateObject("Access.Application.11")
   strPass = ";PWD=test;"
   On Error GoTo fehler
   Set db = accObj.DBEngine.OpenDatabase _
      ("C:\Eigene Dateien\Nordwind.mdb", False, False, sPass)

   accObj.OpenCurrentDatabase "C:\Eigene Dateien\Nordwind.mdb"
   db.Close

   Set db = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 473: Geschützte Datenbank öffnen (DAO) (Forts.)

Im ersten Schritt erstellen Sie mithilfe der Methode `CreateObject` ein neues Access-2003-Objekt. Danach setzen Sie das Kennwort in der Variablen `strPass` zusammen. Die DBEngine ist das Objekt der obersten Ebene in einem DAO-Objektmodell. Das `DBEngine`-Objekt enthält und steuert alle anderen Objekte in der DAO-Objekthierarchie. Über dieses Objekt können Sie die Methode `OpendDatabase` einsetzen, um eine angegebene Datenbank in einem `Workspace`-Objekt zu öffnen. Dabei wird ein Verweis auf das `Database`-Objekt zurückgegeben, das diese Datenbank repräsentiert. Geben Sie in dieser Methode als Argument Ihr Kennwort ein, welches Sie vorher Ihrer Datenbank zugewiesen haben. Öffnen Sie die Datenbank letztendlich über die Methode `OpenCurrentDatabase` in dem neuen Access-Objekt und schließen Sie das Database-Objekt über die Methode `Close`.

Geschützte Datenbank per VBA öffnen (ADO)

Selbstverständlich können Sie eine geschützte Datenbank auch über ADO öffnen und verarbeiten. Im folgenden Beispiel aus Listing 474 wird die Datenbank *Nordwind.mdb* im Hintergrund geöffnet, die Tabelle `ARTIKEL` geöffnet, die Artikelnamen werden ausgelesen, die Tabelle und die Datenbank danach wieder geschlossen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================
```

Listing 474: Geschützte Datenbank öffnen (ADO)

```
Sub DatenbankÖffnenADO()
   Dim conn As ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set conn = New ADODB.Connection

   With conn
      .Provider = "Microsoft.Jet.OLEDB.4.0"
      .Properties("Jet OLEDB:Database Password") = "test"
      .Mode = adModeReadWrite
      .Open "C:\Eigene Dateien\Nordwind.mdb"
   End With

   Set DBS = New ADODB.Recordset
   DBS.CursorType = adOpenDynamic
   DBS.LockType = adLockPessimistic
   DBS.Open "Artikel", conn, , , adCmdTable

   Do While Not DBS.EOF
      Debug.Print DBS!Artikelname
      DBS.MoveNext
   Loop

   DBS.Close
   conn.Close
   Set conn = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 474: Geschützte Datenbank öffnen (ADO) (Forts.)

Über die Eigenschaft `Provider` geben Sie den Namen des Providers für das `Connection`-Objekt an. Über die `Properties`-Auflistung übergeben Sie der Verbindung Ihr Kennwort. Über die Eigenschaft `Mode` können Sie die Berechtigungen für das Ändern von Daten im `Connection`-Objekt `conn` bestimmen. Dabei haben Sie die folgenden Möglichkeiten aus Tabelle 137:

Konstante	Beschreibung
`adModeUnknown`	Diese Standardeinstellung gibt an, dass die Berechtigungen noch nicht festgelegt wurden oder nicht bestimmt werden können.
`adModeRead`	Gibt schreibgeschützte Berechtigungen an.
`adModeWrite`	Gibt Nur-Lese-Berechtigungen an.

Tabelle 137: Die Konstanten der Eigenschaft Mode

Konstante	Beschreibung
adModeReadWrite	Gibt Schreib-/Leseberechtigungen an.
adModeShareDenyRead	Verhindert das Öffnen einer Verbindung mit Leseberechtigungen durch andere Personen.
adModeShareDenyWrite	Verhindert das Öffnen einer Verbindung mit Schreibberechtigungen durch andere Personen.
adModeShareExclusive	Verhindert das Öffnen einer Verbindung durch andere Personen.
adModeShareDenyNone	Verhindert das Öffnen einer Verbindung mit beliebigen Berechtigungen durch andere Personen.

Tabelle 137: Die Konstanten der Eigenschaft Mode (Forts.)

Mithilfe der Methode `Open` öffnen Sie die Datenbank. Im nächsten Schritt erzeugen Sie ein neues `Recordset`-Objekt und geben über die Eigenschaft `CursorType` an, welcher Cursor beim Öffnen des `Recordset`-Objekts verwendet werden soll. Diese Eigenschaft legen Sie über Cursor-Konstanten fest, die Sie in der Tabelle 138 einsehen können.

Konstante	Beschreibung
adOpenForwardOnly	Vorwärts-Cursor (Voreinstellung). Dieser Cursor ist identisch mit einem statischen Cursor, mit dem Unterschied, dass ein Blättern durch die Datensätze nur in Vorwärtsrichtung möglich ist. Dies verbessert die Leistung in Situationen, in denen nur ein einziger Durchlauf durch ein `Recordset` durchgeführt werden muss.
adOpenKeyset	Cursor vom Typ `Keyset`. Ähnelt einem dynamischen Cursor, mit dem Unterschied, dass von anderen Personen hinzugefügte Datensätze nicht angezeigt werden können, obwohl ein Zugriff auf von anderen Benutzern gelöschte Datensätze von Ihrem `Recordset` aus nicht möglich ist. Von anderen Personen vorgenommene Datenänderungen können weiterhin angezeigt werden.
adOpenDynamic	Dynamischer Cursor. Von anderen Personen vorgenommene Zusätze, Änderungen und Löschvorgänge können angezeigt werden. Alle Bewegungsarten durch den Datensatz sind zulässig, außer Lesezeichen (sofern der Provider diese nicht unterstützt).
adOpenStatic	Statischer Cursor. Eine statische Kopie einer Gruppe von Datensätzen, anhand deren Daten gesucht und Berichte generiert werden können. Von anderen Benutzern vorgenommene Zusätze, Änderungen oder Löschvorgänge können nicht angezeigt werden.

Tabelle 138: Die Konstanten der Eigenschaft CursorType

Über die Eigenschaft `LockType` legen Sie die Art der Sperrung fest, die während des Bearbeitens auf Datensätze angelegt wird. Auch bei dieser Eigenschaft können Sie die Sperrung über Konstanten festlegen, die Sie in Tabelle 139 sehen können.

Konstante	Beschreibung
adLockReadOnly	Voreinstellung. Schreibgeschützt – Sie können die Daten nicht ändern.
adLockPessimistic	Vollständiges Sperren, Datensatz für Datensatz – der Provider führt die notwendigen Schritte aus, um das erfolgreiche Bearbeiten der Datensätze sicherzustellen, üblicherweise, indem er Datensätze in der Datenquelle sofort beim Bearbeiten sperrt.
adLockOptimistic	Teilweises Sperren, Datensatz für Datensatz – der Provider verwendet teilweises Sperren, indem er Datensätze nur sperrt, wenn Sie die Update-Methode aufrufen.
adLockBatchOptimistic	Teilweise Stapelaktualisierungen – erforderlich für den Stapelaktualisierungsmodus, im Gegensatz zum Modus für sofortige Aktualisierung.

Tabelle 139: Die Konstanten der Eigenschaft LockType

Öffnen Sie die Tabelle ARTIKEL, indem Sie die Methode `Open` einsetzen. Über die Konstante `adCmdTable` geben Sie an, dass ADO eine SQL-Abfrage generieren soll, um alle Zeilen der in `Source` benannten Tabelle zurückzugeben.

Durchlaufen Sie im Anschluss eine Schleife, die so lange durchlaufen wird, bis die Eigenschaft `EOF` erfüllt ist. Dann ist der letzte Satz erreicht. Innerhalb der Schleife geben Sie die Artikelnamen der Tabelle `Artikel` im Direktbereich von Access aus. Positionieren Sie mit der Methode `MoveNext` jeweils auf den nächsten Datensatz in der Tabelle.

Schließen Sie danach die Tabelle und die Datenbank über die Methode `Close` und heben Sie die Objektverknüpfung auf.

401 Datenbankkennwort ändern

Im nächsten Beispiel ändern Sie ein Kennwort für eine Datenbank über ein VBA-Makro. Dabei entfallen die sonst üblichen Arbeitsschritte zum Ändern eines Kennworts. Stellen Sie sicher, dass der Objektverweis auf die DAO-Bibliothek gesetzt ist, und starten Sie danach das Makro aus Listing 475.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub KennwortÄndern()
   Dim db As DAO.Database
   Dim strK As String
```

Listing 475: Datenbankkennwort ändern (DAO)

```
    Dim strAltPwd As String
    Dim strNeuPwd As String

    strK = ";pwd=test"
    strAltPwd = "test"
    strNeuPwd = "testNeu"

    On Error GoTo fehler
    Set db = OpenDatabase (Name:="C:\Eigene Dateien\Nordwind.mdb", _
       Options:=True, ReadOnly:=False, Connect:=strK)

    With db
       .NewPassword strAltPwd, strNeuPwd
       .Close
    End With

    Set db = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 475: Datenbankkennwort ändern (DAO) (Forts.)

Basteln Sie sich im ersten Schritt einen String, der das Kürzel `PWD` sowie das aktuelle Kennwort Ihrer Datenbank enthält. Danach definieren Sie zwei Strings, in denen Sie das alte sowie das neue Passwort ablegen.

Öffnen Sie die Datenbank über die Methode `OpenDatabase` und setzen Sie danach die Methode `NewPassword` ein, um ein neues Kennwort für die Datenbank einzustellen.

Die Zeichenfolgen `strAltPwd` und `strNeuPwd` können maximal 14 Zeichen lang sein und alle Zeichen außer dem ASCII-Zeichen 0 enthalten.

Sie können ein Kennwort löschen, indem Sie für die Zeichenfolge `strNeuPwd` die Zeichen "" verwenden.

402 Arbeitsgruppe anlegen

Wenn Ihnen der normale Zugriff über die Kennworteingabe nicht ausreicht, können Sie in Access auch eine professionelle Benutzerverwaltung einrichten und ganz gezielt bestimmen, welcher Anwender welche Aufgaben durchführen darf. Dabei können Sie entweder einzelnen Benutzern Rechte einräumen oder Arbeitsgruppen definieren und damit allen Benutzern einer Arbeitsgruppe bestimmte Rechte vergeben.

Die Anlage einer neuen Gruppe kann manuell erfolgen, aber auch über ein VBA-Makro erledigt werden. Sehen Sie sich dazu das Makro aus Listing 476 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub NeueGruppeErzeugen()
   Dim WS As Workspace
   Dim NeueGruppe As Group
   Dim NeueGrT As Group

   Set WS = DBEngine.Workspaces(0)
   With WS
      Set NeueGruppe = .CreateGroup("NeueGruppe", "XYZ09012892")
      .Groups.Append NeueGruppe
      Set NeueGrT = .Users("Admin").CreateGroup("NeueGruppe")
      .Users("Admin").Groups.Append NeueGrT
   End With
End Sub
```

Listing 476: Neue Arbeitsgruppe anlegen (DAO)

Setzen Sie die Methode `CreateGroup` ein, um eine neue Gruppe zu erzeugen. Dabei müssen Sie den Namen sowie eine eindeutige ID angeben. Über die Methode `Append` hängen Sie diese neue Gruppe an. Fügen Sie danach der neuen Gruppe einen Administrator als Mitglied hinzu, indem Sie das entsprechende `Group`-Objekt mithilfe der Methode `CreateGroup` erstellen und der `Groups`-Auflistung des Benutzers über die Methode `Append` hinzufügen.

Wenn Sie diese Aufgabe mit ADO lösen möchten, binden Sie vorher in der Entwicklungsumgebung über das Menü EXTRAS und den Befehl VERWEISE die Bibliothek MICROSOFT ADO EXT 2.5 für Office 2002 bzw. MICROSOFT ADO EXT 2.7 für Office 2003 ein. Starten Sie danach das Makro aus Listing 477.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub GruppeAnlegenADO()
   Dim KDB As ADOX.Catalog

   On Error GoTo fehler
   Set KDB = New ADOX.Catalog

   With KDB
      .ActiveConnection = CurrentProject.Connection
      .Groups.Append "ATeam"
```

Listing 477: Neue Arbeitsgruppe anlegen (ADO)

```
    End With

    Set KDB = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 477: Neue Arbeitsgruppe anlegen (ADO) (Forts.)

Erstellen Sie eine neue Gruppe, indem Sie die Methode `Append` einsetzen. Geben Sie dabei den Namen der Gruppe an.

403 Arbeitsgruppe entfernen (DAO)

Um eine Gruppe wieder zu löschen, können Sie das Makro aus Listing 478 verwenden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=================================================

Sub GruppeEntfernen()
   Dim WS As Workspace
   Dim strGruppe As String

   On Error GoTo fehler
   Set WS = DBEngine.Workspaces(0)
   strGruppe = WS.Groups("NeueGruppe").Name
   WS.Groups.Delete strGruppe
   Exit Sub


   fehler:
      MsgBox "die Gruppe " & strGruppe & _
         " konnte nicht gefunden/gelöscht werden!"
End Sub
```

Listing 478: Arbeitsgruppe löschen (DAO)

Geben Sie den Namen der Gruppe in einer String-Variablen an und übergeben Sie diesen an die Methode `Delete`.

Den entsprechenden Code in ADO können Sie dem Listing 479 entnehmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub GruppeEntfernenADO()
   Dim KDB As ADOX.Catalog

   Set KDB = New ADOX.Catalog
   On Error GoTo fehler
   With KDB
      .ActiveConnection = CurrentProject.Connection
      .Groups.Delete "ATeam"
   End With

   Set catDB = Nothing
   Exit Sub


   fehler:
      MsgBox "Die Gruppe konnte nicht entfernt werden!"
End Sub
```

Listing 479: Arbeitsgruppe löschen (ADO)

Greifen Sie auf das Auflistungsobjekt `Groups` zu, in dem alle Gruppen verzeichnet sind. Setzen Sie dort die Methode `Delete` ein, um die Gruppe zu löschen.

404 Benutzer anlegen

Haben Sie die Gruppe angelegt, fügen Sie die Benutzer in die Gruppe mit dem Makro aus Listing 480 ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub NeuenBenutzerErzeugen()
   Dim WS As Workspace
   Dim Benutzer As User

   On Error GoTo fehler
   Set WS = DBEngine(0)
   Set Benutzer = WS.CreateUser("Walter", "1002", "Test")

   WS.Users.Append Benutzer
```

Listing 480: Neuen Benutzer anlegen (DAO)

```
    Benutzer.Groups.Append WS.CreateGroup("Users")
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 480: Neuen Benutzer anlegen (DAO) (Forts.)

Über die Methode `CreateUser` erstellen Sie einen neuen Benutzer. Dabei müssen Sie den Namen sowie eine eindeutige ID angeben. Optional können Sie noch ein Kennwort für den Benutzer angeben. Mithilfe der Methode `Append` hängen Sie diesen Benutzer endgültig an. Danach weisen Sie dem neuen Benutzer die Gruppe `Benutzer` zu.

Soll der Vorgang mit ADO gelöst werden, dann wenden Sie das Makro aus Listing 481 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub BenutzerAnlegenADO()
   Dim KDB As ADOX.Catalog

   On Error GoTo fehler
   Set KDB = New ADOX.Catalog

   With KDB
      .ActiveConnection = CurrentProject.Connection
      .Users.Append "Hero", "Test"
      .Groups("Users").Users.Append "Hero"
   End With

   Set KDB = Nothing
   Exit Sub

   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 481: Neuen Benutzer anlegen (ADO)

Öffnen Sie das `Catalog`-Objekt über die Verknüpfung mit der aktuellen Datenbank und fügen Sie der `Users`-Auflistung ein neues `User`-Objekt hinzu, indem Sie die Methode `Append` einsetzen. Hängen Sie danach das neue Benutzerkonto an die Standard-Benutzergruppe wiederum über die Methode `Append` an.

405 Benutzer entfernen

Möchten Sie einen Benutzer per VBA wieder entfernen, starten Sie das Makro aus Listing 482.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub BenutzerEntfernen()
   Dim strBenutzer As String
   Dim WS As Workspace

   On Error GoTo fehler
   Set WS = DBEngine(0)
   strBenutzer = WS.Users("Walter").Name
   WS.Users.Delete strBenutzer
   Exit Sub


   fehler:
      MsgBox "Der Benutzer konnte nicht gelöscht werden!"
End Sub
```

Listing 482: Benutzer entfernen (DAO)

Mithilfe der Methode `Delete` löschen Sie einen bereits angelegten Benutzer. Geben Sie dabei den Namen des Benutzers in einem String an.

Das Makro in ADO sehen Sie in Listing 483:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub BenutzerEntfernenADO()
   Dim KDB As ADOX.Catalog

   On Error GoTo fehler
   Set KDB = New ADOX.Catalog

   With KDB
      .ActiveConnection = CurrentProject.Connection
      .Users.Delete "Hero"
   End With
```

Listing 483: Benutzer entfernen (ADO)

```
    Set catDB = Nothing
    Exit Sub


    fehler:
       MsgBox "Der Benutzer konnte nicht entfernt werden!"
End Sub
```

Listing 483: Benutzer entfernen (ADO) (Forts.)

Setzen Sie die Methode `Delete` ein, um einen Benutzer zu löschen. Wenden Sie diese Methode innerhalb des Auflistungsobjekts `Users` an, in dem alle Benutzer verzeichnet sind.

406 Benutzer auslesen

Um zu erfahren, welche Benutzer überhaupt bereits angelegt wurden, können Sie das Auflistungsobjekt `Groups` anzapfen und die Namen der Benutzer im Direktbereich ausgeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub Benutzeranzeigen()
   Dim WS As Workspace
   Dim Benutzer As User

   Set WS = DBEngine(0)

   For Each Benutzer In WS.Users
      Debug.Print Benutzer.Name
   Next Benutzer
End Sub
```

Listing 484: Benutzer auslesen (DAO)

407 Berechtigungen zuweisen

Nachdem Sie sowohl die Gruppe(n) als auch den Benutzer angelegt haben, ordnen Sie die Benutzer einzelnen Gruppen zu. Mithilfe eines Makros können Sie gezielt Berechtigungen zuweisen. Im nächsten Beispiel aus Listing 485 weisen Sie dem Benutzer `Schmidt` eine spezielle Berechtigung für die Tabelle `Artikel` zu.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub BerechtigungenHinzufügenADO()
   Dim conn As New ADODB.Connection
   Dim KDB As New ADOX.Catalog

   On Error GoTo fehler
   conn.Provider = "Microsoft.Jet.OLEDB.4.0"
   Set KDB.ActiveConnection = CurrentProject.Connection
   KDB.Users("Schmidt").SetPermissions "Artikel", _
      adPermObjTable, adAccessSet, adRightInsert
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 485: Berechtigungen hinzufügen (ADO)

Über die Methode `SetPermissions` können Sie die einzelnen Berechtigungen hinzufügen. Dabei hat diese Methode folgende Syntax:

```
SetPermissions Name, ObjectType, Action, Rights
```

Im Argument `Name` geben Sie den Namen des Access-Objekts an, welches Sie schützen möchten. Beim Argument `ObjectType` setzen Sie eine der in der Tabelle 140 aufgelisteten Konstanten ein.

Konstante	Beschreibung
`adPermObjTable`	Objekt ist eine Tabelle.
`adPermObjColumn`	Objekt ist eine Spalte.
`adPermObjDatabase`	Objekt ist eine Datenbank.
`adPermObjProcedure`	Objekt ist eine Prozedur.
`adPermObjView`	Objekt ist eine Ansicht.
`adPermObjSchema`	Objekt ist ein Schema.
`adPermObjDomain`	Objekt ist eine Domäne.
`adPermObjCollation`	Objekt ist eine Zusammenstellung.
`adPermObjSchemaRowset`	Objekt ist ein Rowset eines Schemas.
`adPermObjCharacterSet`	Objekt ist ein Zeichensatz.
`adPermObjTranslation`	Objekt ist eine Übersetzung.

Tabelle 140: Die Objekttypen der Methode SetPermissions

Im Argument `Action` geben Sie an, welcher Aktivitätstyp beim Festlegen von Berechtigungen durchgeführt werden soll. Die folgenden Konstanten stellen gültige Werte für `Action` dar:

Konstante	Beschreibung
`adAccessGrant`	Die Gruppe oder der Benutzer erhält mindestens die angeforderten Berechtigungen.
`adAccessSet`	Die Gruppe oder der Benutzer erhält genau die angeforderten Berechtigungen.
`adAccessDeny`	Der Gruppe oder dem Benutzer werden die angegebenen Berechtigungen verweigert.
`adAccessRevoke`	Alle der Gruppe oder dem Benutzer ausdrücklich gewährten Zugriffsrechte werden widerrufen.
`adAccessAuditSuccess`	Die Gruppe wird überprüft, wenn das Objekt, das die angeforderten Berechtigungen verwendet, erfolgreich geöffnet wird.
`adAccessAuditFailure`	Die Gruppe wird überprüft, wenn das Öffnen des Objekts, das die angeforderten Berechtigungen verwendet, fehlschlägt.

Tabelle 141: Die Action-Konstanten der Methode SetPermissions

Im letzten Argument `Rights` geben Sie die einzelnen Rechte an, die der Benutzer/die Gruppe bekommen soll.

Konstante	Beschreibung
`adRightExecute`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt auszuführen.
`adRightRead`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt zu lesen.
`adRightUpdate`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt zu aktualisieren.
`adRightInsert`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt einzufügen.
`adRightDelete`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt zu löschen.
`adRightReference`	Der Benutzer bzw. die Gruppe hat die Berechtigung, auf das Objekt zu verweisen.
`adRightCreate`	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt zu erstellen.

Tabelle 142: Die Rights-Konstanten der Methode SetPermissions

Konstante	Beschreibung
adRightWithGrant	Der Benutzer bzw. die Gruppe hat die Berechtigung, Berechtigungen für das Objekt zu erteilen.
adRightDesign	Der Benutzer bzw. die Gruppe hat die Berechtigung, das Objekt zu entwerfen.
adRightAll	Der Benutzer bzw. die Gruppe hat alle Berechtigungen für das Objekt.

Tabelle 142: Die Rights-Konstanten der Methode SetPermissions (Forts.)

Überprüfen Sie die zugewiesenen Berechtigungen, indem Sie aus dem Menü EXTRAS den Befehl SICHERHEIT/BENUTZER- UND GRUPPENBERECHTIGUNGEN auswählen.

Hinweis

Möchten Sie mehrere Rechte vergeben, trennen Sie die einzelne Rechte durch ein Pluszeichen voneinander ab.

```
KDB.Users("Schmidt").SetPermissions "Artikel", adPermObjTable, adAccessSet,
adRightInsert + adRightRead
```

408 Der Zugriff auf die VBE

Ganz oben in der Hierarchie der Objekte der Entwicklungsumgebung steht das Objekt `VBE`. Das VBE-Objekt hat mehrere Auflistungsobjekte bzw. Objekte für den Zugriff auf die einzelnen Elemente:

Objekt	Beschreibung
`AddIns`-Auflistung	Sie regelt den Zugriff auf die Auflistung der Add-Ins.
`Windows`-Auflistung	Stellt Methoden und Eigenschaften für den Zugriff auf die Fenster, wie z. B. Projekt- und Eigenschaftenfenster, bereit.
`CodePanes`-Auflistung	Ist für den Zugriff auf die geöffneten Code-Bereiche eines Projekts verantwortlich.
`CodeModul`	Enthält den Code für Komponenten, wie z. B. Formulare, Klassen oder Dokumente.
`CommandBars`- Auflistung	Kümmert sich um den Zugriff auf die Auflistung der Befehlsleisten.
`VBComponent`	Stellt eine Komponente dar, wie z. B. ein Klassenmodul oder ein Standardmodul eines Projekts.

Tabelle 143: Die wichtigsten Objekte in der VBE

Objekt	Beschreibung
`VBProjects`-Auflistung	In diesem Auflistungsobjekt sind alle geöffneten Projekte in der Entwicklungsumgebung verzeichnet.

Tabelle 143: Die wichtigsten Objekte in der VBE (Forts.)

Auf den folgenden Seiten können Sie typische Beispiele rund um den Einsatz der VBE-Programmierung nachschlagen.

Hinweis

Wenn Sie herausfinden möchten, welche Objekte, Methoden und Eigenschaften in einer bestimmten Bibliothek enthalten sind, müssen Sie diese vorher einbinden. Anschließend können Sie in der Entwicklungsumgebung die Taste F2 drücken, um den Objektkatalog anzuzeigen.

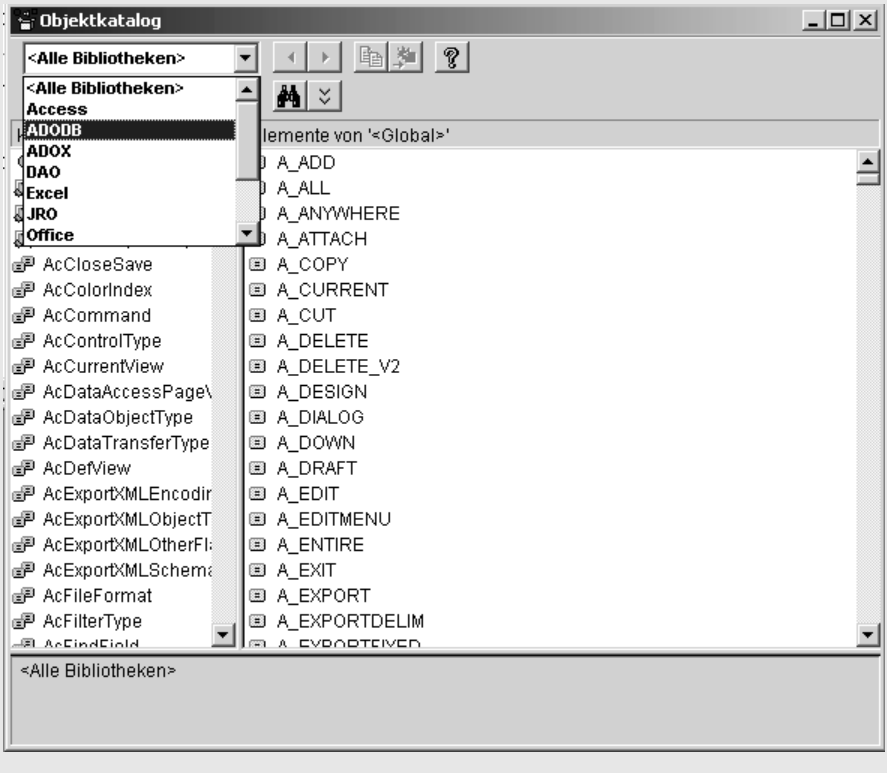


Abbildung 305: Der Objektkatalog gibt Auskunft über die einsetzbaren Befehle einer Bibliothek.

Wählen Sie aus dem Drop-down-Menü links oben die gewünschte Bibliothek aus, zu der Sie die zur Verfügung stehenden Objekte, Methoden und Eigenschaften einsehen möchten. Standardmäßig ist der Eintrag ALLE BIBLIOTHEKEN ausgewählt, was bedeutet, dass alle Objekte, Methoden und Eigenschaften der momentan aktivierten Bibliotheken angezeigt werden.

409 Die VBE-Bibliothek einbinden

Als wichtigste Bibliothek, bevor Sie überhaupt mit der VBE-Programmierung beginnen sollten, muss zunächst die VBE-Bibliothek eingebunden werden. Mithilfe der Methoden und Eigenschaften dieser Bibliothek können Sie »frei« in der Entwicklungsumgebung herumprogrammieren.

Manuelles Einbinden der VBE-Bibliothek

Um die Objektbibliothek MICROSOFT VISUAL BASIC FOR APPLICATIONS EXTENSIBILITY 5.3 einzubinden, verfahren Sie wie folgt:.

1. Wechseln Sie in die Entwicklungsumgebung.
2. Wählen Sie aus dem Menü EXTRAS den Befehl VERWEISE.

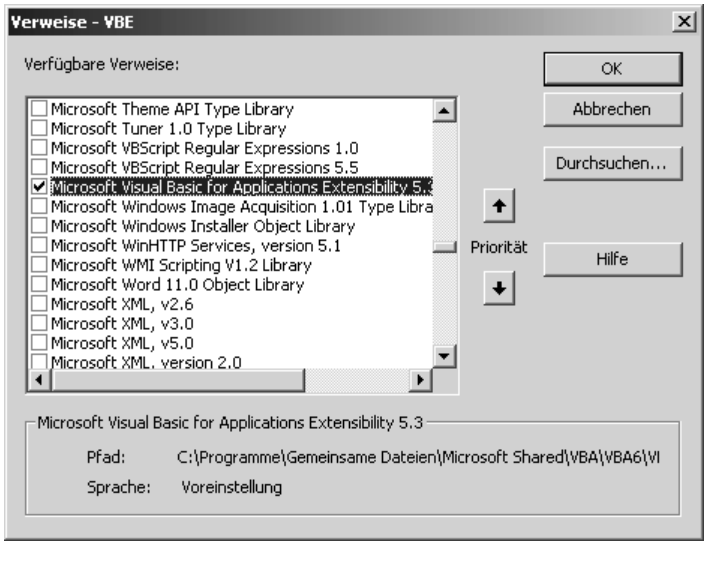


Abbildung 306: Die VBE-Bibliothek einbinden

3. Klicken Sie im Listenfeld VERFÜGBARE VERWEISE auf das Kontrollkästchen der Bibliothek MICROSOFT VISUAL BASIC FOR APPLICATIONS EXTENSIBILITY 5.3.
4. Bestätigen Sie diese Einstellung mit OK.

Automatisches Einbinden der VBE-Bibliothek

Selbstverständlich können Sie diese manuelle Einstellung auch elegant durch das Makro aus Listing 486 vornehmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub VBEAktivieren()
   Dim VBEObj As Object

   On Error Resume Next
   VBEObj = Application.VBE.ActiveVBProject.References. _
      AddFromGuid("{0002E157-0000-0000-C000-000000000046}", 5, 3)
   Set VBObj = Nothing
End Sub
```

Listing 486: Die VBE-Bibliothek über ein Makro aktivieren

Die Methode `AddFromGuid` fügt der `References`-Auflistung einen Verweis hinzu, wobei der global eindeutige Bezeichner (GUID) des Verweises verwendet wird. Die Syntax der Methode lautet:

```
AddFromGuid(GUID, HauptNr, NebenNr) As Reference
```

Das Argument `Guid` gibt einen Wert vom Typ `String` zurück, der die Klassen-ID eines Objekts enthält. Bei der `Guid` handelt es sich um eine eindeutige Nummer, welche die Bibliothek identifiziert.

Das Argument `HauptNr` gibt einen Wert vom Typ `Long` zurück, der die Hauptversionsnummer der Klassenbibliothek, auf die verwiesen wird, enthält.

Das Argument `NebenNr` gibt einen Wert vom Typ `Long` zurück, der die Nebenversionsnummer der Klassenbibliothek, auf die verwiesen wird, anzeigt. Beide Nummern sind notwendig, um die Bibliothek richtig zu adressieren. Anhand dieser beiden Nummern durchsucht die Methode `AddFromGuid` die Registrierung, um den hinzuzufügenden Verweis zu ermitteln und einzubinden.

Hinweis

Sobald Sie die Bibliothek für die VBE-Programmierung eingebunden haben, können Sie auf alle Objekte, Methoden und Eigenschaften zugreifen, die in dieser Bibliothek gespeichert sind. Um mehr Informationen zu einem bestimmten Befehl zu erhalten, können Sie sich entweder das Objektmodell ansehen und sich dann hierarchisch von oben nach unten durcharbeiten oder den Befehl mit dem Mauszeiger markieren und die Taste F1 drücken, um die Online-Hilfe aufzurufen.

Die kleine bzw. große Hilfe

Da es nahezu unmöglich ist, sämtliche GUIDs, Haupt- und Nebennummern auswendig zu wissen, wenden Sie das Makro aus Listing 487 an, um alle Informationen diesbezüglich im Direktfenster der Entwicklungsumgebung auszulesen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub VerweiseGUID()
   Dim Verweis As Reference

   For Each Verweis In Application.References
      Debug.Print Verweis.Name
      Debug.Print "GUID" & Verweis.Guid
      Debug.Print "HauptNr: " & Verweis.Major
      Debug.Print "NebenNr: " & Verweis.Minor & vbLf
   Next Verweis
End Sub
```

Listing 487: Die GUIDs der eingebundenen Bibliotheken auslesen

In einer `For Each Next`-Schleife werden alle eingebundenen Bibliotheken ausgelesen. Innerhalb der Schleife geben Sie die GUIDS, Haupt- und Nebennummern über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

Weitere Bibliotheken einbinden

Mithilfe des Makros aus Listing 487 ist es jetzt nicht mehr allzu schwierig, weitere Bibliotheken bei Bedarf auszulesen und ein Makro zu schreiben, welche die Verweise automatisch setzt.

Im folgenden Makro aus Listing 488 werden die Objektbibliotheken für MICROSOFT EXCEL sowie MICROSOFT WORD automatisch gesetzt.

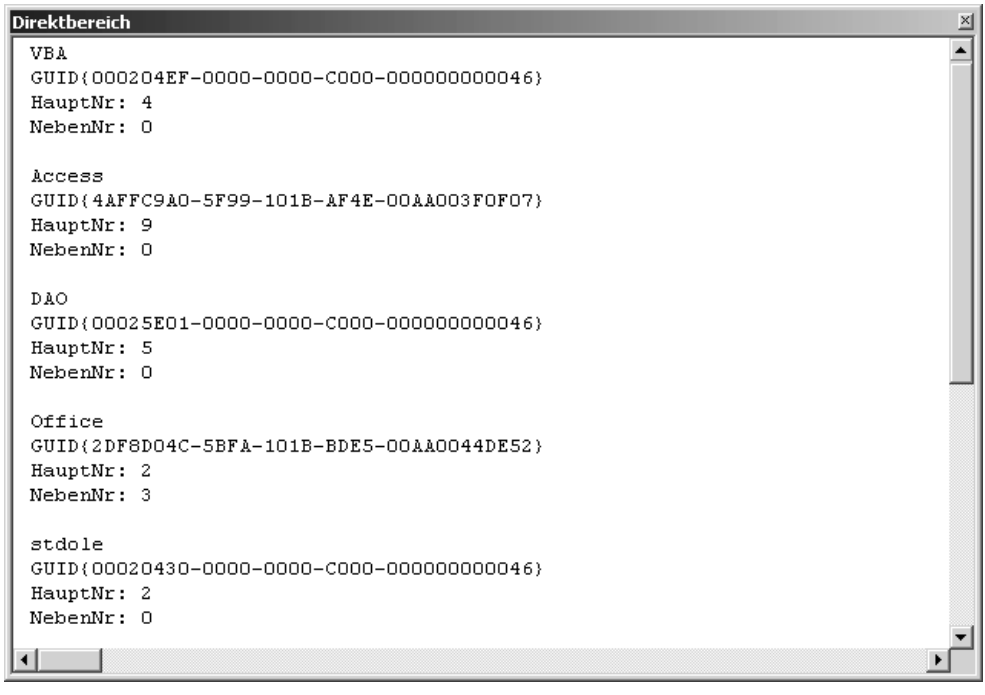


Abbildung 307: Alle GUIDs, Haupt- und Nebennummern wurden ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub WeitereBibliothekenEinbindenVar1()
   Dim VBEObj As Object

   On Error Resume Next
   'Microsoft Excel 2003 einbinden
   VBEObj = Application.VBE.ActiveVBProject.References. _
      AddFromGuid("{00020813-0000-0000-C000-000000000046}", 1, 5)

   'Microsoft Word 2003 einbinden
   VBEObj = Application.VBE.ActiveVBProject.References. _
      AddFromGuid("{00020905-0000-0000-C000-000000000046}", 8, 3)

   Set VBEObj = Nothing
End Sub
```

Listing 488: Bibliotheken einbinden – Variante 1

Übergeben Sie der Methode `AddFRomGuid` die gewünschten Informationen. Kontrollieren Sie danach über den Menübefehl EXTRAS/VERWEISE, ob die Bibliotheken MICROSOFT EXCEL und MICROSOFT WORD nun gesetzt sind.

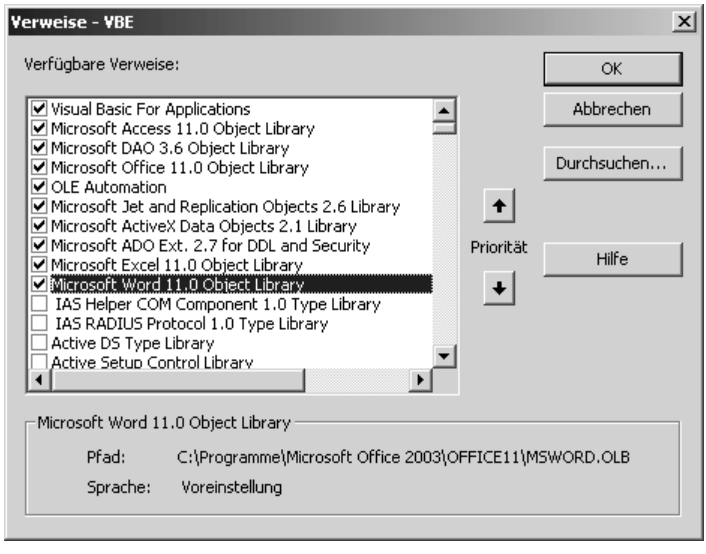


Abbildung 308: Die Bibliotheken für Excel und Word wurden eingebunden.

Die Alternative zum Einbinden

Eine alternative Möglichkeit, Bibliotheken einzubinden, stellt die Methode `AddFromFile` zur Verfügung.

Mit der Methode `AddFromFile` fügen Sie dem Projekt einen Verweis aus einer Datei hinzu. Dazu muss allerdings der Name der Bibliothek bekannt sein. Diesen können Sie über den Menübefehl EXTRAS/VERWEISE im zugehörigen Dialogfeld nachsehen.

Im folgenden Beispiel aus Listing 489 werden die beiden Bibliotheken für Excel und Word mithilfe der Methode `AddFromFile` eingebunden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub WeitereBibliothekenEinbindenVar2()
   Dim VBEObj As Object

   Set VBEObj = Application.VBE.ActiveVBProject.References
   On Error Resume Next
```

Listing 489: Bibliotheken einbinden – Variante 2

```
    VBEObj.AddFromFile "Excel.exe"
    VBEObj.AddFromFile "msword.olb"
    Set VBObj = Nothing
End Sub
```

Listing 489: Bibliotheken einbinden – Variante 2 (Forts.)

Übergeben Sie den Namen der Bibliothek der Methode `AddFromFile`.

Bibliotheks-Objektnamen auslesen

Sollen die Verweise der beiden Bibliotheken wieder entfernt werden, dann kommt die Methode `Remove` zum Einsatz. Wenn Sie die Methode `Remove` zum Entfernen der Verweise auf eine Bibliothek einsetzen, müssen Sie wissen, wie der entsprechende Objektname der Bibliothek lautet. Um also zu erfragen, wie beispielsweise die Objektnamen der Bibliotheken Excel und Word heißen, starten Sie das Makro aus Listing 490, das die Namen aller Bibliotheken ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub Verweisnamen()
   Dim ref As Object

   For Each ref In Application.VBE.ActiveVBProject.References
      Debug.Print ref.Name
   Next ref
End Sub
```

Listing 490: Die Namen der aktiven Verweise werden ausgelesen.

In einer `For Each Next`-Schleife werden alle aktiven Verweise durchlaufen. Dabei werden innerhalb der Schleife die Namen der eingebundenen Bibliotheken über die Eigenschaft `Name` ermittelt und mithilfe der Anweisung `Debug.Print` im Direktfenster ausgegeben.

Bibliotheken wieder deaktivieren

Für Excel wird der Objektname EXCEL, für Word der Objektname WORD verwendet. Übergeben Sie diese Namen wie in Listing 491 gezeigt jetzt an die Methode `Remove`, um die beiden Objektbibliotheken wieder zu deaktivieren.

Abbildung 309: Die Namen der Verweise wurden ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub BibliothekenDeaktivieren()
   Dim VBEObj As Object

   On Error Resume Next
   Set VBEObj = Application.VBE.activevbproject.References
   VBEObj.Remove VBEObj("Word")
   VBEObj.Remove VBEObj("Excel")
   Set vbobj = Nothing
End Sub
```

Listing 491: Bibliotheken deaktivieren

Übergeben Sie der Methode `Remove` die jeweiligen Objektnamen, um die Bibliotheken in der Entwicklungsumgebung zu deaktivieren. Heben Sie danach den Objektverweis wieder auf, indem die Verbindung der Objektvariablen zum zugehörigen Objekt aufzuheben.

410 Verweise auslesen

Je nachdem, welche VBA-Befehle Sie einsetzen möchten, muss die eine oder andere Bibliothek in der Entwicklungsumgebung eingebunden sein, da es sonst zu Fehlern und Makroabstürzen kommt.

Das folgende Makro aus Listing 492 listet alle eingebundenen Bibliotheken im Direktfenster der Entwicklungsumgebung auf.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub VerweiseAusgeben()
   Dim Verweis As Reference

   For Each Verweis In Application.References
      Debug.Print Verweis.Name & vbLf & Verweis.FullPath & vbLf
   Next Verweis
End Sub
```

Listing 492: Alle eingebundenen Bibliotheken auslesen

Definieren Sie im ersten Schritt eine Objektvariable vom Typ `Reference`. Danach durchlaufen Sie in einer Schleife alle Verweise, indem Sie das Auflistungsobjekt `References` abfragen. Geben Sie danach den Namen des Verweises mithilfe der Eigenschaft `Name` sowie den kompletten Pfad des Verweises über die Eigenschaft `FullPath` im Direktfenster der Entwicklungsumgebung aus (siehe Abbildung 310).

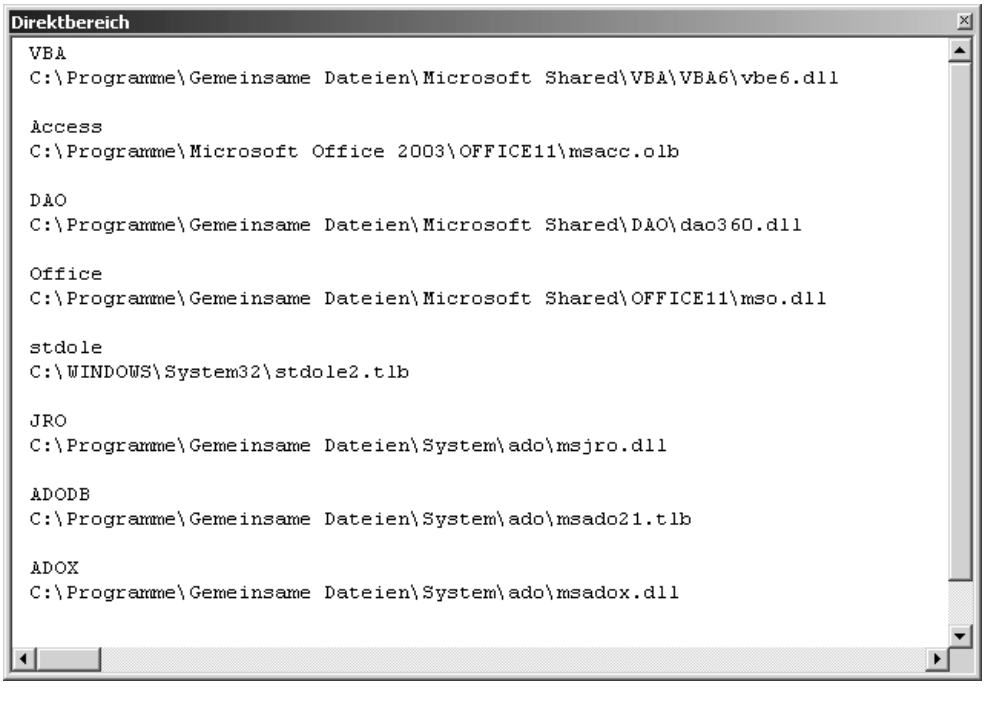


Abbildung 310: Die Namen und Speicherpfade der eingebundenen Bibliotheken ausgeben

411 Auf der Suche nach fehlerhaften Verweisen

Ist eine Bibliothek in der Entwicklungsumgebung nicht verfügbar, dann kann es beim Ausführen von Makros zu Fehlern kommen. Mit dem Makro aus Listing 493 können Sie im Vorfeld schon fehlerhafte Verweise aufspüren und im Direktfenster der Entwicklungsumgebung ausgeben.

```
'================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'================================================

Sub VerweisCheck()
   Dim Verweis As Reference
   For Each Verweis In Application.References
      If Verweis.IsBroken = True Then
         Debug.Print Verweis.Name & vbLf & Verweis.FullPath & vbLf
      End If
   Next Verweis
End Sub
```

Listing 493: Fehlerhafte Verweise aufspüren

Definieren Sie im ersten Schritt eine Objektvariable vom Typ `Reference`. Danach durchlaufen Sie in einer `For Each Next`-Schleife alle momentan gesetzten Verweise, indem Sie das Auflistungsobjekt `References` abfragen.

Über die Eigenschaft `IsBroken` können Sie prüfen, ob der jeweilige Verweis gültig ist. Kann ein Verweis nicht ermittelt werden, dann liefert diese Eigenschaft den Wert `True`. In diesem Fall geben Sie den Namen des Verweises mithilfe der Eigenschaft `Name` sowie den kompletten Pfad des Verweises über die Eigenschaft `FullPath` im Direktfenster der Entwicklungsumgebung aus.

412 VB-Komponenten identifizieren

Neben einzelnen Modulen kann es weitere Komponenten in der Entwicklungsumgebung geben, wie zum Beispiel Formulare, Berichte und Klassenmodule.

Im Beispiel aus Listing 494 sehen Sie ein Makro, das die verschiedenen Komponenten der Datenbank im Direktfenster ausgibt.

```
'================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'================================================
```

Listing 494: Die einzelnen VB-Komponenten ermitteln

```
Sub KomponentenAusgeben()
   Dim VBKomp As VBComponent

   For Each VBKomp In Application.VBE.ActiveVBProject.VBComponents
      Debug.Print "Name der Komponente: " & VBKomp.Name & _
         vbLf & "Typ der Komponente: " & _
         VBKomp.Type & vbLf
   Next VBKomp
   Set VBKomp = Nothing
End Sub
```

Listing 494: Die einzelnen VB-Komponenten ermitteln (Forts.)

In Abbildung 311 sehen Sie, dass jede Komponente einen eindeutigen Typ aufweist, der über die Eigenschaft `Type` ermittelt werden kann. Dies ermöglicht später das gezielte Ansprechen einzelner Komponenten.

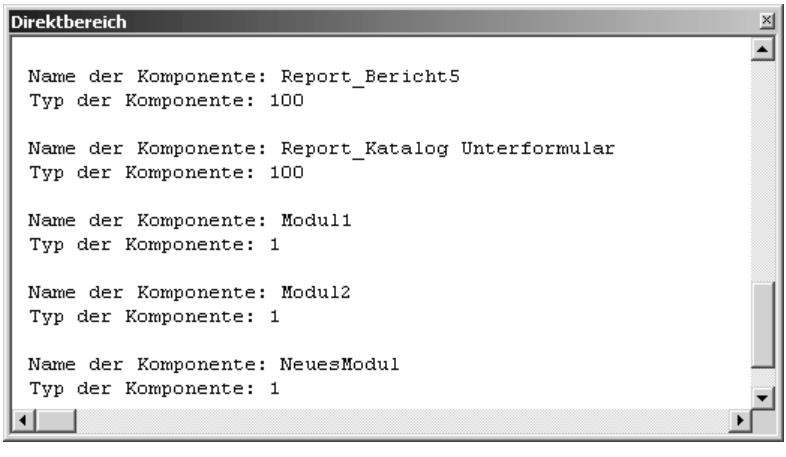


Abbildung 311: Die einzelnen Typen ermitteln

413 Die VBE aufrufen

Standardmäßig drücken Sie die Tastenkombination Alt + F11, um in die Entwicklungsumgebung zu gelangen.

Diesen Vorgang können Sie aber auch durch ein Makro ausführen, das Sie im Makro aus Listing 495 sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================
```

Listing 495: Die Entwicklungsumgebung aufrufen

```
Sub VBEAufrufen()
   With Application.VBE.MainWindow
      .SetFocus
      .Visible = True
   End With
End Sub
```

Listing 495: Die Entwicklungsumgebung aufrufen (Forts.)

Die Eigenschaft `MainWindow` gibt ein `Window`-Objekt zurück, das das Hauptfenster der Entwicklungsumgebung von Visual Basic darstellt. Die Methode `SetFocus` verschiebt den Fokus auf das angegebene Fenster. Über die Eigenschaft `Visible` wird für das `Window`-Objekt ein Wert vom Typ `Boolean` zurückgegeben oder festgelegt, der die Sichtbarkeit von Fenstern angibt. Setzen Sie diese Eigenschaft auf den Wert `True`, um ein Fenster anzuzeigen.

414 Neue Module einfügen

Um einer Datenbank ein neues, noch leeres Modul hinzuzufügen, wählen Sie in der Entwicklungsumgebung aus dem Menü EINFÜGEN den Befehl MODUL aus.

Wenn diese Aufgabe über ein Makro erledigt werden soll, dann setzen Sie dazu die Methode `Add` ein. Die Syntax dieser Methode lautet:

```
Objekt.Add(Komponente)
```

Argument	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `VBComponents`-Objekt zurückgibt.
`Komponente`	Erforderlich. Für die `LinkedWindows`-Auflistung ein Objekt. Für die `VBComponents`-Auflistung eine aufgelistete Konstante, die ein Klassenmodul, ein Formular oder ein Standardmodul darstellt.

Tabelle 144: Die Argumente der Methode Add

Sie können eine der folgenden Konstanten für das Argument `Komponente` verwenden:

Konstante	Beschreibung
`vbext_ct_ClassModule`	Fügt der Auflistung ein Klassenmodul hinzu.
`Vbext_ct_MSForm`	Fügt der Auflistung ein Formular hinzu.
`Vbext_ct_StdModule`	Fügt der Auflistung ein Standardmodul hinzu.
`vbext_pt_StandAlone`	Fügt der Auflistung ein StandAlone-Projekt hinzu.

Tabelle 145: Die verfügbaren Konstanten der Methode Add

Im folgenden Beispiel aus Listing 496 wird dem aktuellen VBA-Projekt ein neues Modul mit dem Namen NEUESMODUL hinzugefügt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub ModulEinfügen()
   Dim VBKomp As VBComponent

   On Error GoTo fehler
   Set VBKomp = _
   Application.VBE.ActiveVBProject.VBComponents.Add(vbext_ct_StdModule)
   VBKomp.Name = "NeuesModul"
   Set VBKomp = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 496: Ein neues Modul einfügen

Da es sich bei einem neuen Modul um eine VB-Komponente handelt, deklarieren Sie zu Beginn des Makros eine Objektvariable vom Typ `VBComponent`. Das Füllen der Objektvariablen wird im Zusammenspiel mit dem Anlegen des neuen Moduls durchgeführt. Wenden Sie die Methode `Add` auf das aktive Projekt an, indem Sie der `VBComponents`-Auflistung ein weiteres Modul hinzufügen. Über die Eigenschaft `Name` geben Sie diesem neuen Modul einen Namen und heben danach den Objektverweis wieder auf.

415 Modul löschen

Wenn Sie bereits vorhandene Module in einer Datenbank löschen möchten, um z.B. im Anschluss daran ein neues überarbeitetes Modul einzufügen, dann starten Sie das folgende Makro aus Listing 497.

Abbildung 312: Das neue Modul wurde angelegt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub ModulLöschen()
   Dim VBmodul As Object

   On Error GoTo fehler
   Set VBmodul = Application.VBE.ActiveVBProject

   With Vbmodul
     .VBComponents.Remove .VBComponents("NeuesModul")
   End With
   Set VBmodul = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 497: Modul löschen

Mit der Methode `Remove` können Sie ein bestimmtes Modul löschen.

416 Alle Module löschen

Das folgende Makro aus Listing 498 geht noch einen Schritt weiter. Es löscht alle Module der aktiven Datenbank bis auf das MODUL1 aus der Datenbank.

Hinweis

Vorsicht bei der Anwendung dieses Makros! Danach sind tatsächlich alle Module weg!

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub AlleModuleBisAufEinesLöschen()
   Dim VBKomp As Object

   On Error GoTo fehler
   With Application.VBE.ActiveVBProject

      For Each VBKomp In .VBComponents
         If VBKomp.Type = 1 Or VBKomp.Type = 2 Then
            If VBKomp.Name <> "Modul1" Then .VBComponents.Remove VBKomp
         End If
      Next VBKomp
   End With
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 498: Alle Module bis auf das Modul1 werden ohne Rückfrage gelöscht.

Bei allen VB-Komponenten, die den `Type` 1 oder 2 zurückmelden, handelt es sich entweder um Klassen- oder Standardmodule. Diese können Sie dann direkt über die Methode `Remove` aus der Datenbank entfernen.

417 Modulcheck durchführen

Im letzten Abschnitt haben Sie erfahren, wie Sie ein Modul löschen. Was noch fehlt, ist eine Prüfung, ob ein bestimmtes Modul sich überhaupt in der Datenbank befindet. Diese Prüfung können Sie beispielsweise über die Funktion aus Listing 499 durchführen. Dabei wird überprüft, ob sich das Modul mit dem Namen MODUL2 in der aktuell geladenen Datenbank befindet.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'====================================================

Function ModulDa(ModulName As String) As Boolean
   On Error Resume Next
   ModulDa = _
      Len(Application.VBE.ActiveVBProject.VBComponents(ModulName).Name) <> 0
End Function


Sub ModulCheck()
   If ModulDa("Modul2") = True Then
      MsgBox "Modul existiert!", vbInformation
   Else
      MsgBox "Modul existiert nicht!", vbCritical
   End If
End Sub
```

Listing 499: Überprüfen, ob ein Modul existiert

Über die Funktion `Len` können Sie prüfen, ob eine korrekte Länge, also ungleich 0, zurückgegeben wird. Dies ist der Fall, wenn der übergebene Modulname in der Datenbank gefunden werden kann.

418 Module drucken

Das Drucken von Modulen ist standardmäßig über das Menü DATEI und den Befehl DRUCKEN möglich. Dabei besteht aber keine Möglichkeit, das Aussehen des Ausdrucks sowie die Formatierung der Schriftart speziell für den Druck vorzunehmen.

Die Lösung aus Listing 500 für dieses Problem besteht darin, zuerst das komplette Modul in eine Textdatei zu schreiben, diese Textdatei dann beispielsweise in Excel zu öffnen und zu formatieren. Nach dem Ausdruck dieser formatierten Datei erfolgt die Löschung derselben.

```
'====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'====================================================

Sub ModulDrucken()
   Dim XL As Object

   Application.VBE.ActiveVBProject.VBComponents _
      ("Modul1").Export "c:\Eigene Dateien\Codes.txt"
```

Listing 500: Ein Listing drucken

```
    Set XL = CreateObject("Excel.Application")

    XL.Workbooks.OpenText "c:\Eigene Dateien\Codes.txt"

    With XL.Activeworkbook.ActiveSheet.Cells
       .Font.Name = "Courier"
       .Font.Size = 8
       .PrintOut
    End With

    XL.Activeworkbook.Close savechanges:=False
    Kill "c:\Eigene Dateien\Codes.txt"
    Set XL = Nothing
End Sub
```

Listing 500: Ein Listing drucken (Forts.)

Deklarieren Sie zunächst eine Objektvariable vom Typ `Object`. Danach wenden Sie die Methode `Export` an, um das gewünschte Modul als Textdatei zu exportieren. Geben Sie dabei neben dem Dateinamen auch den Pfadnamen an. Dies erleichtert später das Wiederfinden der Textdatei. Erstellen Sie nun ein Excel-Objekt und öffnen Sie die als Text gespeicherte Codedatei mithilfe der Methode `OpenText`. Formatieren Sie danach alle Zellen der aktiven Tabelle, indem Sie eine gewünschte Schriftart sowie Schriftgröße über die Eigenschaften `Name` und `Size` einstellen. Über die Methode `Printout` geben Sie die so formatierte Textdatei auf dem Drucker aus. Schließen Sie die noch geöffnete Excel-Mappe über die Methode `Close`. Geben Sie dabei als Argument `SaveChanges:=False` an. Diese Einstellung bewirkt, dass Sie keine Rückfrage erhalten, ob die Datei gespeichert werden soll. Wenden Sie zum Schluss des Makros noch die Anweisung `Kill` an, um die Textdatei zu löschen.

419 Makro löschen

Möchten Sie aus einem Modul ein bestimmtes Makro entfernen, dann müssen Sie dieses Makro zuerst finden. Im folgenden Makro aus Listing 501 wird in MODUL1 nach dem Makro TEST gesucht. Wenn dieses Makro gefunden wird, dann soll es aus dem Modul entfernt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub MakroLöschen()
   Dim VBKomp As Object
   Dim int_Start As Integer
   Dim int_Anzahl As Integer
```

Listing 501: Makro entfernen

```
    Set VBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("Modul1").CodeModule

    On Error GoTo fehler
    With VBKomp
       int_Start = .ProcStartLine("Test", vbext_pk_Proc)
       int_Anzahl = .ProcCountLines("Test", vbext_pk_Proc)
       .DeleteLines int_Start, int_Anzahl
    End With
    Exit Sub


    fehler:
       MsgBox "Ein Makro mit diesem Namen gibt es nicht!"
 End Sub
```

Listing 501: Makro entfernen (Forts.)

Die Eigenschaft `ProcStartLine` gibt die Zeile zurück, an der die festgelegte Prozedur beginnt. Die Eigenschaft `ProcCountLines` gibt die Anzahl der Zeilen in der festgelegten Prozedur an. Die Syntax dieser Eigenschaften lautet:

```
Objekt.ProcStartLine(ProzName, ProzArt) As Long
Objekt.ProcCountLine(ProzName, ProzArt) As Long
```

Teil	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `CodeModul`-Objekt zurückgibt.
`ProzName`	Erforderlich. Ein Wert vom Typ `String`, der den Namen der Prozedur enthält.
`ProzArt`	Erforderlich. Legt die Art der zu suchenden Prozedur fest.

Tabelle 146: Die Argumente der Eigenschaft ProcStartLine

Sie können eine der folgenden Konstanten für das Argument `ProzArt` verwenden:

Konstante	Beschreibung
`vbext_pk_Get`	Legt eine Prozedur fest, die den Wert einer Eigenschaft zurückgibt.
`vbext_pk_Let`	Legt eine Prozedur fest, die einer Eigenschaft einen Wert zuweist.
`vbext_pk_Set`	Legt eine Prozedur fest, die einen Verweis auf ein Objekt angibt.
`vbext_pk_Proc`	Legt alle Prozeduren mit Ausnahme von Eigenschaftenprozeduren fest.

Tabelle 147: Die Konstanten des Arguments ProzArt

Bei erfolgreicher Aufspürung des Makros wenden Sie die Methode `DeleteLines` an, um das Makro zu löschen. Dabei geben Sie der Methode sowohl die erste als auch die letzte zu löschende Zeile an. Die Syntax dieser Methode lautet:

```
Objekt.DeleteLines (Startzeile) [, Zähler]
```

Teil	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `CodeModul`-Objekt zurückgibt.
`Startzeile`	Erforderlich. Ein Wert vom Typ `Long`, der die erste zu löschende Zeile angibt.
`Zähler`	Optional. Ein Wert vom Typ `Long`, der die Anzahl der zu löschenden Zeilen angibt.

Tabelle 148: Die Argumente der Methode DeleteLines

Für den Fall, dass das gesuchte Makro nicht gefunden werden kann, sorgen Sie mithilfe der `On Error`-Anweisung dafür, dass auch dieses Makro nicht abstürzt.

420 Makrocheck durchführen

Bevor Sie ein Makro löschen bzw. auf ein Makro in irgendeiner Form zugreifen, können Sie vorab eine Prüfung vornehmen, ob das entsprechende Makro überhaupt in der Datenbank existiert. Dazu schreiben Sie eine Funktion, die so aussehen kann wie in Listing 502 dargestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Function ModulDa(ModulName As String) As Boolean
   On Error Resume Next
   ModulDa = _
   Len(Application.VBE.ActiveVBProject.VBComponents(ModulName).Name) <> 0
End Function


Function MakroDa(MakroName As String, _
   ModulName As String) As Boolean

   On Error Resume Next
   If ModulDa(ModulName) = True Then
      MakroDa = Application.VBE.ActiveVBProject.VBComponents(ModulName) _
      .CodeModule.ProcStartLine(MakroName, vbext_pk_Proc) <> 0
```

Listing 502: Existiert ein bestimmtes Makro in einem Modul?

```
    End If
End Function

Sub MakroCheck()
    If MakroDa("DatumUndZeit", "Modul1") = True Then
        MsgBox "Makro existiert!", vbInformation
    Else
        MsgBox "Makro existiert nicht!", vbCritical
    End If
End Sub
```

Listing 502: Existiert ein bestimmtes Makro in einem Modul? (Forts.)

Übergeben Sie der Funktion MakroDa den Namen des Makros sowie den Namen des Moduls, in dem das Makro gespeichert ist. Über die Eigenschaft ProcStartLine wird dann in der Funktion versucht, das Makro zu finden. Ist die Suche erfolgreich, dann wird eine Länge ungleich Null zurückgegeben. Den Abgleich mit dem Modul können Sie in der Beschreibung unterhalb der Funktion aus Listing 499 nachlesen.

421 Formularereignisse löschen

Neben Makros, die in Standardmodulen oder Klassenmodulen gespeichert sind, gibt es auch noch Ereignismakros, die sich direkt hinter Formularen oder Berichten befinden.

Um alle Makros, die hinter einem Formular liegen, zu entfernen, starten Sie das Makro aus Listing 503.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularCodeLöschen()
    Dim VBKomp As Object

    Set VBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("Form_Artikel").CodeModule

    On Error GoTo fehler
    With VBKomp
        .DeleteLines 1, VBKomp.CountOfLines
    End With
    Exit Sub


    fehler:
        MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 503: Alle Makros hinter dem Formular werden gelöscht.

Übergeben Sie der Methode `DeleteLines` als erstes Argument die Startzeile, ab der die Löschung des Quellcodes beginnen soll. Da Sie im Beispiel aus Listing 503 den kompletten Code entfernen möchten, übergeben Sie hierbei die Zeile 1. Im zweiten Argument der Methode `DeleteLines` ermitteln Sie mithilfe der Eigenschaft `CountOfLines` die Anzahl der Codezeilen, die sich hinter dem Formular befinden.

Steuerelemente löschen

Nun ja, wie Sie die Codezeilen hinter einem Formular löschen können, wissen Sie jetzt. Was aber, wenn Sie das komplette Formular bzw. alle Steuerelemente auf einem Formular löschen möchten?

Im Makro aus Listing 504 werden alle Steuerelemente des Formulars ARTIKEL2 gelöscht.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub EinzelneSteuerelementeEnfernen()
   Dim frm As Form
   Dim arrnamen(50) As Variant
   Dim ctrl As Control
   Dim intz As Integer

   DoCmd.OpenForm "Artikel2", acDesign
   Set frm = Forms!Artikel2

   intz = 1
   For Each ctrl In frm.Controls
      arrnamen(intz) = ctrl.Name
      intz = intz + 1
   Next ctrl

   On Error Resume Next
   For intz = LBound(arrnamen) To Ubound(arrnamen)
      DeleteControl "Artikel2", arrnamen(intz)
   Next intz
End Sub
```

Listing 504: Alle Steuerelemente eines Formulars einfernen

Öffnen Sie zuerst das Formular `Artikel2` in der Entwurfsansicht, indem Sie die Methode `OpenForm` mit der Konstanten `acDesign` aufrufen. Danach weisen Sie der Objektvariablen `frm` den Namen des Formulars zu, indem Sie die Anweisung `Set` einsetzen. In einer `For each next`-Schleife durchlaufen Sie alle Steuerelemente des Formulars und füllen dabei das Datenfeld `arrNamen`. In einer zweiten Schleife, beginnend vom ersten Eintrag im Datenfeld (`Lbound`) bis zum letzten Eintrag im Datenfeld (`Ubound`), werden die einzelnen Steuerelemente über die Methode `DeleteControl` gelöscht.

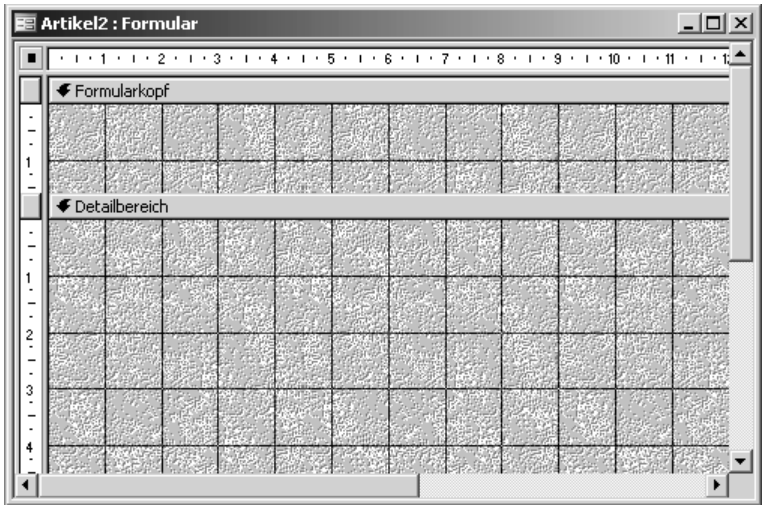


Abbildung 313: Das Formular wurde geputzt.

Formular löschen

Soll das ganze Formular gelöscht werden, dann wenden Sie das Makro aus Listing 505 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularLöschen()
   DoCmd.DeleteObject acForm, "Artikel2"
End Sub
```

Listing 505: Ein Formular löschen

Setzen Sie die Methode `DeleteObject` ein, um das Formular zu löschen.

422 Einzelne Texte/Befehle im Quellcode finden

Möchten Sie einen bestimmten Text oder Befehl im Quellcode finden, dann können Sie die Methode `Find` einsetzen.

Im folgenden Beispiel aus Listing 506 wird nach der Methode `ProcCountLines` im MODUL1 gesucht. Ist die Suche erfolgreich verlaufen, dann soll der entsprechende Befehl im Quellcode markiert werden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub BefehlFinden()
   Dim VBKomp As Object
   Dim lngZeileStart As Long
   Dim lngZeileEnde As Long
   Dim lngSpalteStart As Long
   Dim lngSpalteEnde As Long
   Dim b As Boolean

   Set VBKomp = _
      Application.VBE.ActiveVBProject.VBComponents("mdlVBE").CodeModule

   With VBKomp
      b = .Find("ProcCountLines", lngZeileStart, lngSpalteStart, _
         lngZeileEnde, lngSpalteEnde)

      If b = True Then
         .CodePane.SetSelection lngZeileStart, lngSpalteStart, _
             lngZeileEnde, lngSpalteEnde
      Else
         MsgBox "Der Befehl konnte nicht gefunden werden!"
      End If
   End With

   Set VBKomp = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 506: Befehl im Quellcode suchen und markieren

Die Methode `Find` hat fünf Argumente, die gefüllt werden müssen. Im ersten Argument geben Sie den Text an, der im Quellcode gesucht werden soll. Im zweiten Argument wird die Zeilennummer festgelegt, bei der die Suche beginnen soll. Das dritte Argument legt die Startspalte fest, ab der gesucht werden soll. Dabei beginnt die erste Spalte ganz links. Im vierten Argument wird die Endzeile angegeben, also die Zeile, bis zu der gesucht werden soll. Im letzten Argument wird die Spalte angegeben, bis zu der gesucht werden soll.

War die Suche erfolgreich, dann wird ein Wahrheitswert zurückgegeben. In diesem Fall wenden Sie die Methode `SetSelection` an und übergeben ihr die gerade durch die Methode `Find` gefüllten Variablen. Damit wird der gefundene Text im Quellcode markiert.

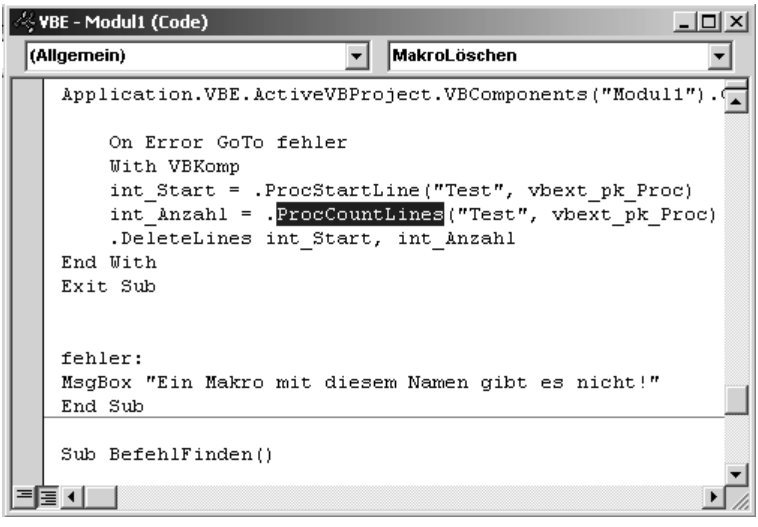


Abbildung 314: Der Befehl wurde gefunden und markiert.

423 Texte ersetzen

Im folgenden Beispiel aus Listing 507 soll eine E-Mail-Adresse in einem bestimmten Modul ausgetauscht werden. Da diese Mail-Adresse dort mehrfach vorkommen kann, muss die Lösung dynamisch programmiert werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub TextAustauschen()
   Dim lngZeileStart As Long
   Dim lngZeileEnde As Long
   Dim lngSpalteStart As Long
   Dim lngSpalteEnde As Long
   Dim strAlt As String
   Dim strNeu As String
   Dim VBCodeMod As CodeModule
   Dim b As Boolean

   Const SuchText = "Machero@aol.com"
   Const ErsetzText = "Held-office@t-online.de"

   On Error GoTo fehler
   Set VBCodeMod = Application.VBE.ActiveVBProject.VBComponents _
      ("mdlVBE").CodeModule
```

Listing 507: Textteile im Code suchen und ersetzen

```
    b = VBCodeMod.Find(SuchText, lngZeileStart + 1, _
       lngSpalteStart, lngZeileEnde, lngSpalteEnde, False, False)

    While b = True
       strAlt = VBCodeMod.Lines(lngZeileEnde, 1)
       strNeu = Replace(strAlt, SuchText, ErsetzText)
       VBCodeMod.ReplaceLine lngZeileEnde, strNeu
       lngZeileStart = lngZeileEnde + 1
       lngZeileEnde = VBCodeMod.CountOfLines
       lngSpalteStart = 1
       lngSpalteEnde = 250
       b = VBCodeMod.Find(SuchText, lngZeileStart + 1, _
          lngSpalteStart, lngZeileEnde, lngSpalteEnde)
    Wend
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 507: Textteile im Code suchen und ersetzen (Forts.)

Deklarieren Sie zu Beginn des Makros zunächst den Text, der gesucht werden soll, sowie den Text, der als Ersatz genommen werden soll. Danach legen Sie über die Anweisung `Set` fest, in welchem Modul gesucht werden soll. Wenden Sie dann die Methode `Find` an, um den gewünschten Text aufzuspüren. Speichern Sie das Ergebnis der Suche in einer booleschen Variablen. War die Suche erfolgreich, dann steht in dieser Variablen der Wert `True`. Außerdem sind danach alle Argumente der Methode `Find` gefüllt, so dass feststeht, wo der gesuchte Text sich im Modul befindet. Die Syntax der Methode `Find` lautet:

```
Find(Ziel, Startzeile, Startspalte, Endzeile, Endspalte [, GanzesWort] [, Groß/Klein]
[, Mustersuche]) As Boolean
```

Teil	Beschreibung
`Ziel`	Erforderlich. Ein Wert vom Typ `String`, der den zu suchenden Text oder das zu suchende Muster enthält.
`Startzeile`	Erforderlich. Ein Wert vom Typ `Long`, der die Zeile angibt, in der mit der Suche begonnen werden soll; falls die Suche erfolgreich ist, wird dieser Wert auf die gefundene Zeile gesetzt. Die erste Zeile hat die Nummer 1.
`Startspalte`	Erforderlich. Ein Wert vom Typ `Long`, der die Spalte angibt, in der mit der Suche begonnen werden soll; falls die Suche erfolgreich ist, wird dieser Wert auf die gefundene Spalte gesetzt. Die erste Spalte hat die Nummer 1.

Tabelle 149: Die Argumente der Methode Find

Teil	Beschreibung
Endzeile	Erforderlich. Ein Wert vom Typ Long, der die letzte Zeile der Übereinstimmung angibt, falls die Suche erfolgreich war. Die letzte Zeile kann als –1 angegeben werden.
Endspalte	Erforderlich. Ein Wert vom Typ Long, der die letzte Spalte der Übereinstimmung angibt, falls die Suche erfolgreich war. Die letzte Spalte kann als –1 angegeben werden.
GanzesWort	Optional. Ein Wert vom Typ Boolean, der angibt, ob nur nach übereinstimmenden ganzen Wörtern gesucht werden soll. Entspricht dieser Wert True, so werden nur ganze Wörter gesucht. False entspricht der Voreinstellung.
Groß/Klein	Optional. Ein Wert vom Typ Boolean, der angibt, ob die Groß-/Kleinschreibung berücksichtigt wird. Entspricht dieser Wert True, so wird bei der Suche die Groß-/Kleinschreibung beachtet. False entspricht der Voreinstellung.
Mustersuche	Optional. Ein Wert vom Typ Boolean, der angibt, ob die Zielzeichenfolge ein reguläres Ausdrucksmuster angibt. Entspricht dieser Wert True, so wird die Zielzeichenfolge als reguläres Ausdrucksmuster behandelt. False entspricht der Voreinstellung.

Tabelle 149: Die Argumente der Methode Find (Forts.)

Mithilfe der Funktion Replace ersetzen Sie jetzt die alte E-Mail-Adresse durch die neue. Die Syntax dieser Funktion sieht wie folgt aus:

```
Replace(expression, find, replace[, start[, count[, compare]]])
```

Teil	Beschreibung
Expression	Erforderlich. Zeichenfolgenausdruck, der die zu ersetzende, untergeordnete Zeichenfolge enthält.
Find	Erforderlich. Die untergeordnete Zeichenfolge, nach der gesucht wird.
Replace	Erforderlich. Die untergeordnete Ersatzzeichenfolge.
Start	Optional. Position in expression, an der die Suche nach der untergeordneten Zeichenfolge beginnt. Wird diese Angabe ausgelassen, wird bei 1 begonnen.
Count	Optional. Anzahl der durchzuführenden Ersetzungen der untergeordneten Zeichenfolge. Wird diese Angabe ausgelassen, ist die Standardeinstellung –1, d.h., alle möglichen Zeichenfolgen werden ersetzt.
Compare	Optional. Numerischer Wert, der die Art des Vergleichs angibt, der beim Beurteilen von untergeordneten Zeichenketten verwendet werden soll.

Tabelle 150: Die Argumente der Methode Replace

Die eigentliche Codezeile wird über die Methode `ReplaceLine` ausgetauscht. Die Syntax für diese Methode lautet:

```
ReplaceLine(Zeile, Code)
```

Teil	Beschreibung
`Zeile`	Erforderlich. Ein Wert vom Typ `Long`, der die Position der zu ersetzenden Zeile angibt.
`Code`	Erforderlich. Ein Wert vom Typ `String`, der den einzufügenden Code enthält.

Tabelle 151: Die Argumente der Methode ReplaceLine

Nach dem ersten »Suchtreffer« erhöhen Sie die Variable `lngZeileStart` um den Wert 1, um die Suche nicht wieder von vorne beginnen zu lassen. Wiederholen Sie in einer Schleife so lange die Suche, bis kein Suchergebnis mehr gefunden werden kann. In diesem Fall wird die boolesche Variable `b` dann auf den Wert `False` gesetzt, was den Abbruch der Schleife zur Folge hat.

424 Makros eines Moduls auflisten

Um einen Überblick über alle in einem Modul enthaltenen Makros zu gewinnen, können Sie das Makro aus Listing 508 einsetzen:

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub MakrosAuflisten()
   Dim VbKomp As Object
   Dim lngStart As Long

   Set VbKomp = Application.VBE.ActiveVBProject.VBComponents("mdlVBE").CodeModule

   With VbKomp
      lngStart = .CountOfDeclarationLines + 1
      Do Until lngStart >= .CountOfLines
         Debug.Print .ProcOfLine(lngStart, vbext_pk_Proc)
         lngStart = lngStart + .ProcCountLines(.ProcOfLine(lngStart, _
            vbext_pk_Proc), vbext_pk_Proc)
      Loop
   End With
   Set VbKomp = Nothing
End Sub
```

Listing 508: Eine Makroliste erstellen

Zu Beginn des Makros erfahren Sie mithilfe der Eigenschaft `CountOfDeclarationLines` die Anzahl der Codezeilen im Deklarationsabschnitt des Codemoduls. Mithilfe der Eigenschaft `ProcOfLine` können Sie den Namen des jeweiligen Makros im MDLVBE ermitteln. Diese Eigenschaft benötigt zwei Argumente: Im ersten Argument geben Sie die Startzeile an, ab der nach dem Makronamen gesucht werden soll. Das zweite Argument benennt die Art der zu suchenden Prozedur. Über die Konstante `vbext_pk_Proc` wird dabei festgelegt, dass Makros und Funktionen ermittelt werden sollen. Über die Eigenschaft `CountOfLines` ermitteln Sie die Gesamtzahl der Codezeilen in einem Modul.

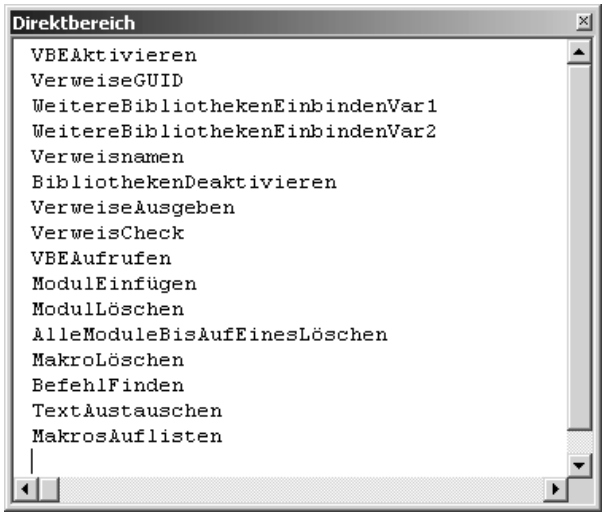


Abbildung 315: Alle Makros aus dem Modul1

Und jetzt alle ...

Um das Makro aus allen Module der Datenbank auszubauen, starten Sie das Makro aus Listing 509:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub MakrosAuflistenKomplett()
   Dim VbKomp As VBComponent
   Dim obj As Object
   Dim lngStart As Long

   For Each VbKomp In Application.VBE.ActiveVBProject.VBComponents
      If VbKomp.Type = 1 Then
         Set obj = VbKomp.CodeModule
```

Listing 509: Eine komplette Makroliste erstellen

```
            With obj
               lngStart = .CountOfDeclarationLines + 1
               Do Until lngStart >= .CountOfLines
                  Debug.Print obj.Name & " --- > " & _
                     .ProcOfLine(lngStart, vbext_pk_Proc)
                  lngStart = lngStart + .ProcCountLines(.ProcOfLine(lngStart, _
                     vbext_pk_Proc), vbext_pk_Proc)
               Loop
            End With
         End If
      Next VbKomp

      Set obj = Nothing
      Set VbKomp = Nothing
   End Sub
```

Listing 509: Eine komplette Makroliste erstellen (Forts.)

Über die Eigenschaft `Type` können Sie feststellen, ob es sich bei dem jeweiligen Objekt auch um ein Standardmodul handelt. In diesem Fall meldet die Eigenschaft den Wert 1 zurück.

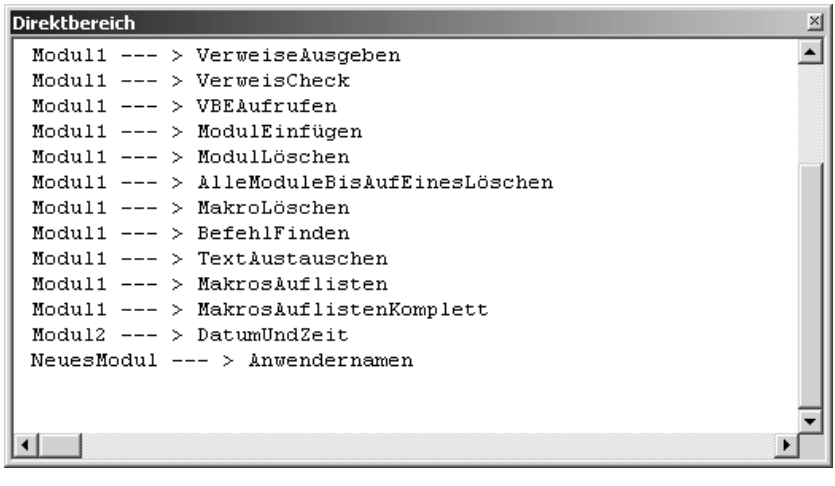


Abbildung 316: Die Makroliste mit Modulbezug wurde erstellt.

425 Makros einfügen

Wenn Sie Makros per Code in eine Datenbank einfügen möchten, dann stehen Ihnen zwei Möglichkeiten zur Verfügung:

- zeilenweises Einfügen in das Modul,
- Import einer Textdatei, die das komplette Makro in das Modul einfügt.

Variante 1 – die Lösung über InsertLines

Im ersten Beispiel fügen Sie Quellcode zeilenweise in Ihr neues Modul ein. Sehen Sie sich dazu Listing 510 an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub MakroZeilenHinzufügen()
   Dim CodeModul As CodeModule
   Dim intZ As Integer

   On Error GoTo fehler
   Set CodeModul = Application.VBE.ActiveVBProject.VBComponents _
      ("NeuesModul").CodeModule

   With CodeModul
      intZ = .CountOfLines + 1
      .InsertLines intZ, "Sub DatumUndZeit()" & vbLf & _
         " Msgbox ""Jetzt ist " & Now & "!"" " & vbLf & "End Sub"
   End With
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 510: Ein Makro einfügen – Variante 1

Im Makro wird ein Code zeilenweise übertragen. Die Eigenschaft `CountOfLines` ermittelt, wie viele Codezeilen im Modul bereits enthalten sind, und addiert den Wert 1 darauf. Diese Maßnahme ist notwendig, um eventuell bereits bestehende Makros nicht zu überschreiben. Die Konstante `vbLF` im Makro sorgt jeweils für den Zeilenvorschub. Die Eigenschaft `Now` gibt das aktuelle Datum inklusive der Uhrzeit aus.

Variante 2 – die Lösung über AddFromFile

Selbstverständlich ist die Variante 1 nur bei kürzeren Makros geeignet. Bei der folgenden zweiten Variante speichern Sie ein Makro in einer Textdatei. Legen Sie zu diesem Zweck über den Notizblock im Windows-Zubehör eine Textdatei mit dem Namen CODE.TXT an und geben Sie dann folgenden Code aus Abbildung 318 ein.

Um die Textdatei *codes.txt* als Makro in das Modul NEUESMODUL einzufügen, starten Sie das Makro aus Listing 511.

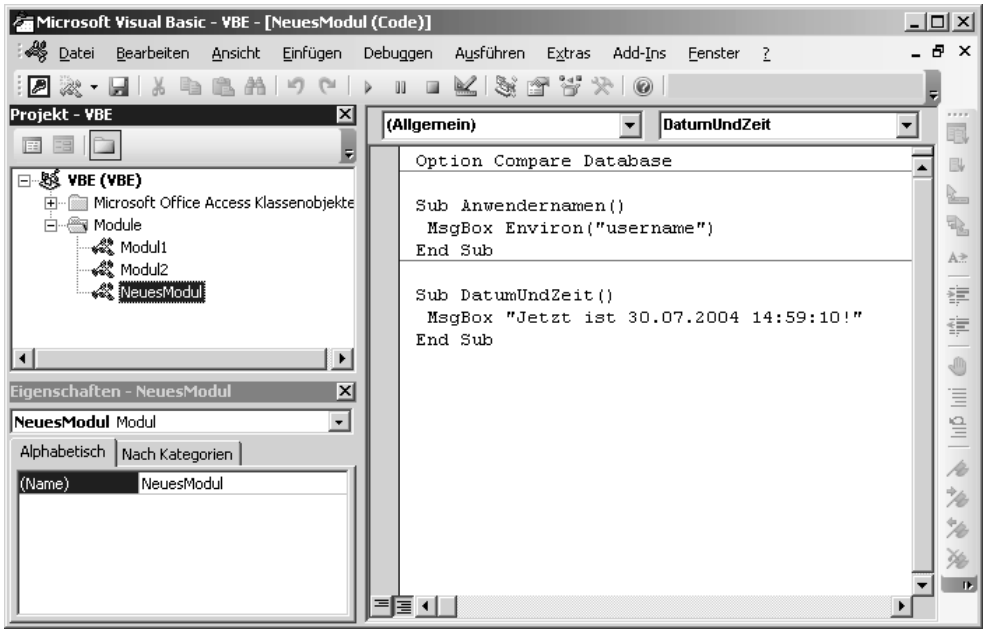


Abbildung 317: Ein kürzeres Makro per Quellcode einfügen

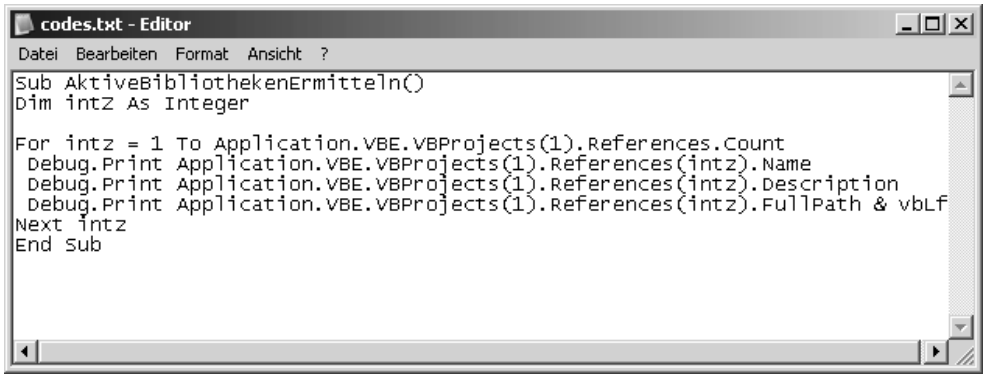


Abbildung 318: Dieses Makro soll importiert werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub MakroAusTextdateiImportieren()
   Dim CodeModul As CodeModule
   Const ImportDatei = "C:\codes.txt"
```

Listing 511: Ein Makro einfügen – Variante 2

```
    On Error GoTo fehler
    Set CodeModul = Application.VBE.ActiveVBProject.VBComponents _
       ("NeuesModul").CodeModule

    With CodeModul
       .AddFromFile ImportDatei
    End With
    Set CodeModul = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 511: Ein Makro einfügen – Variante 2 (Forts.)

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `CodeModule`. Das Objekt `CodeModule` kann Komponenten wie Formulare, Klassen, Module oder Dokumente enthalten. Über die Anweisung `Set` füllen Sie die Variable `CodeModul`, indem Sie den Namen des Zielmoduls abgeben.

Mithilfe der Methode `AddFromFile` fügen Sie dem Modul NEUESMODUL den Quellcode aus der Textdatei CODES.TXT hinzu. Die Syntax dieser Methode lautet:

```
Objekt.AddFromFile(Dateiname)
```

Teil	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `CodeModul`-Objekt oder eine `Reference`-Auflistung zurückgibt.
`Dateiname`	Erforderlich. Ein Zeichenfolgenausdruck, der den Namen der Datei angibt, die Sie dem Projekt oder dem Modul hinzufügen möchten.

Tabelle 152: Die Argumente der Methode AddFromFile

Nach dem Import der Textdatei kann der Objektverweis wieder aufgehoben werden und das Makro über die Anweisung `Exit Sub` auf direktem Wege verlassen werden.

426 Formularcode importieren

Das manuelle Einstellen von einzelnen Ereignissen, das Sie in Kapitel 6 nachschlagen können, ist eine Lösung. Eine zweite Lösung besteht darin, Ereignisse in einem Formular einzustellen und diese dabei aus einer Textdatei zu importieren.

Im Beispiel aus Listing 512 wird die Textdatei *Artikelformular.txt* direkt hinter das Formular ARTIKEL importiert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub FormularCodeImportieren()
   Dim CodeModul As CodeModule
   Const ImportDatei = "C:\Artikelformular.txt"

   On Error GoTo fehler
   Set CodeModul = Application.VBE.ActiveVBProject.VBComponents _
      ("Form_Artikel").CodeModule

   With CodeModul
      .AddFromFile ImportDatei
   End With

   Set CodeModul = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 512: Formularcode importieren

Geben Sie den Codenamen des Formulars bei der `Set`-Anweisung bekannt. Danach importiert die Methode `AddFromFile` den Quellcode aus der Textdatei genau hinter das Formular.

427 Formularereignis einstellen

Für das Einfügen eines einzelnen Ereignisses bietet sich auch die Methode `CreateEventProc` an. Im folgenden Beispiel aus Listing 513 wird dem Formular `Artikel2` ein Ereignis hinzugefügt. Dabei soll beim Starten des Formulars das aktuelle Datum am Bildschirm angezeigt werden.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub EreignisHinzufügen()
   Dim CodeModul As CodeModule
   Dim intz As Integer
```

Listing 513: Ein Ereignis hinzufügen

```
    On Error GoTo fehler
    Set CodeModul = Application.VBE.ActiveVBProject.VBComponents _
       ("Form_Artikel2").CodeModule

    With CodeModul
       intz = .CountOfLines + 1
       intz = .CreateEventProc("Load", "Form") + 1
       .InsertLines intz, " Msgbox ""Hallo " & Date & "!"" "
    End With
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 513: Ein Ereignis hinzufügen (Forts.)

Bevor Sie das Ereignis hinzufügen, ermitteln Sie zunächst, wie viele Codezeilen bereits hinter dem Formular `Artikel2` stehen. Diese Aufgabe erledigen Sie über die Eigenschaft `CountOfLines`. Danach wenden Sie die Methode `CreateEventProc` an, um das Ereignis `Load` einzustellen. Die Syntax dieser Methode lautet:

```
Objekt.CreateEventProc(Ereignisname, Objektname) As Long
```

Teil	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `CodeModul`-Objekt zurückgibt.
`Ereignisname`	Erforderlich. Ein Zeichenfolgenausdruck, der den Namen des Ereignisses angibt, das Sie hinzufügen möchten.
`Objektname`	Erforderlich. Ein Zeichenfolgenausdruck, der den Namen des Objekts angibt, das die Ereignisquelle darstellt.

Tabelle 153: Die Argumente der Methode CreateEventProc

Über die Methode `InsertLines` fügen Sie innerhalb des Ereignisses die gewünschten Zeilen ein

.

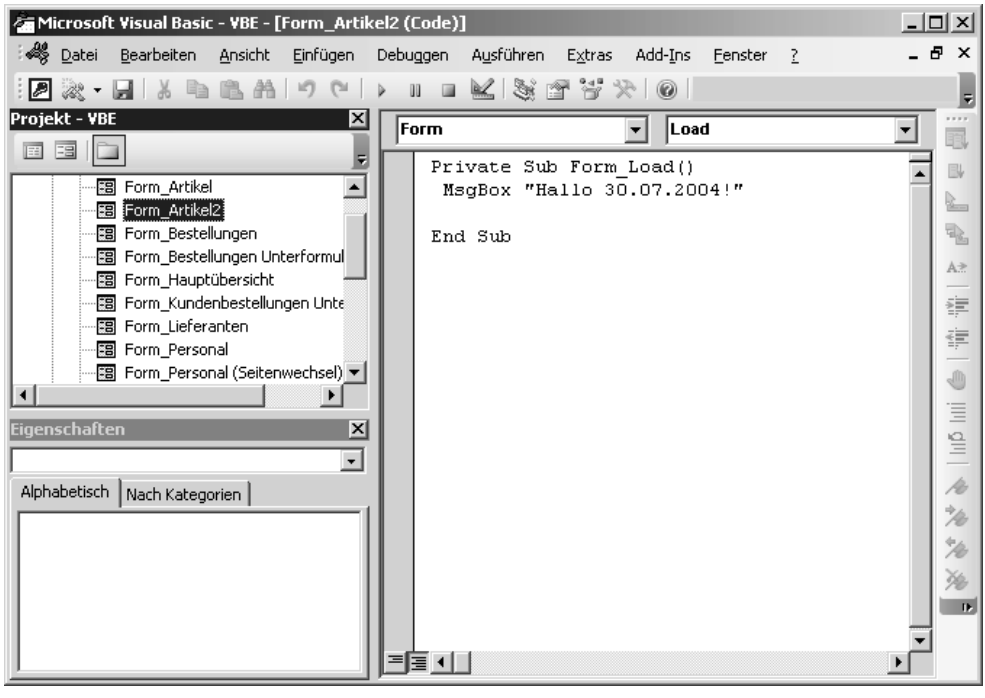


Abbildung 319: Das Ereignis wurde eingefügt.

428 Makros exportieren

Neben dem Import von Textdateien in Modulen ist auch eine Exportfunktion für Module in Access vorgesehen, mit der Sie Ihre Programmierung in Textdateien sichern können.

Über den Einsatz der Methode `Export` können Sie ein Modul in einer Textdatei speichern. Die Syntax dieser Methode lautet:

```
Objekt.Export(Dateiname)
```

Teil	Beschreibung
`Objekt`	Erforderlich. Ein Objektausdruck, der ein `VBComponent`-Objekt zurückgibt.
`Dateiname`	Erforderlich. Ein Wert vom Typ `String`, der den Namen der Datei angibt, in die Sie die Komponente exportieren möchten.

Tabelle 154: Die Argumente der Methode Export

Das Makro aus Listing 514 exportiert alle Makros im Modul MDLVBE in eine Textdatei und speichert diese im Verzeichnis *C:\Eigene Dateien* unter dem Namen *Sicherung.txt*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub ModulInTextdateiSichern()
   Dim VBKomp As Object
   Const ExportDatei = "C:\Eigene Dateien\Sicherung.txt"

   On Error GoTo fehler
   Set VBKomp = Application.VBE.ActiveVBProject.VBComponents("mdlVBE")

   With VBKomp
      .Export ExportDatei
   End With
   Set VBKomp = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 514: Ein Modul in eine Textdatei exportieren

Mit der Methode `Export` sichern Sie alle Makros aus MODUL1 als Textdatei.

429 Formularcodes sichern

Hinter den einzelnen Formularen kann sich Quellcode in Form von Ereignismakros befinden. Diese Codes können Sie ebenso über die Methode `Export` in eine Textdatei überführen.

Das folgende Makro aus Listing 515 rettet alle Ereignismakros, die sich hinter dem Formular `Artikel` befinden, und speichert sie in der Datei *ArtikelFormular* ab.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularInTextdateiSichern()
   Dim VBKomp As Object
   Const ExportDatei = "C:\ArtikelFormular.txt"

   On Error GoTo fehler
   Set VBKomp = _
      Application.VBE.ActiveVBProject.VBComponents("Form_Artikel")
```

Listing 515: Auch Formularcodes können exportiert werden.

```
    With VBKomp
       .Export ExportDatei
    End With

    Set VBKomp = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 515: Auch Formularcodes können exportiert werden. (Forts.)

Geben Sie den Codenamen des Formulars bei der Anweisung `Set` an.

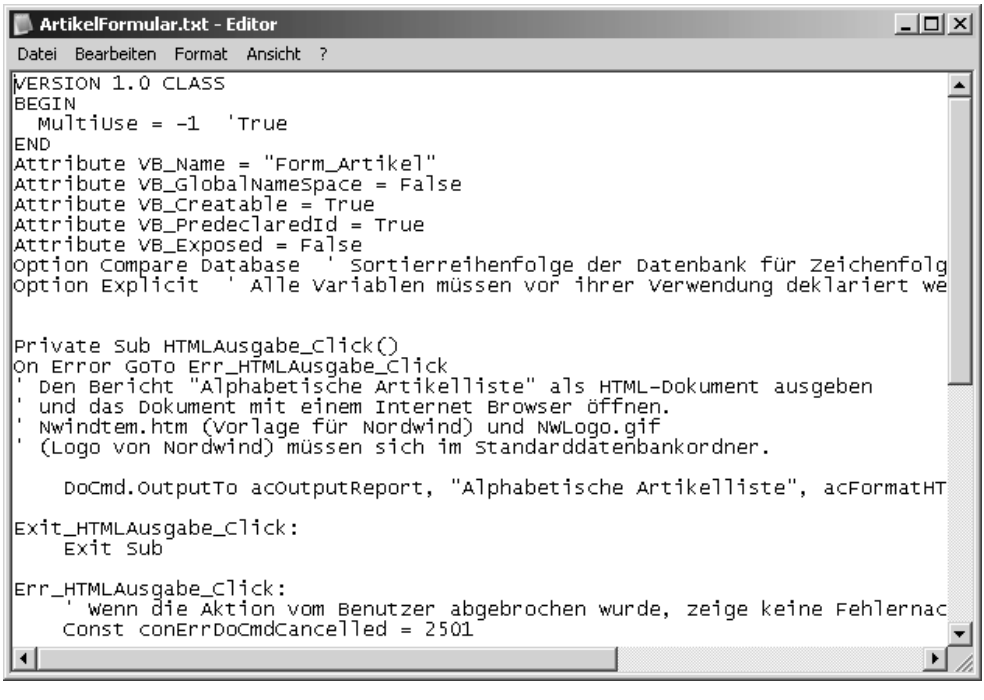


Abbildung 320: Die Ereignisse des Formulars Artikel wurden gesichert.

Die Alternative

Eine alternative Möglichkeit, die ohne den Einsatz von VBE auskommt, sehen Sie in Listing 516. Dabei werden alle Formulare der Datenbank, die Quellcode enthalten, als separate Textdateien gespeichert.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap09
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=================================================

Sub FormularCodeExportieren()
   Dim dbs As Database
   Dim doc As Document
   Dim frm As Form
   Dim strName As String

   Set dbs = CurrentDb
   With dbs.Containers!Forms
      For Each doc In .Documents
         strName = doc.Name
         DoCmd.OpenForm strName, acDesign
         Set frm = Forms(strName)
         If frm.HasModule Then
            DoCmd.OutputTo acModule, Forms(strName).Module, _
               acFormatTXT, "C:\Eigene Dateien\" & strName & ".Txt"
            DoCmd.Close acForm, strName
         Else
            DoCmd.Close acForm, strName
         End If
      Next
   End With
   dbs.Close
   Set dbs = Nothing
End Sub
```

Listing 516: Alle Formularcodes in Textdateien überführen

In einer `for each next`-Schleife durchlaufen Sie alle Dokumente des Containers. Innerhalb der Schleife öffnen Sie die einzelnen Formulare in der Entwurfsansicht und prüfen über die Eigenschaft `HasModul`, ob Quellcode hinter dem Formular liegt. Wenn ja, dann wenden Sie die Methode `OutPutTo` an, um die Codes zu exportieren. Dabei speichern Sie die Dateien im Verzeichnis *C:\Eigene Dateien* unter dem jeweiligen Formularnamen und mit der Endung *txt*. Schließen Sie danach das jeweilige Formular über die Methode `Close`.

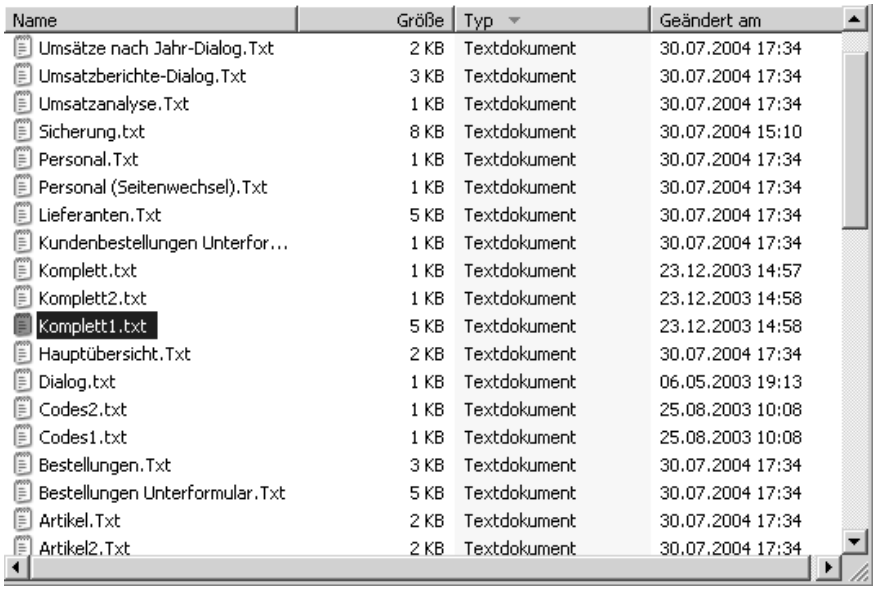


Abbildung 321: Alle »Code-Formulare« wurden exportiert.

Access und ...

In diesem Kapitel können Sie Lösungen nachschlagen, die über einen bloßen Einsatz in Access hinausgehen. Eine Stärke der Programmiersprache VBA ist die universelle Verwendbarkeit im ganzen Office-Paket. So können Sie Daten zwischen den einzelnen Anwendungen problemlos importieren sowie exportieren und somit die Stärke einer jeder Office-Anwendung einsetzen.

430 Early-Binding und Late-Binding

Um von Access aus andere Office-Anwendungen einsetzen zu können, gibt es zwei Vorgehensweisen: Early-Binding und Late-Binding. Bei den in diesem Kapitel vorgestellten Lösungen werden beide Methoden gleichermaßen beliebig eingesetzt.

Early-Binding

Beim Early-Binding werden bereits vor dem eigentlichen Makrostart die später gebrauchten Funktionen in Form einer Bibliothek eingebunden. Dabei wird in der Entwicklungsumgebung aus dem Menü EXTRAS der Befehl VERWEISE ausgewählt und die gewünschte Bibliothek eingebunden. Danach stehen alle Objekte, Methoden und Eigenschaften aus dieser Bibliothek bereits vor dem Makrostart zur Verfügung.

Diese Methode ist die schnellere von beiden. Außerdem stehen Ihnen schon beim Editieren des Quellcodes die neuen Methoden und Eigenschaften sowie die Online-Hilfe zur eingebundenen Bibliothek zur Verfügung.

Late-Binding

Beim Late-Binding wird zur Laufzeit des Makros ein Verweis auf die Bibliothek durchgeführt. Dazu wird im ersten Schritt ganz allgemein eine Objektvariable vom Typ `Object` deklariert. Danach wird die Funktion `CreateObject` eingesetzt, die ein Objekt der gewünschten Anwendung erstellt. Auch danach sind alle Methoden und Eigenschaften zu diesem Objekt verfügbar.

Im Beispiel aus Listing 517 wird ein Word-Objekt erstellt und danach ein Dokument geöffnet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================
```

Listing 517: Eine Word-Sitzung wird erstellt.

```
Sub LateBinding()
   Dim wordObj As Object

   Set wordObj = CreateObject("Word.Application")
      wordObj.Documents.Open "C:\Eigene Dateien\Dokument1.doc"
   'weitere Befehle

   Set wordObj = Nothing
End Sub
```

Listing 517: Eine Word-Sitzung wird erstellt. (Forts.)

431 Outlook bereits gestartet?

Access und das Internet haben einige Berührungspunkte, wie zum Beispiel das Verschicken von E-Mails direkt aus Access heraus. Dazu müssen Sie nicht extra Ihr E-Mail-Programm starten, eine Datenbank anhängen, die Empfänger festlegen und die E-Mail-Nachricht abschicken. Sämtliche Aufgaben können Sie auch direkt in Access erledigen.

Im ersten Beispiel zum Zusammenspiel von Access mit Outlook schreiben Sie die Funktion aus Listing 518, über die Sie feststellen können, ob Outlook bereits gestartet ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Function OutlookOffen() As Boolean
Dim OutApp As Outlook.Application

On Error Resume Next
Set OutApp = GetObject(, "Outlook.Application")

 If Err.Number = 0 Then
   OutlookOffen = True
 Else
   OutlookOffen = False
 End If
End Function
```

Listing 518: Ist Outlook bereits aktiv?

Verwenden Sie die `GetObject`-Funktion für den Zugriff auf die Objektbibliothek von Outlook und weisen Sie das Objekt einer Objektvariablen zu. Verwenden Sie die `Set`-Anweisung, um das von `GetObject` zurückgegebene Objekt der Objektvariablen zuzuweisen. Ist dieser Vorgang erfolgreich, dann meldet das Objekt `Err` den Wert `0`. In diesem Fall ist Outlook bereits gestartet. Als Rückgabewert geben Sie daher den Wahrheitswert `True` an das aufrufende Makro aus Listing 519 zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub StartOutlook()
   Dim OutApp As Object
   Dim objOrdner As Object
   Dim objNameSpace As Object

   If OutlookOffen = False Then
      Set OutApp = CreateObject("Outlook.Application")
      Set objNameSpace = OutApp.GetNamespace("MAPI")
      Set objOrdner = objNameSpace.GetDefaultFolder(olFolderInbox)
      objOrdner.Display
   End If
End Sub
```

Listing 519: Outlook starten oder nicht

Im Makro aus Listing 519 wurde das Late-Binding durchgeführt und einige Objektvariablen wurden deklariert. Meldet die Funktion `OutlookOffen` den Rückgabewert `False`, dann ist Outlook noch nicht geöffnet. In diesem Fall wenden Sie die Funktion `CreateObject` an, um ein Outlook-Objekt zu erstellen. Danach haben Sie Zugriff auf alle Methoden und Eigenschaften des Objekts. Geben Sie im Anschluss daran über die Methode `GetNameSpace` das Mapi-Protokoll an und definieren Sie über die Methode `GetDefaultFolder` den Ordner, den Sie anzeigen möchten.

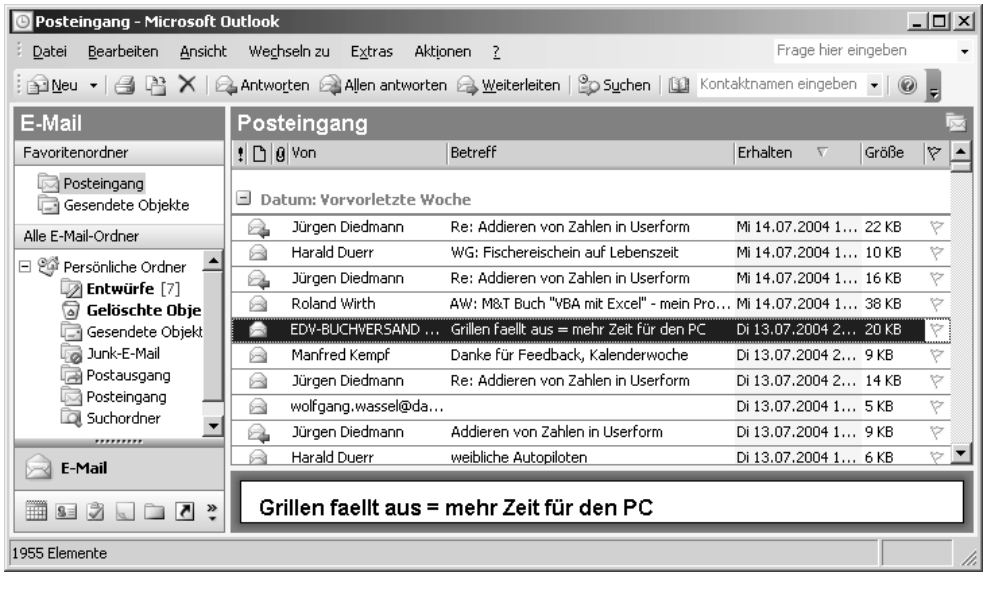


Abbildung 322: Outlook wird nach einer Prüfung gestartet.

Da in diesem Beispiel der Posteingangsordner angezeigt werden soll, geben Sie als Argument der Methode `GetDefaultFolder` die Konstante `olFolderInbox` an. Mithilfe der Eigenschaft `Display` können Sie danach den Posteingangsordner anzeigen.

Weitere Ordner entnehmen Sie der folgenden Tabelle 155.

Konstante	Ordner
`olFolderCalendar`	Kalender
`olFolderContacts`	Kontakte
`olFolderDeletedItems`	Gelöschte Objekte
`olFolderDrafts`	Entwürfe
`olFolderInbox`	Posteingangsordner
`olFolderJournal`	Journal
`olFolderNotes`	Notizen
`olFolderOutbox`	Postausgangsordner
`olFolderSentMail`	Ordner *Gesendete Objekte*
`olFolderTasks`	Aufgaben
`olFolderJunk`	Junk-E-Mails

Tabelle 155: Die wichtigsten Ordner von Outlook

432 E-Mails verschicken

Für das Versenden von E-Mails lässt sich in Access die Methode `SendObject` einsetzen. Dabei können Sie ganz genau festlegen, welchen Bestandteil einer Access-Datenbank Sie versenden möchten. Ferner übergeben Sie dieser Methode die Adressaten sowie den Begleittext der E-Mail.

Die Methode `SendObject` hat folgende Syntax:

```
SendObject(Objekttyp, Objektname, Ausgabeformat, An, Cc, Bcc, Betreff, Nachricht,
NachrichtBearbeiten, Vorlagedatei)
```

Das erste Argument der Methode `SendObject` lautet `Objekttyp`. Dabei geben Sie in einer Konstanten an, welchen Bestandteil der Datenbank Sie per E-Mail versenden möchten. Folgende Konstanten stehen Ihnen dabei zur Verfügung:

- `acSendDataAccessPage`: Eine Access-Datenzugriffsseite wird einem E-Mail-Empfänger zugestellt.
- `acSendForm`: Eine Formular soll über eine E-Mail versendet werden.
- `acSendModule`: Ein Modul wird per E-Mail versendet.
- `acSendNoObject`: Es wird lediglich eine Text-E-Mail ohne Anhang versendet. Es handelt sich dabei um die Standardeinstellung.

- `acSendQuery`: Hierbei soll eine Abfrage per E-Mail versendet werden.
- `acSendReport`: Bei dieser Angabe wird ein Bericht versendet.
- `acSendTable`: Diese Konstante steht für das Versenden einer bestimmten Tabelle aus einer Datenbank.

Im nächsten Argument `Objektname` muss der Name des Objekts angegeben werden, der per E-Mail versendet werden soll.

Mithilfe des Arguments `Ausgabeformat` können Sie festlegen, in welcher Form das Access-Objekt versendet werden soll. Dabei haben Sie unter anderem die Auswahl zwischen folgenden Konstanten:

- `acFormatHTML`: Hier erfolgt eine Ausgabe des Access-Objekts über das HTML-Format, das Sie mit jedem Browser ansehen können.
- `acFormatRTF`: Beim RTF-Format handelt es sich um ein Textformat, das Sie mit nahezu jedem Textverarbeitungsprogramm öffnen können.
- `acFormatTXT`: Dieses Textformat ist mit jedem Texteditor, beispielsweise Notepad im Zubehör von Windows, zu lesen.
- `acFormatXLS`: Dabei handelt es sich um das Excel-Tabellenformat.
- `AcFormatDAP`: Bei dieser Konstante handelt es sich um Datenzugriffsseiten.

Beim Argument `An` müssen Sie die Empfänger auflisten, deren Namen in die AN-Zeile der E-Mail aufgenommen werden sollen. Die Empfängernamen in diesem Argument müssen durch Semikola (;) oder durch jenes Listentrennzeichen voneinander getrennt werden, das auf der Registerkarte ZAHLEN des Dialogfelds LÄNDEREINSTELLUNGEN in der Windows-Systemsteuerung festgelegt ist.

Das Argument `Cc` gibt an, an welche E-Mail-Empfänger Sie die E-Mail als Kopie schicken möchten. Es gelten dabei dieselben Optionen wie auch beim Argument `An`.

Mit dem Argument `Bcc` können Sie E-Mail-Empfängern eine »blinde Kopie« der E-Mail schicken, ohne dass der eigentliche Empfänger der E-Mail, der unter dem Argument `An` angegeben wurde, etwas davon erfährt.

Das Argument `Betreff` repräsentiert die Betreff-Zeile der E-Mail. Geben Sie dort einen Betreff in doppelten Anführungsstrichen an.

Im Argument `Nachricht` geben Sie den Text an, der in die E-Mail eingefügt werden soll. Wenn Sie dieses Argument nicht angeben, wird nur das Objekt, jedoch kein Text, in die E-Mail aufgenommen.

Mithilfe des Arguments `NachrichtBearbeiten` entscheiden Sie, ob Sie die E-Mail direkt absenden oder zur weiteren Bearbeitung vorher öffnen möchten. Setzen Sie dieses Argument auf den Wert `False`, um die Nachricht direkt zu versenden. Setzen Sie das Argument auf den Wert `True`, um die E-Mail zur weiteren Bearbeitung zu öffnen.

Beim Argument `Vorlagedatei` handelt es sich um einen optionalen Wert, der den vollständigen Namen und Pfad der Datei angibt, die als Vorlage für eine HTML-Datei verwendet werden soll.

Im Beispiel aus Listing 520 wird ein Text-E-Mail an mehrere Personen versendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub TextEMailVersenden()
   DoCmd.SendObject , , , "Held-office@t-online.de", _
      "Machero@aol.com", , "Feedback erwünscht", _
      "Sehr geehrter Herr Müller," & vbCrLf & _
      "Bitte schicken Sie mir ein Feedback" & _
      vbCrLf & "zu meiner Anfrage vom 03.08.2004" & _
      vbCrLf & vbCrLf & _
      "Viele Grüße" & vbCrLf & _
      "Bernd Held", True
End Sub
```

Listing 520: E-Mail über ein Makro versenden

Die ersten drei Argumente der Methode `SendObject` werden nicht benötigt. Geben Sie daher drei Kommas nacheinander ein. Nennen Sie dann erst die Empfänger-E-Mail-Adresse und nachfolgend den E-Mail-Empfänger, der die E-Mail als Kopie erhalten soll. Geben Sie dann den Titel sowie den eigentlichen E-Mail-Text an. Um Zeilenumbrüche im E-Mail-Text einzufügen, arbeiten Sie mit der Konstanten `vbCrLf`. Damit wird ein Zeilenumbruch in der E-Mail erzeugt. Setzen Sie das Argument `NachrichtBearbeiten` auf den Wert `True`, um die E-Mail vor dem Versenden für eine weitere Bearbeitung zu öffnen. Ist Outlook bereits geöffnet, wird Ihnen das E-Mail-Fenster direkt angezeigt.

Als Alternative können Sie eine einfache Text-E-Mail auch mithilfe der API-Funktion `ShellExecute` aus Listing 521 versenden.

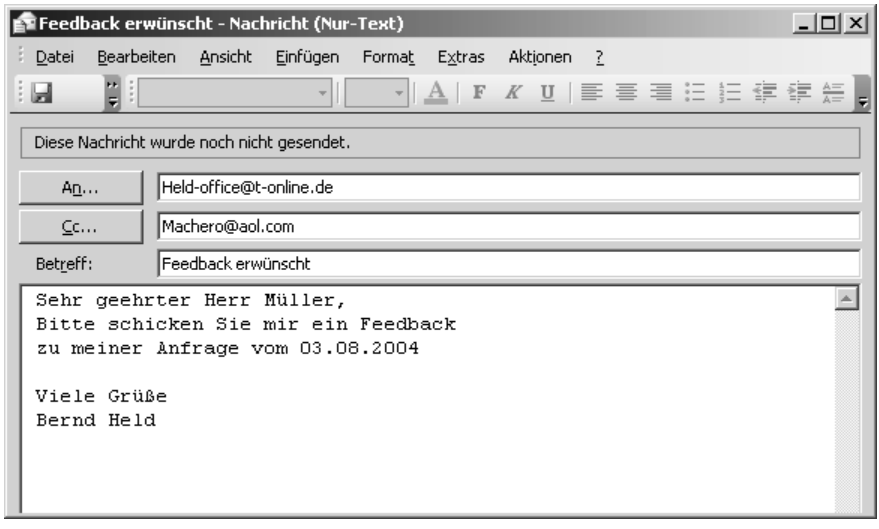


Abbildung 323: Die E-Mail ist vorbereitet und muss nur noch versendet werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Private Declare Function ShellExecute Lib "Shell32.dll" _
   Alias "ShellExecuteA" (ByVal hWnd As Long, _
   ByVal lpOperation As String, ByVal lpFile As String, _
   ByVal lpParameters As String, ByVal lpDirectory As String, _
   ByVal nShowCmd As Long) As Long


Sub Mail(eMail As String, Optional Subject As String, _
   Optional Body As String)
   Call ShellExecute(0&, "Open", "mailto:" + eMail + _
      "?Subject=" + Subject + "&Body=" + Body, "", "", 1)
End Sub


Sub MailSenden()
   Call Mail("Held-office@t-online.de", "Anfrage", "Bitte melden")
End Sub
```

Listing 521: E-Mail über eine API-Funktion versenden

Übergeben Sie der API-Funktion als Argument Ihre E-Mail-Adresse sowie den gewünschten Titel und den Text.

433 E-Mail mit Anhang versenden

Möchten Sie aus einer Access-Datenbank ein bestimmtes Objekt wie beispielsweise eine Tabelle oder gar ein Modul per E-Mail verschicken, dann wenden Sie die Methode `SendObject` an und geben genau an, welches Objekt Sie in welchem Ausgabeformat versenden möchten.

Im folgenden Beispiel aus Listing 522 wird die Tabelle ARTIKEL im Excel-Format an mehrere E-Mail-Empfänger versendet.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=================================================

Sub EMailVersendenMitAnhang()
   DoCmd.SendObject acSendTable, "Artikel", acFormatXLS, _
      "Held-office@t-online.de; Machero@aol.com", _
      , , "Die aktuellen Artikel", _
      "Hallo Kollegen, " & vbCrLf & _
      "Anbei die aktuelle Artikelliste zur Ansicht!" & _
      vbCrLf & "Viele Grüße" & vbCrLf & _
      "Erwin Primeldieter", False
End Sub
```

Listing 522: Eine E-Mail mit Dateianhang versenden

Geben Sie im Argument `Objekttyp` der Methode `SendObject` den Typ des Access-Objekts an. Spezifizieren Sie danach noch den Namen des Objekts und legen Sie das Ausgabeformat fest. Setzen Sie das Argument `NachrichtBearbeiten` auf den Wert `True`, um die E-Mail vor dem Senden noch einmal anzuzeigen.

434 Mail-Adressen aus Outlook herauslesen (Early-Binding)

Im nächsten Beispiel aus Listing 523 werden Sie alle Namen, Vornamen und E-Mail-Adressen aus Ihrem Kontaktordner von Outlook herauslesen und im Direktfenster der Entwicklungsumgebung ausgeben.

Damit es beim Zugriff auf die Kontakte in Outlook zu keinen Problemen kommt, binden Sie in der Entwicklungsumgebung von Access die Bibliothek MICROSOFT OUTLOOK 10.0 OBJECT LIBRARY für Outlook 2002 bzw. MICROSOFT OUTLOOK 11.0 OBJECT LIBRARY für Outlook 2003 im Menü EXTRAS über den Befehl VERWEISE ein.

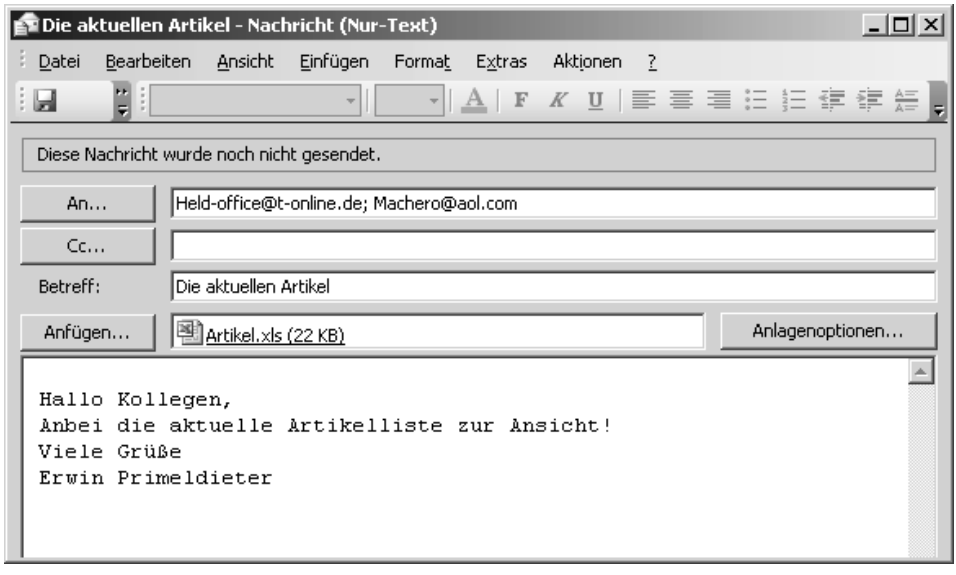


Abbildung 324: An die E-Mail wurde eine Excel-Datei angehängt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub MailAdressenAuslesen()
   Dim outArbeitsVerz As Object
   Dim OutlN As New Outlook.Application
   Dim objKon As Object
   Dim intz As Integer
   Dim strName As String
   Dim strVorname As String
   Dim strMail As String

   On Error GoTo fehler
   Set outArbeitsVerz = OutlN.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

   For intz = 1 To outArbeitsVerz.Items.Count
      Set objKon = outArbeitsVerz.Items(intz)
      strName = objKon.LastName
      strVorname = objKon.FirstName
      strMail = objKon.Email1Address
      Debug.Print strVorname & ", " & strName & "; " & strMail
   Next intz

   Set objKon = Nothing
   Set OutlN = Nothing
```

Listing 523: Kontaktdaten ermitteln (Early-Binding)

```
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 523: Kontaktdaten ermitteln (Early-Binding) (Forts.)

Über die Methode `GetDefaultFolder` können Sie auf den Kontaktordner von Outlook zugreifen. Danach ermitteln Sie über die Funktion `Count`, wie viele Kontakte in diesem Ordner erfasst sind. Mithilfe einer Schleife ermitteln Sie den Nachnamen, den Vornamen sowie die E-Mail-Adresse der jeweiligen Kontaktperson und schreiben diese Informationen über die Anweisung `Debug.Print` in das Direktfenster der Entwicklungsumgebung.

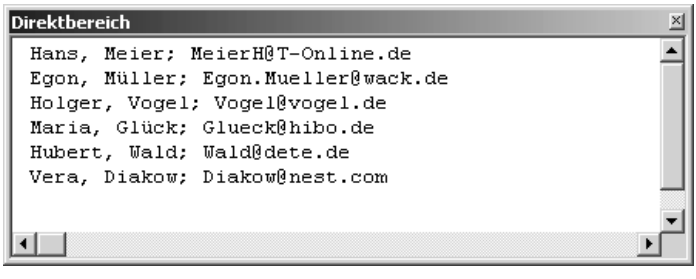


Abbildung 325: Die Kontaktdaten wurden ausgelesen.

435 Mail-Adressen aus Outlook herauslesen (Late-Binding)

Im nächsten Beispiel aus Listing 524 wird das vorherige Beispiel aus Listing 523 noch um ein paar Informationen erweitert. Außerdem wird dieses Mal die Aufgabe über das Late-Binding ausgeführt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub KontaktdatenAuslesen()
   Dim OutlN As Object
   Dim OutArbeitsverz As Object
   Dim OutKon As Object
   Dim intz As Integer
```

Listing 524: Kontaktdaten ermitteln (Late-Binding)

```
    On Error GoTo fehler
    Set OutIN = CreateObject("Outlook.application")
    Set OutArbeitsverz = _
    OutIN.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

    For intz = 1 To OutArbeitsverz.Items.Count
       Set OutKon = OutArbeitsverz.Items(intz)
       With OutKon
          Debug.Print .LastName
          Debug.Print .FirstName
          Debug.Print .BusinessAddressStreet
          Debug.Print .BusinessAddressPostalCode
          Debug.Print .BusinessAddressCity
          Debug.Print .BusinessAddressState
          Debug.Print .BusinessTelephoneNumber
          Debug.Print .BusinessFaxNumber
          Debug.Print .Birthday
          Debug.Print .Email1Address & vbLf
       End With
    Next intz

    Set OutArbeitsverz = Nothing
    Set OutKon = Nothing
    Set OutIN = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
 End Sub
```

Listing 524: Kontaktdaten ermitteln (Late-Binding) (Forts.)

Deklarieren Sie zunächst drei Objektvariablen vom Typ `Object`. Über die Funktion `CreateObject` erstellen Sie danach ein Outlook-Objekt, das Sie der Objektvariablen `OutIn` zuweisen. Jetzt greifen Sie auf den Standard-Kontaktordner von Outlook zu und lesen in einer Schleife alle darin enthaltenen Kontaktdaten aus, die Sie über die diversen Eigenschaften wie beispielsweise `LastName`, `FirstName` usw. abfragen können.

436 Kontaktdaten nach Outlook übertragen

Beim nächsten Beispiel liegen Kontaktdaten in einer Access-Tabelle vor. Diese Tabelle hat folgenden Aufbau.

Die Aufgabe besteht nun darin, die in der Tabelle `Kontakte` enthaltenen Adressen in den Kontaktordner von Outlook zu transferieren. Binden Sie vor dem Start des Makros aus Listing 525 die Objektbibliothek Microsoft Outlook ein.

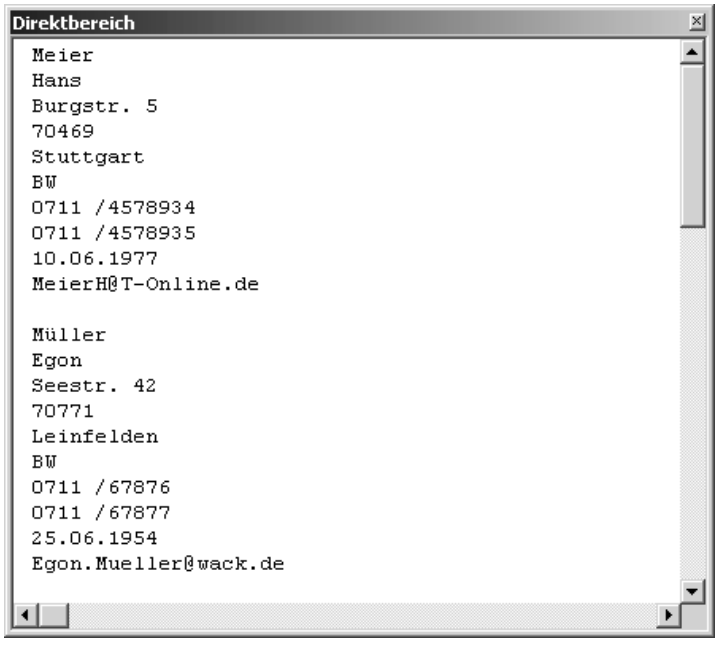


Abbildung 326: Die Kontakte wurden aus Outlook ausgelesen.

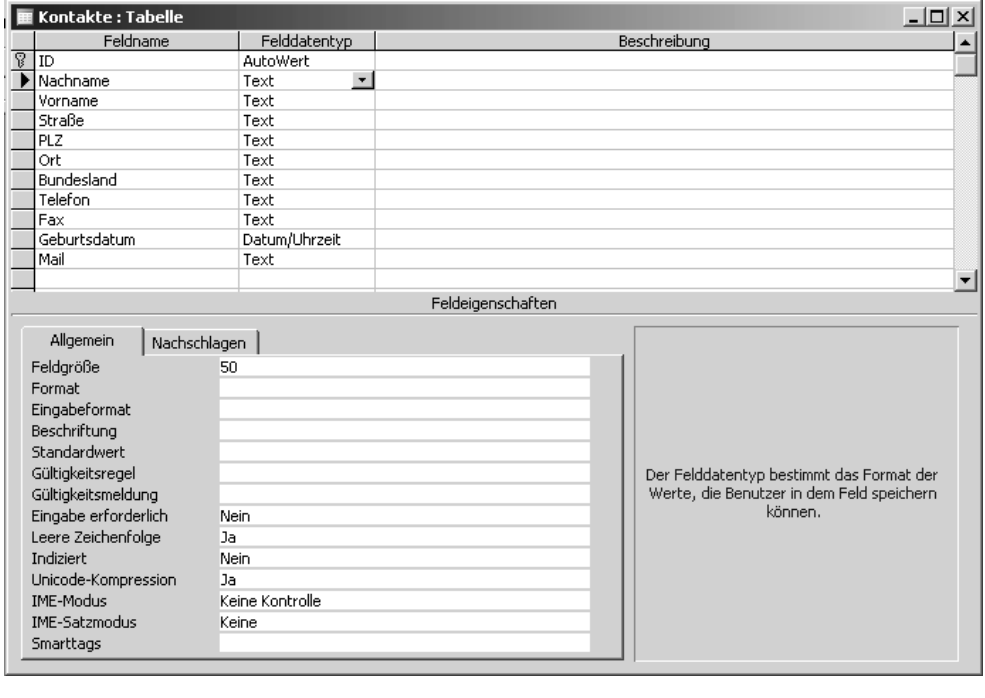


Abbildung 327: Der Aufbau der Tabelle Kontakte

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub KontakteVonAccessInOutlookÜbertragen()
   Dim appOutLook As Outlook.Application
   Dim conoutlook As Outlook.ContactItem
   Dim DBS As New ADODB.Recordset
   Dim intz As Integer

   Set appOutLook = CreateObject("Outlook.Application.11")
   DBS.Open "Kontakte", CurrentProject.Connection
   intz = 0

   On Error GoTo fehler
   Do While Not DBS.EOF
      Set conoutlook = appOutLook.CreateItem(olContactItem)
      conoutlook.Display
      With conoutlook
         .LastName = DBS!Nachname
         .FirstName = DBS!Vorname
         .BusinessAddressStreet = DBS!Straße
         .BusinessAddressPostalCode = DBS!PLZ
         .BusinessAddressCity = DBS!Ort
         .BusinessAddressState = DBS!Bundesland
         .BusinessTelephoneNumber = DBS!Telefon
         .BusinessFaxNumber = DBS!Fax
         .Birthday = DBS!Geburtsdatum
         .Email1Address = DBS!Mail
         .Save
         intz = intz + 1
         DBS.MoveNext
      End With
   Loop

   MsgBox "Es wurden " & intz & " Kontakte übertragen!"
   DBS.Close

   Set conoutlook = Nothing
   Set appOutLook = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 525: Kontaktdaten von Access nach Outlook übertragen

Definieren Sie im ersten Schritt zwei Objektvariablen für Outlook. Die eine Variable `AppOutlook` gibt Ihnen die Möglichkeit, das Mailing-Programm direkt über VBA-Befehle anzusprechen. Die zweite Objektvariable `conoutlook` wählen Sie, um später auf den Kontaktordner von Outlook zugreifen zu können.

Setzen Sie danach die Funktion `CreateObject` ein, um einen Verweis auf die Outlook-Bibliothek zu setzen. Dabei bedeutet die Zahl 11, dass Outlook 2003 gemeint ist. Outlook 2002 weist die Zahl 10 auf. Möchten Sie mit Outlook 2000 arbeiten, tragen Sie hier die Zahl 9 ein. Haben Sie noch Outlook 97 im Einsatz, dann lassen Sie die Zahl weg.

Öffnen Sie im nächsten Schritt Ihre Access-Tabelle `Kontakte` über die Methode `Open`. Setzen Sie danach eine Schleife auf, die so lange durchlaufen wird, bis der letzte Datensatz in der Tabelle abgearbeitet wurde. Damit Satz für Satz verarbeitet werden kann, müssen Sie daran denken, am Ende der Schleife die Methode `MoveNext` einzusetzen.

Mithilfe der Methode `CreateItem` erstellen Sie ein neues Outlook-Objekt. Welches Outlook-Objekt Sie genau brauchen, können Sie über eine Konstante festlegen. Dabei stehen Ihnen folgende Konstanten zur Verfügung:

- `olAppointmentItem`: Mithilfe dieser Konstante fügen Sie einen neuen Termin in Ihrem Terminkalender ein.
- `olContactItem`: Über diese Konstante können Sie einen neuen Kontakt erstellen.
- `olDistributionListItem`: Erstellt einen Eintrag in der Verteilerliste von Outlook.
- `olJournalItem`: Dabei erstellen Sie einen neuen Journaleintrag.
- `olMailItem`: Hierbei können Sie einen neuen E-Mail-Eintrag erzeugen.
- `olNoteItem`: Über diese Konstante legen Sie eine neue Notiz an.
- `olPostItem`: Über diese Konstante können Sie eine E-Mail verschicken.
- `olTaskItem`: Über diese Konstante fügen Sie einen neuen Eintrag in Ihrer Aufgabenliste ein.

Für unser Beispiel erstellen Sie also einen Kontakteintrag und verwenden daher die Konstante `olContactItem`. Der Kontakt wird damit angelegt und wartet auf seine Befüllung. Übertragen Sie die einzelnen Datenfelder aus Ihrer Access-Tabelle, indem Sie folgende Eigenschaften verwenden:

Eigenschaft	Bedeutung
LastName	Nachname der Kontaktperson
FirstName	Vorname der Kontaktperson
BusinessAddressStreet	Straße der Kontaktperson (Arbeitsplatz).
BusinessAddressPostalCode	Postleitzahl des Arbeitgebers
BusinessAddressCity	Geschäftssitz (Ort)
BusinessAddressState	Bundesland des Arbeitsgebers
BusinessTelephoneNumber	Geschäftliche Telefonnummer
BusinessFaxNumber	Faxnummer vom Arbeitsplatz
Birthday	Geburtstag der Kontaktperson
Email1Address	E-Mail-Adresse der Kontaktperson

Tabelle 156: Einige Angaben zur Kontaktperson

Selbstverständlich gibt es noch einige weitere Infos, die Sie in Outlook im Kontaktordner speichern können. Diese finden Sie in der Online-Hilfe von Outlook.

Wenn Sie alle Informationen in den Outlook-Kontakt übertragen haben, verwenden Sie die Methode `Save`, um den Kontakt im Kontaktordner zu speichern. Um am Ende des Makros ausgeben zu können, wie viele Kontakte übertragen wurden, addieren Sie die Zählvariable `i` nach jedem Schleifendurchlauf um den Wert `1`. Wurden alle Datenbankfelder der Tabelle `Kontakte` abgearbeitet, wird die Schleife verlassen. Geben Sie am Ende des Makros in einer Bildschirmmeldung die Anzahl der übertragenen Adressen aus. Vergessen Sie nicht, die Verweise auf die Outlook-Objekte zu entfernen, um den reservierten Arbeitsspeicher wieder freizugeben

437 Kontaktdaten von Outlook importieren

Das folgende Beispiel aus Listing 526 stellt genau den umgekehrten Vorgang wie gerade beschrieben dar. Dabei werden alle Kontaktdaten aus Outlook in die Access-Tabelle `Kontakte` geschrieben. Als Ausgangs-Kontaktordner liegt der Ordner aus Abbildung 328 vor.

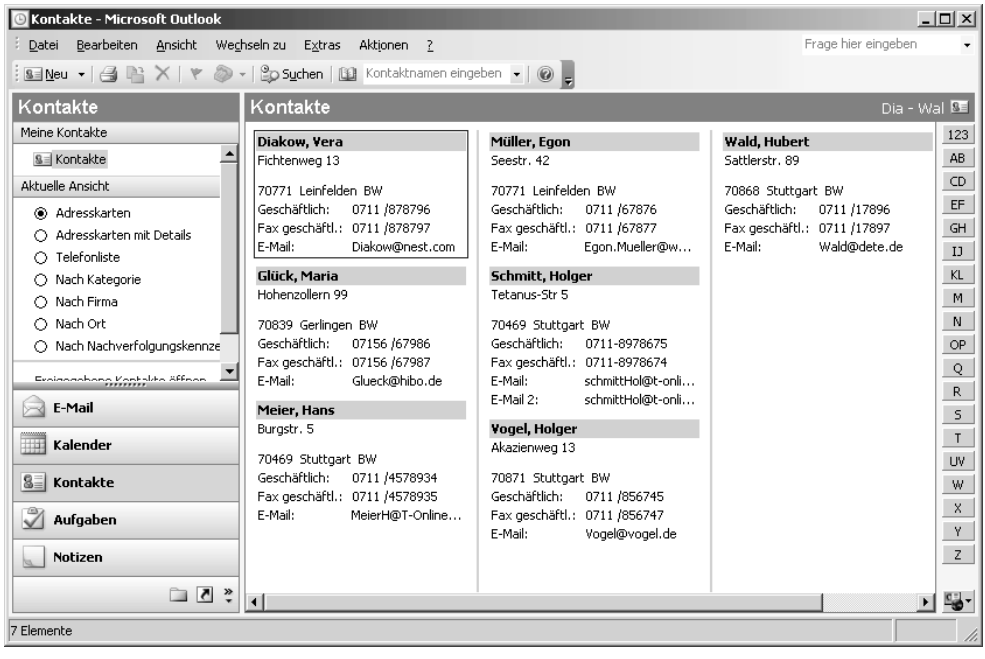


Abbildung 328: Diese Kontakte sollen in eine Access-Tabelle übertragen werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub KontakteAusOutlookÜbernehmen()
   Dim OutlN As Object
   Dim OutArbeitsverz As Object
   Dim OutKon As Object
   Dim DBS As Recordset
   Dim CONN As Database
   Dim intz As Integer
   Dim intNeu As Integer
   Dim intUpd As Integer
   Dim strNachname As String
   Dim strVorname As String

   intz = 0
   intNeu = 0
   intUpd = 0

   Set CONN = CurrentDb
   Set DBS = CONN.OpenRecordset("Kontakte", dbOpenDynaset)
```

Listing 526: Kontaktdaten aus Outlook nach Access transferieren

```
Set OutIN = CreateObject("Outlook.application.11")
Set OutArbeitsverz = _
OutIN.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

For intz = 1 To OutArbeitsverz.Items.Count
Set OutKon = OutArbeitsverz.Items(intz)
strNachname = OutKon.LastName
strVorname = OutKon.FirstName
strNachname = "Nachname = '" & strNachname & "'"

With OutKon
   DBS.FindFirst strNachname
   If DBS.NoMatch Then
      DBS.AddNew
      DBS!Nachname = .LastName
      DBS!Vorname = .FirstName
      DBS!Straße = .BusinessAddressStreet
      DBS!PLZ = .BusinessAddressPostalCode
      DBS!Ort = .BusinessAddressCity
      DBS!Bundesland = .BusinessAddressState
      DBS!Telefon = .BusinessTelephoneNumber
      DBS!Fax = .BusinessFaxNumber
      DBS!Geburtsdatum = .Birthday
      DBS!Mail = .Email1Address
      intNeu = intNeu + 1
   Else
      If DBS!Vorname = strVorname Then
         DBS.Edit
         DBS!Straße = .BusinessAddressStreet
         DBS!PLZ = .BusinessAddressPostalCode
         DBS!Ort = .BusinessAddressCity
         DBS!Bundesland = .BusinessAddressState
         DBS!Telefon = .BusinessTelephoneNumber
         DBS!Fax = .BusinessFaxNumber
         DBS!Geburtsdatum = .Birthday
         DBS!Mail = .Email1Address
         intUpd = intUpd + 1
      Else
         DBS.AddNew
         DBS!Nachname = .LastName
         DBS!Vorname = .FirstName
         DBS!Straße = .BusinessAddressStreet
         DBS!PLZ = .BusinessAddressPostalCode
         DBS!Ort = .BusinessAddressCity
         DBS!Bundesland = .BusinessAddressState
         DBS!Telefon = .BusinessTelephoneNumber
         DBS!Fax = .BusinessFaxNumber
         DBS!Geburtsdatum = .Birthday
         DBS!eMail = .Email1Address
         intNeu = intNeu + 1
      End If
```

Listing 526: Kontaktdaten aus Outlook nach Access transferieren (Forts.)

```
         End If
      End With
      DBS.Update
   Next intz
   MsgBox "Datentransfer erfolgreich beendet! " & vbLf & _
      "Es wurden " & intNeu & " Sätze angelegt, " & vbLf & _
      "Es wurden " & intUpd & " Sätze upgedatet!"
   DBS.Close

   Set OutKon = Nothing
   Set OutArbeitsverz = Nothing
   Set OutlN = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

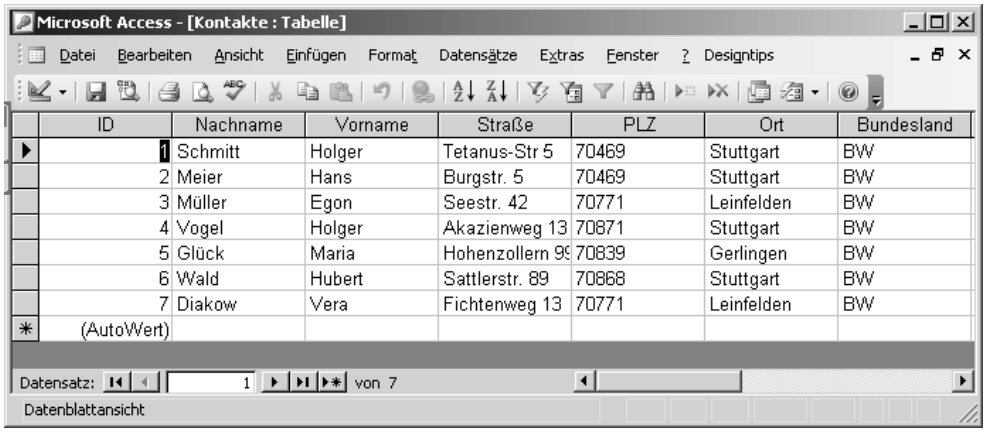
Listing 526: Kontaktdaten aus Outlook nach Access transferieren (Forts.)

Das Makro ermittelt zuerst, wie viele Kontakte im Outlook-Kontaktordner angelegt sind. In Abhängigkeit davon wird eine Schleife durchlaufen, die alle Kontakte aus der lokalen Outlook-Datei (*Outlook.pst*) in die zentrale Access-Kontakttabelle überträgt. Dabei wird geprüft, ob der zu übertragende Satz sich bereits in der Access-Kontakttabelle befindet. Als Prüfkriterium wird der Name herangezogen, welcher in der Variablen `strNachname` zwischengespeichert wird. Mit der `FindFirst`-Methode wird der erste Datensatz in der Tabelle gesucht, welcher dem Kriterium entspricht. Wird kein zutreffender Satz gefunden, gibt die Eigenschaft `NoMatch` den Rückgabewert `True` zurück. In diesem Fall muss der komplette Kontakt aus Outlook in die Access-Kontakttabelle übernommen werden. Liefert die Eigenschaft `NoMatch` hingegen den Wert `False` zurück, entspricht das Suchkriterium `NAME` einem bereits angelegten Namen. Nun ist zu prüfen, ob der `VORNAME` aus Outlook mit dem Vornamen in der Kontakttabelle übereinstimmt. Wenn dem so ist, darf kein neuer Satz angelegt werden. Es darf lediglich ein Update des bereits bestehenden Satzes erfolgen. Als Informationen für ein Update sind hier die Felder `Straße`, `PLZ`, `Ort`, `Bundesland`, `Telefon` und `Fax` vorgesehen. Das Update wird eingeleitet über die Methode `Edit`, welche den aktuellen Datensatz aus dem aktualisierbaren `Recordset`-Objekt in den Kopierpuffer kopiert, damit er anschließend bearbeitet werden kann. Jetzt werden die einzelnen Informationen übertragen.

Stimmt hingegen der vorher in der Variablen `strVorname` gespeicherte Vornamen nicht mit dem Outlook-`VORNAMEN` überein, muss der Satz in der Kontakttabelle neu angelegt werden. Dies geschieht über die Methode `AddNew`. Gleich danach werden alle Kontaktfelder aus Outlook in den Kopierpuffer hineinkopiert.

Egal, ob es sich um ein Update oder eine Neuanlage handelt – erst die Methode `Update` sorgt dafür, dass der Inhalt des Kopierpuffers letztendlich in den Datensatz geschrieben wird. Für den Update-Fall sowie den Neuanlage-Fall werden zwei verschiedene Zähler verwendet, welche am Ende des Datenaustauschs am Bildschirm ausgegeben werden. Um den Speicher am

Ende wieder freizugeben, verwenden Sie das Schlüsselwort `Nothing`. Damit heben Sie die Verbindung der Objektvariablen zu den zugehörigen Objekten wieder auf.



ID	Nachname	Vorname	Straße	PLZ	Ort	Bundesland
1	Schmitt	Holger	Tetanus-Str 5	70469	Stuttgart	BW
2	Meier	Hans	Burgstr. 5	70469	Stuttgart	BW
3	Müller	Egon	Seestr. 42	70771	Leinfelden	BW
4	Vogel	Holger	Akazienweg 13	70871	Stuttgart	BW
5	Glück	Maria	Hohenzollern 99	70839	Gerlingen	BW
6	Wald	Hubert	Sattlerstr. 89	70868	Stuttgart	BW
7	Diakow	Vera	Fichtenweg 13	70771	Leinfelden	BW
(AutoWert)						

Abbildung 329: Die Kontakte wurden erfolgreich übertragen.

438 Termine aus dem Outlook-Kalender abfragen

Im nächsten Beispiel aus Listing 527 werden Termine aus dem Outlook-Kalender ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub TermineAuslesen()
   Dim OutIn As Object
   Dim OutKal As Object
   Dim OutTermin As Object
   Dim intz As Integer

   Set OutIn = CreateObject("Outlook.Application.11")
   Set OutKal = _
   OutIn.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar)

   For intz = 1 To OutKal.Items.Count
      Set OutTermin = OutKal.Items(intz)
      With OutTermin
         Debug.Print .Subject
         Debug.Print .Body
         Debug.Print .Categories
         Debug.Print .Start & vbLf
      End With
```

Listing 527: Termine aus dem Outlook-Kalender abfragen

```
    Next intz

    MsgBox "Es wurden " & intz & " Termine gefunden!"

    Set OutTermin = Nothing
    Set OutTasks = Nothing
    Set OutIn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 527: Termine aus dem Outlook-Kalender abfragen (Forts.)

Als Standardordner zapfen Sie in Outlook nun den Kalenderordner an, der über die Konstante `olFolderCalendar` angesprochen werden kann. Danach arbeiten Sie alle eingetragenen Termine ab und geben diese im Direktfenster der Entwicklungsumgebung aus.

Die typischen Eigenschaften, die Sie beim Terminkalender einsetzen können, entnehmen Sie bitte der folgenden Tabelle 157.

Eigenschaft	Beschreibung
`Subject`	Mit dieser Eigenschaft legen Sie den Betreff des Termins fest.
`Body`	Hier können Sie eine nähere Beschreibung hinterlegen, die im Textkörper ausgegeben wird.
`RequiredAttendees`	Damit werden die an dem Termin beteiligten Personen aufgelistet.
`Location`	Diese Eigenschaft legt den Ort des Termins fest.
`Categories`	Über diese Eigenschaft können Sie den Termin in einer Obergruppe zusammenfassen.
`Start`	Hiermit legen Sie das Startdatum sowie die Startzeit fest.
`End`	Über diese Eigenschaft können Sie das Ende eines Termins bestimmen.
`Duration`	Über diese Eigenschaft geben Sie die Dauer der Besprechung bzw. des Termins in Minuten an.
`ReminderMinutesBeforeStart`	Gibt die Zahl der Minuten an, die eine Erinnerung vor dem Beginn eines Termins auf dem Bildschirm angezeigt werden soll.

Tabelle 157: Einige Angaben zum Terminkalender

Eigenschaft	Beschreibung
ReminderPlaySound	Mithilfe dieser Eigenschaft können Sie die Erinnerungsmeldung auf dem Bildschirm noch zusätzlich mit einem Sound untermalen lassen.
ReminderSet	Über diese Eigenschaft schalten Sie die Erinnerungsfunktion ein.

Tabelle 157: Einige Angaben zum Terminkalender (Forts.)

439 Aufgaben abfragen

Im folgenden Beispiel aus Listing 528 werden alle Aufgaben aus der Aufgabenliste von Outlook abgefragt und im Direktfenster der Entwicklungsumgebung ausgegeben. Bei der folgenden Aufgabe wird von der Liste aus Abbildung 330 ausgegangen.

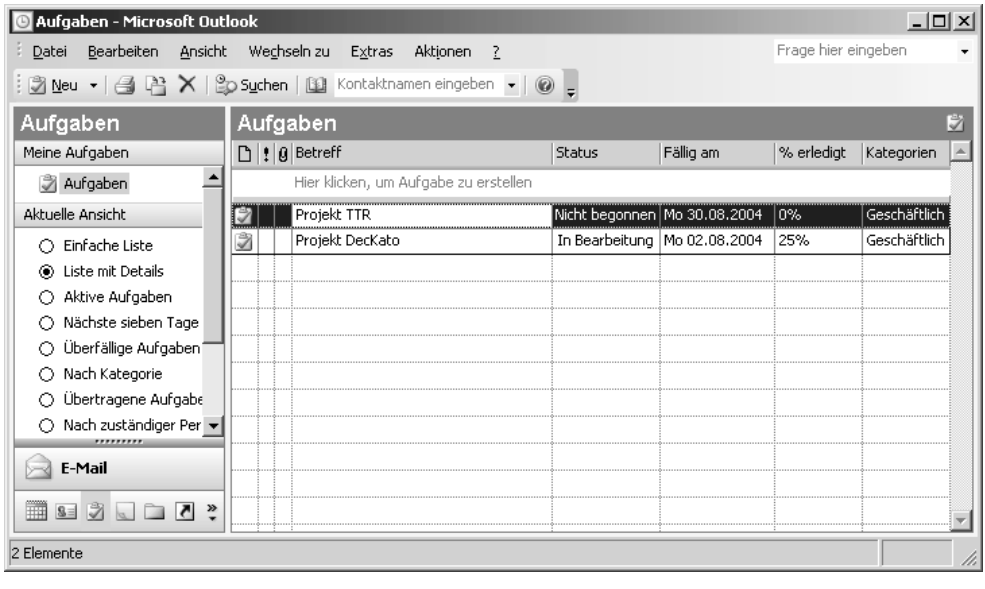


Abbildung 330: Diese Aufgabenliste wird ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub AufgabenAuslesen()
   Dim OutIn As Object
   Dim OutTask As Object
   Dim OutAufgabe As Object
```

Listing 528: Aufgaben aus Outlook auslesen

```
    Dim intz  As Integer

    Set OutIn = CreateObject("Outlook.Application.11")
    Set OutTask = _
    OutIn.GetNamespace("MAPI").GetDefaultFolder(olFolderTasks)

    For intz = 1 To OutTask.Items.Count
       Set OutAufgabe = OutTask.Items(intz)
       With OutAufgabe
          Debug.Print "Titel: " & .Subject
          Debug.Print "Einzelheiten : " & .Body
          Debug.Print "Fertigstellung: " & .Complete
          Debug.Print "Startdatum: " & .StartDate
          Debug.Print "Fälligkeit: " & .DueDate
          Debug.Print "Kategorie: " & .Categories
          Debug.Print "Erinnerung: " & .ReminderTime & vbLf
       End With
    Next intz

    Debug.Print "Es wurden " & intz & " Aufgaben gefunden!"
    Set OutAufgabe = Nothing
    Set OutTask = Nothing
    Set OutIn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 528: Aufgaben aus Outlook auslesen (Forts.)

Möchten Sie die Aufgabenliste von Outlook ansprechen, dann müssen Sie der Methode die Konstante `olFolderTasks` der Methode `GetDefaultFolder` mitgeben. Damit haben Sie unter anderem Zugriff auf die Eigenschaften, die in der folgenden Tabelle aufgelistet werden.

Eigenschaft	Beschreibung
`Subject`	Durch diese Eigenschaft können Sie den Betreff der Aufgabe festlegen.
`Body`	Hier können Sie eine nähere Beschreibung hinterlegen, die im Textkörper ausgegeben wird.
`Complete`	Diese Eigenschaft setzen Sie auf den Wert `True`, wenn Sie die Aufgabe bereits abgeschlossen haben.
`StartDate`	Über diese Eigenschaft legen Sie den Starttermin der Aufgabe fest.
`DueDate`	Mithilfe dieser Eigenschaft bestimmen Sie den Fälligkeitstermin Ihrer Aufgabe.

Tabelle 158: Einige Eigenschaften zur Aufgabenliste von Outlook

Eigenschaft	Beschreibung
ActualWork	Über diese Eigenschaft können Sie einen Wert festlegen, der den Ist-Aufwand Ihrer Aufgabe in Minuten angibt.
Categories	Über diese Eigenschaft können Sie den Termin in einer Obergruppe zusammenfassen.
Complete	Dieser Eigenschaft geben Sie den Wert True, wenn die Aufgabe abgeschlossen ist.
ReminderMinutesBeforeStart	Gibt die Zahl der Minuten an, die eine Erinnerung vor dem Beginn einer Aufgabe auf dem Bildschirm angezeigt werden soll.
ReminderPlaySound	Mithilfe dieser Eigenschaft können Sie die Erinnerungsmeldung auf dem Bildschirm noch zusätzlich mit einem Sound untermalen lassen.
ReminderSet	Über diese Eigenschaft schalten Sie die Erinnerungsfunktion ein.
ReminderSoundFile	Mithilfe dieser Eigenschaft können Sie eine andere Sounddatei angeben, die erklingen soll, wenn Sie das Erinnerungsfenster an eine Aufgabe erinnern soll.

Tabelle 158: Einige Eigenschaften zur Aufgabenliste von Outlook (Forts.)

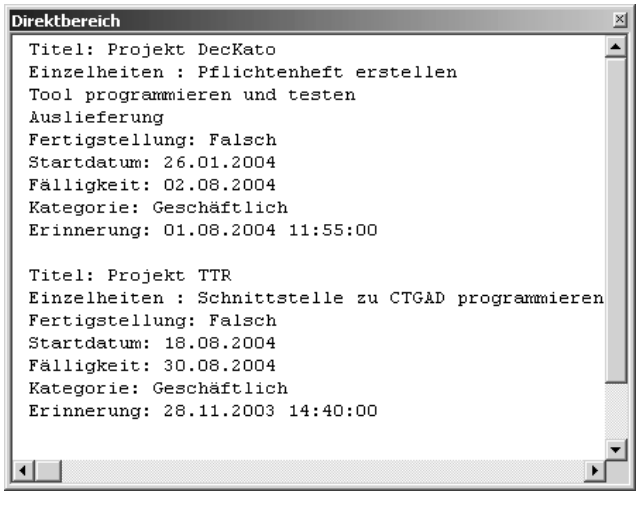


Abbildung 331: Die Aufgaben wurden direkt aus Outlook extrahiert.

440 E-Mails in Access-Tabelle schreiben

Möchten Sie Ihre eingegangenen E-Mails aus dem Outlook-Posteingangsordner in eine Datenbank überführen und auswerten? Dann können Sie eine Tabelle anlegen und die

E-Mail-Informationen aus Outlook in diese Tabelle per VBA-Makro einfügen. Dabei sollen folgende Felder dokumentiert werden:

- Titel der E-Mail
- Name des Absenders der E-Mail
- Inhalt der Mail
- Datum/Uhrzeit des E-Mail-Eingangs
- Anzahl der Dateianhänge der E-Mail
- Lesestatus der E-Mail

Legen Sie zunächst eine Tabelle nach dem Vorbild aus Abbildung 332 an.

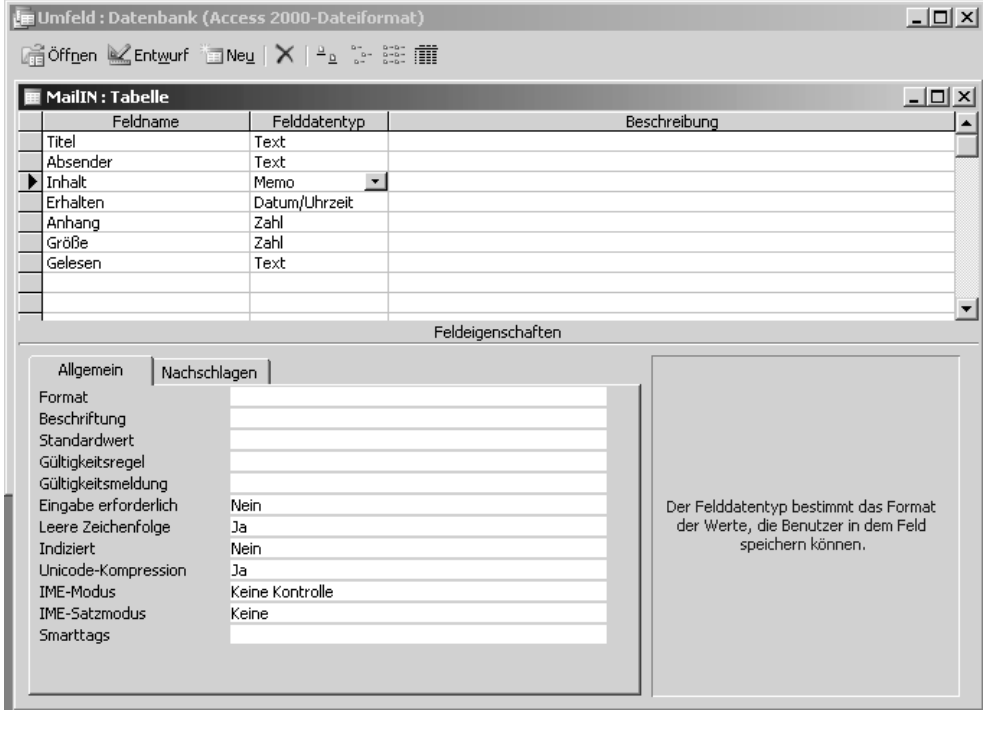


Abbildung 332: Der Aufbau der Tabelle MailIn

Erfassen Sie jetzt das Makro aus Listing 529, welches die Tabelle `MailIn` öffnet, im Hintergrund auf Ihren Posteingangsordner zugreift und die einzelnen E-Mail-Informationen dort einträgt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub MailsSichern()
   Dim Out As Outlook.MAPIFolder
   Dim intGes As Integer
   Dim intz As Integer
   Dim Conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset

   On Error GoTo fehler
   Set Conn = CurrentProject.Connection
   Set DBS = New ADODB.Recordset
   DBS.Open "MailIN", Conn, adOpenKeyset, adLockOptimistic

   Set Out = GetObject("", "Outlook.Application").GetNamespace _
      ("MAPI").GetDefaultFolder(olFolderInbox)

   intGes = Out.Items.Count
   intz = 0

   For intz = 1 To intGes
      With Out.Items(intz)
         DBS.AddNew
         DBS!Titel = .Subject
         DBS!Absender = .SenderName
         DBS!Inhalt = .Body
         DBS!Erhalten = Format(.ReceivedTime, "dd.mm.yyyy hh:mm")
         DBS!Anhang = .Attachments.Count
         DBS!Größe = .Size
         If Not .UnRead = -1 Then
            DBS!Gelesen = "JA"
         Else
            DBS!Gelesen = "Nein"
         End If
         DBS.Update
      End With
   Next intz

   DBS.Close

   With DBS
      .CursorLocation = adUseClient
      .Open "MailIN", Conn, adOpenKeyset, adLockOptimistic
      .Sort = "Erhalten ASC"
   End With

   Set Out = Nothing
```

Listing 529: Alle Mails aus dem Posteingang werden in eine Access-Tabelle geschrieben.

```
    Set DBS = Nothing
    Set Conn = Nothing
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 529: Alle Mails aus dem Posteingang werden in eine Access-Tabelle geschrieben. (Forts.)

Über das Objekt `CurrentProject` öffnen Sie die gerade angelegte Tabelle über die Methode `Open`.

Mithilfe der Methoden `GetObject` und `GetDefaultFolder`, der Sie die Konstante `olFolderInbox` übergeben, gewinnen Sie direkten Zugriff auf Ihren Posteingangs-Ordner in Outlook.

Über die Eigenschaft `Count` zählen Sie zunächst, wie viele E-Mails (Items) sich im Posteingang befinden. Arbeiten Sie danach alle E-Mails ab, indem Sie eine Schleife einsetzen. Innerhalb der Schleife setzen Sie die Methode `AddNew` ein, um einen neuen Datensatz anzulegen. Danach fragen Sie die Betreff-Zeile über die Eigenschaft `Subject` ab und speichern diese Information im Datenbankfeld `Titel`.

Über die Eigenschaft `SenderName` finden Sie heraus, von wem die E-Mail an Sie gesendet wurde. Sichern Sie diese Information in das Datenbankfeld ABSENDER.

Mithilfe der Eigenschaft `Body` können Sie den kompletten E-Mail-Text auslesen.

Die Eigenschaft `ReceivedTime` liefert Ihnen das genaue Datum sowie die Uhrzeit des E-Mail-Eingangs. Diese Zeitangaben können Sie über die Funktion `Format` in das gewünschte Format überführen und im Datenbankfeld ERHALTEN ablegen.

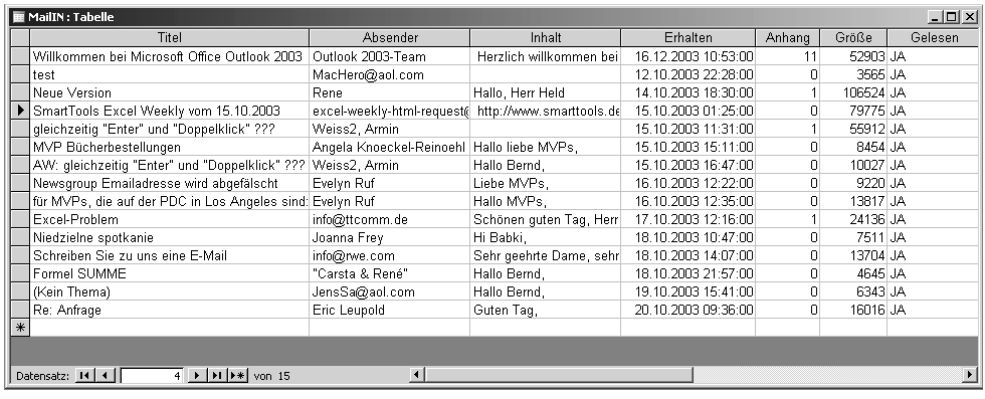
Mithilfe der Eigenschaft `Count`, welche Sie auf das Objekt `Attachments` anwenden, erfahren Sie, wie viele Anhänge in der E-Mail vorhanden waren. Diesen Wert übertragen Sie in das Datenbankfeld `Anhang`.

Die Eigenschaft `UnRead` liefert den Wert `-1`, sofern eine E-Mail noch nicht gelesen wurde. In Abhängigkeit dieses Werts schreiben Sie entweder den Text `Ja` bzw. den Text `Nein` in das Datenbankfeld `Gelesen`.

Setzen Sie die Methode `Update` ein, um den jeweils angelegten Datensatz tatsächlich auch dauerhaft zu sichern.

Wenn alle E-Mails abgearbeitet wurden, verlässt Access die Schleife und wendet die Methode `Close` an, um die Datenbanktabelle zu schließen.

Öffnen Sie die Tabelle gleich wieder direkt im Anschluss und setzen Sie die Eigenschaft `Sort` ein, um die dokumentierten E-Mails nach dem Posteingangsdatum zu sortieren.



MailIN : Tabelle

Titel	Absender	Inhalt	Erhalten	Anhang	Größe	Gelesen
Willkommen bei Microsoft Office Outlook 2003	Outlook 2003-Team	Herzlich willkommen bei	16.12.2003 10:53:00	11	52903	JA
test	MacHero@aol.com		12.10.2003 22:28:00	0	3565	JA
Neue Version	Rene	Hallo, Herr Held	14.10.2003 18:30:00	1	106524	JA
SmartTools Excel Weekly vom 15.10.2003	excel-weekly-html-request@	http://www.smarttools.de	15.10.2003 01:25:00	0	79775	JA
gleichzeitig "Enter" und "Doppelklick" ???	Weiss2, Armin		15.10.2003 11:31:00	1	55912	JA
MVP Bücherbestellungen	Angela Knoeckel-Reinoehl	Hallo liebe MVPs,	15.10.2003 15:11:00	0	8454	JA
AW: gleichzeitig "Enter" und "Doppelklick" ???	Weiss2, Armin	Hallo Bernd,	15.10.2003 16:47:00	0	10027	JA
Newsgroup Emailadresse wird abgefälscht	Evelyn Ruf	Liebe MVPs,	16.10.2003 12:22:00	0	9220	JA
für MVPs, die auf der PDC in Los Angeles sind:	Evelyn Ruf	Hallo MVPs,	16.10.2003 12:35:00	0	13817	JA
Excel-Problem	info@ttcomm.de	Schönen guten Tag, Herr	17.10.2003 12:16:00	1	24136	JA
Niedzielne spotkanie	Joanna Frey	Hi Babki,	18.10.2003 10:47:00	0	7511	JA
Schreiben Sie zu uns eine E-Mail	info@rwe.com	Sehr geehrte Dame, sehr	18.10.2003 14:07:00	0	13704	JA
Formel SUMME	"Carsta & René"	Hallo Bernd,	18.10.2003 21:57:00	0	4645	JA
(Kein Thema)	JensSa@aol.com	Hallo Bernd,	19.10.2003 15:41:00	0	6343	JA
Re: Anfrage	Eric Leupold	Guten Tag,	20.10.2003 09:36:00	0	16016	JA

Datensatz: 4 von 15

Abbildung 333: Alle Mails werden in Access gesichert.

441 Sammel-Mails versenden

Bei der folgenden Lösung aus Listing 530 werden gleich mehrere Mails an unterschiedliche Adressaten geschickt. Dabei werden die einzelnen E-Mail-Adressen aus der Access-Tabelle `Kontakte` entnommen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub MehrereEMailsVersenden()
   Dim OutVerz As Object
   Dim OutMail As Object
   Dim OutMapi As New Outlook.Application
   Dim CONN As Database
   Dim DBS As Recordset
   Dim strText As String
   Const Titel = "ACHTUNG"
   Const MailText = "Auf dem Server befinden sich neue Updates!"

   On Error GoTo fehler
   Set OutVerz = OutMapi.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

   Set CONN = CurrentDb()
   strText = "SELECT Mail from Kontakte"
   Set DBS = CONN.OpenRecordset(strText)

   Do Until DBS.EOF
      If Not IsNull(DBS!Mail) Then
         Set OutMail = CreateItem(olMailItem)
         With OutMail
```

Listing 530: Eine Sammel-Mail versenden

```
            .Subject = Titel
            .Body = MailText
            .Attachments.Add "C:\Eigene Dateien\Info.txt"
            .To = DBS!Mail
            .Save
         End With
      End If
      DBS.MoveNext
   Loop

   Set OutVerz = Nothing
   Set OutMail = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 530: Eine Sammel-Mail versenden (Forts.)

Den Titel sowie den eigentlichen Text der E-Mail können Sie als Konstanten zu Beginn des Makros definieren. Danach stellen Sie die Verbindung zur Tabelle her und formulieren die SQL-Abfrage zunächst in einer String-Variablen. Öffnen Sie die Datentabelle über die Methode `OpenRecordSet` und durchlaufen Sie alle Datensätze dieser Tabelle.

In einer Schleife arbeiten Sie alle Datensätze der Tabelle `Kontakte` ab. Innerhalb der Schleife wenden Sie die Funktion `IsNull` an. Verhindern Sie, dass Datensätze, die keine E-Mail-Adresse beinhalten, nicht mit verarbeitet werden. Danach erstellen Sie über die Methode `CreateItem` eine neue, noch leere Mail, indem Sie der Methode die Konstante `olMailItem` zuweisen. Über die Eigenschaft `Subject` können Sie den Titel der E-Mail festlegen. Die Eigenschaft `Body` wird dazu eingesetzt, die eigentliche Nachricht abzusetzen. Über die Methode `Add`, die auf das Objekt `Attachment` angewendet wird, wird die Meldung noch als Dateianhang zusätzlich an die Mail angehängt. Mithilfe der Methode `Save` speichern Sie zunächst alle Mails im Outlook-Ordner ENTWÜRFE. Von dort können sie dann nach einer kurzen Überprüfung zu einem späteren Zeitpunkt direkt aus Outlook versendet werden (siehe Abbildung 334).

442 E-Mail-Anhänge speichern

Bei der folgenden Lösung aus Listing 531 handelt es sich um einen Automatismus, der auf den Posteingangsordner von Outlook zugreift und alle Mails abarbeitet. Dabei werden alle Mailanhänge in einem bestimmten Ordner gespeichert.

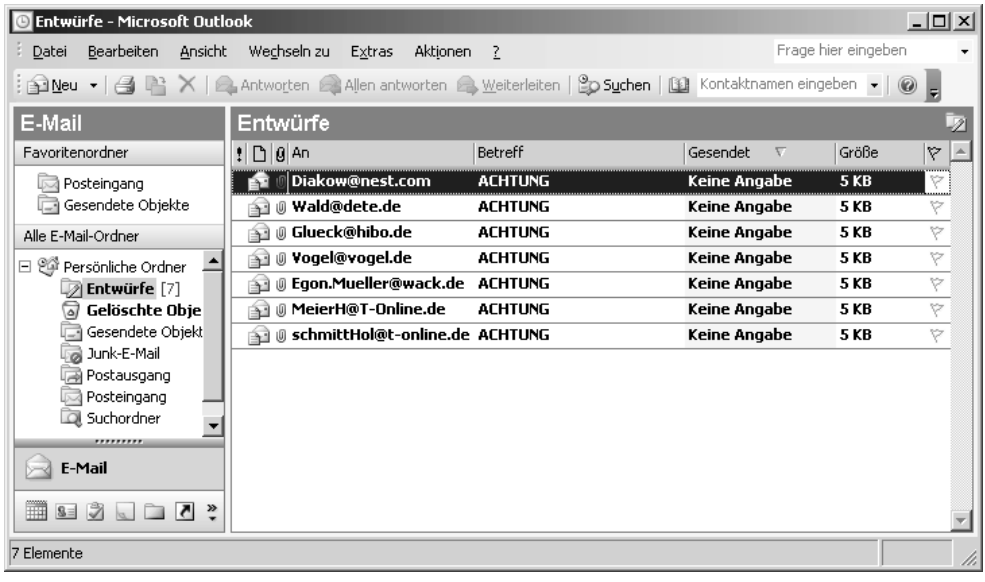


Abbildung 334: Mehrere Mails wurden in den Ordner Entwurf geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub AnhängeSpeichern()
   Dim OutMapi As New Outlook.Application
   Dim OutVerz As Object
   Dim OutMail As Object
   Dim intz As Integer
   Dim intAnhZ As Integer

   On Error GoTo fehler
   Set OutVerz = OutMapi.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox)

   For intz = 1 To OutVerz.Items.Count
      Set OutMail = OutVerz.Items(intz)

      If OutMail.Attachments.Count > 0 Then
         For intAnhZ = 1 To OutMail.Attachments.Count
            OutMail.Attachments.Item(intAnhZ).SaveAsFile _
               "C:\Test\" & OutMail.Attachments.Item(intAnhZ)
         Next intAnhZ
      End If
      Set OutMail = Nothing
   Next intz
```

Listing 531: E-Mail-Anhänge werden separat gespeichert.

```
    Set OutMapi = Nothing
    Set OutVerz = Nothing
    MsgBox "Verarbeitung beendet!"
    Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 531: E-Mail-Anhänge werden separat gespeichert. (Forts.)

Im Makro aus Listing 531 werden alle E-Mails des aktiven Mailordners gezählt, das heißt, man muss vor dem Start des Makros den gewünschten Ordner in Outlook aktivieren. Das Zählen der E-Mails erfolgt über die Funktion `Count`. Danach werden alle E-Mails des vorher eingestellten Ordners in Outlook über eine Schleife abgearbeitet. Für den Fall, dass eine E-Mail einen Anhang hat, meldet die Anweisung `OutMail.Attachments.Count` einen Wert größer Null. In diesem Fall wird der Anhang über die Methode `SaveAsFile` im angegebenen Verzeichnis gespeichert. Am Ende des Makros werden die Objektverweise über das Schlüsselwort `Nothing` aufgehoben, um den Speicher wieder freizugeben.

443 Eine Diashow im Internet Explorer programmieren

Auch der Internet Explorer kann von Access aus gesteuert werden. Um die Objekte, Methoden und Eigenschaften des Internet Explorers vorab einmal anzusehen, binden Sie in der Entwicklungsumgebung die Bibliothek MICROSOFT INTERNET CONTROLS ein. Rufen Sie danach die Objektbibliothek über die Taste F2 auf und wählen Sie aus dem Kombinationsfeld den Eintrag SHDOCVW aus

Im folgenden Beispiel aus Listing 532 wird ein Verzeichnis nach Grafikdateien durchsucht. Diese werden im Abstand von ein paar Sekunden im Internet Explorer ähnlich einer Dia-Show angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlIE
'=====================================================

Declare Sub Sleep Lib "Kernel32.dll" (ByVal SchlafZeit As Long)

Sub InternetExplorer()
   Dim IE As New InternetExplorer
   Dim varDat as Variant

   IE.Visible = True
```

Listing 532: Eine Dia-Show im Internet Explorer durchführen

```
    With Application.FileSearch
       .NewSearch
       .LookIn = "C:\Eigene Dateien\"
       .FileName = "*.jpg"
       .SearchSubFolders = False
       If .Execute() > 0 Then
          For Each varDat In .FoundFiles
             IE.Navigate varDat
             Sleep 2000
          Next varDat
       End If
    End With
    IE.Quit
    Set IE = Nothing
 End Sub
```

Listing 532: Eine Dia-Show im Internet Explorer durchführen (Forts.)

Erstellen Sie zunächst ein neues Internet-Explorer-Objekt über die Deklaration `Dim IE as New InternetExplorer`. Danach haben Sie automatisch Zugriff auf alle Methoden und Eigenschaften, die dieses Objekt standardmäßig anbietet. Danach weisen Sie der Eigenschaft `Visible` den Wert `True` zu, um den Internet Explorer anzuzeigen. Im nächsten Schritt führen Sie eine Suche über das Objekt `FileSearch` durch und ermitteln dabei alle Grafikdateien eines Verzeichnisses, welches Sie über die Eigenschaft `LookIn` angeben. Damit ausschließlich Grafikdateien vom Typ JPG gesucht werden, übergeben Sie diese Information der Eigenschaft `FileName`. Mithilfe der Eigenschaft `SearchSubFolders` können Sie bestimmen, ob auch noch weitere Verzeichnisse, die in der Hierarchie unterhalb des bei der Eigenschaft `LookIn` angegebenen Verzeichnisses liegen, durchsucht werden sollen. Im Beispiel aus Listing 532 wird dies jedoch nicht zugelassen.

Über die Methode `Execute` wird die eigentliche Suche gestartet. Dabei wird die Verarbeitung im Makro nur fortgesetzt, wenn überhaupt etwas gefunden wird. Nach einer erfolgreichen Suche stehen die Namen aller gefundenen Dateien im Objekt `FoundFiles`, das Sie nun in einer `For each Next`-Schleife auslesen können. Innerhalb der Schleife wenden Sie die Methode `Navigate` an, um das jeweilige Bild in den Internet Explorer zu laden. Bevor jetzt das nächste Bild im Internet Explorer angezeigt wird, erfolgt eine Pause von 2000 Millisekunden. Diese Pause wird über die API-Funktion `Sleep` realisiert.

444 URLs auslesen

Im folgenden Beispiel wird der Text einer Internetseite ausgelesen. Dazu erfassen Sie die Funktion aus Listing 533.

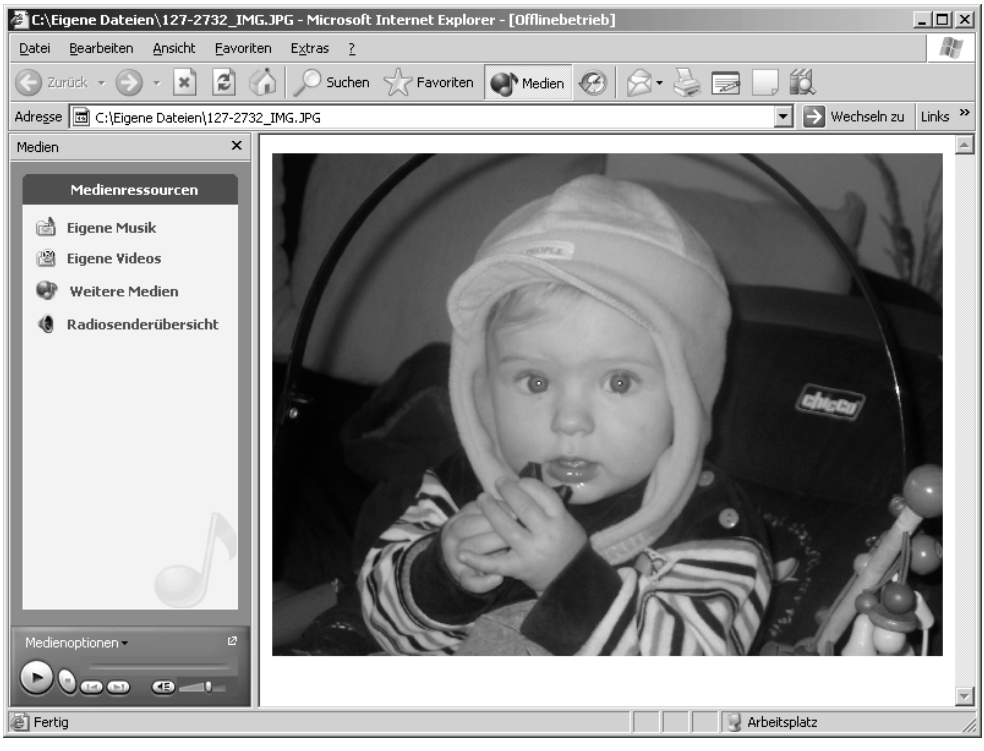


Abbildung 335: Die Diashow über den Internet Explorer

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlIE
'=====================================================

Function URLText(sURL As String) As String
   Dim IE As Object

   Set IE = CreateObject("InternetExplorer.Application")
   IE.Navigate sURL
   URLText = IE.Document.Body.InnerText

   IE.Quit
   Set IE = Nothing
End Function
```

Listing 533: Internetseitentext auslesen

Durch die Anweisung `CreateObject` wird eine neue Internet-Explorer-Sitzung erstellt. Über die Methode `Navigate` laden Sie die angegebene Seite. Mithilfe der Eigenschaft `Innertext`, mit der Sie auf das Objekt `Body` im aktuell geladenen Dokument zugreifen, können Sie den Text

der Internetseite abfragen. Die Anwendung wird am Ende über die Methode `Quit` beendet. Danach wird der Objektverweis durch die Anweisung `Set IE = Nothing` aufgehoben.

```
Sub OeffnenIE()
   Dim strSeite As String

   strSeite = URLText(Application.CurrentProject.Path & "\Artikel.htm")
   MsgBox strSeite
End Sub
```

Listing 534: URL-Text einer Seite auslesen

445 Ist Word bereits gestartet?

Möchten Sie Daten von Access nach Word übertragen, können Sie im ersten Schritt die Methoden `CreateObject` und `GetObject` einsetzen. So rufen Sie Ihre Textverarbeitung Word auf. Dabei prüfen Sie mithilfe der Funktion `GetObject`, ob Word bereits gestartet ist (für Word 2002 müssen Sie die Versionsnummer 10, für Word 2000 die Versionsnummer 9 angeben). Wenn nicht, bekommen Sie die Fehlernummer 429 zurück, die besagt, dass die Objektkomponente nicht verfügbar ist. In diesem Fall erstellen Sie über die Funktion `CreateObject` einen Verweis auf Word.

Als erstes Beispiel für das Zusammenspiel von Access und Word schreiben Sie die Funktion Listing 535, die überprüft, ob Word bereits gestartet ist. Dabei muss die Word-Bibliothek unter EXTRAS/VERWEISE als Voraussetzung eingebunden sein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================

Function WordOffen() As Boolean
   Dim WordApp As Word.Application

   On Error Resume Next
   Set WordApp = GetObject(, "Word.Application")

   If Err.Number = 0 Then
      WordOffen = True
   Else
      WordOffen = False
   End If
End Function
```

Listing 535: Funktion, die prüft, ob Word gestartet ist

Verwenden Sie die `GetObject`-Funktion für den Zugriff auf die Objektbibliothek von Word, und weisen Sie das Objekt einer Objektvariablen zu. Verwenden Sie die `Set`-Anweisung, um

das von `GetObject` zurückgegebene Objekt der Objektvariablen zuzuweisen. Ist dieser Vorgang erfolgreich, dann meldet das Objekt `Err` den Wert `0`. In diesem Fall ist Word bereits gestartet. Als Rückgabewert geben Sie daher den Wahrheitswert `True` an das aufrufende Makro aus Listing 536 zurück.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=================================================

Sub StartWord()
   Dim WordApp As Object

   If WordOffen = False Then
      Set WordApp = CreateObject("Word.Application")
      WordApp.Documents.Add
      WordApp.Visible = True
   End If
End Sub
```

Listing 536: Word wird gestartet.

Im Makro aus Listing 536 wurde das Late-Binding durchgeführt und die Objektvariable `WordApp` deklariert. Meldet die Funktion `WordOffen` den Rückgabewert `False`, dann ist Word noch nicht geöffnet. In diesem Fall wenden Sie die Funktion `CreateObject` an, um ein Word-Objekt zu erstellen. Danach haben Sie Zugriff auf alle Methoden und Eigenschaften des Objekts. Verwenden Sie im Anschluss daran die Methode `Add` des Objekts `Documents`, um ein neues Dokument einzufügen. Wenden Sie danach die Eigenschaft `Visible` an, um die Applikation anzuzeigen.

446 Access-Tabelle übertragen

Im nächsten Beispiel soll die Access-Tabelle `Artikel` in ein neues Word-Dokument eingefügt werden. Dabei soll in Word eine neue leere Tabelle eingefügt werden, die danach mit den einzelnen Datenfeldern aus der Datenbanktabelle gefüllt wird. Das etwas längere Makro für diese Aufgabe sehen Sie in Listing 537.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=================================================
```

Listing 537: Eine Access-Tabelle in ein Word-Dokument übertragen – Variante 1

```
Sub AccessTabelleNachWord()
   Dim WordObj As Object
   Dim WordDoc As Object
   Dim intPos As Object
   Dim WordTab As Object
   Dim dbs As New ADODB.Recordset
   Dim intz As Integer

   On Error GoTo fehler
   dbs.Open "Artikel", CurrentProject.Connection

   Set WordObj = GetObject(, "Word.Application.11")
   If Err.Number = 429 Then
      Set WordObj = CreateObject("Word.Application.11")
      Err.Number = 0
   End If

   WordObj.Visible = True

   Set WordDoc = WordObj.Documents.Add

   With WordObj.Selection
      .TypeText Text:="Artikelliste aus : " & CurrentProject.Name
      .TypeParagraph
      .TypeText Text:="vom  " & Format(Now(), "dd-mmm-yyyy")
      .TypeParagraph
   End With

   Set intPos = WordDoc.Range(Start:=0, End:=0)
   Set WordTab = WordDoc.Tables.Add(Range:=intPos, NumRows:=80, NumColumns:=6)

   With WordTab
      .Cell(1, 1).Range.Text = "Artikelname"
      .Cell(1, 2).Range.Text = "Liefereinheit"
      .Cell(1, 3).Range.Text = "Einzelpreis"
      .Cell(1, 4).Range.Text = "Mindestbestand"
      .Cell(1, 5).Range.Text = "Lagerbestand"
      .Cell(1, 6).Range.Text = "Bestellte Einheiten"
   End With
   intz = 2

   Do Until dbs.EOF
      With WordTab
         .Cell(intz, 1).Range.Text = dbs!Artikelname
         .Cell(intz, 2).Range.Text = dbs!Liefereinheit
         .Cell(intz, 3).Range.Text = dbs!Einzelpreis
         .Cell(intz, 4).Range.Text = dbs!Mindestbestand
         .Cell(intz, 5).Range.Text = dbs!Lagerbestand
         .Cell(intz, 6).Range.Text = dbs!BestellteEinheiten
         intz = intz + 1
         dbs.MoveNext
```

Listing 537: Eine Access-Tabelle in ein Word-Dokument übertragen – Variante 1 (Forts.)

```
      End With
   Loop

   WordDoc.SaveAs "c:\Eigene Dateien\Artikel.doc"
   WordDoc.Close

   Set WordObj = Nothing
   Set WordDoc = Nothing
   Set WordTab = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 537: Eine Access-Tabelle in ein Word-Dokument übertragen – Variante 1 (Forts.)

Legen Sie im ersten Schritt die benötigten Objektvariablen an. Unter anderem brauchen Sie ein Objekt, um die Textverarbeitung Word zu verwalten, und eines, um das neue Dokument darin ansprechen zu können. Für Ihre Access-Datentabelle benötigen Sie ein `Recordset`-Objekt, über das Sie alle Sätze in der Tabelle `Artikel` programmiertechnisch bearbeiten können. Nachdem Sie Ihre Tabelle sowie die Textverarbeitung Word mithilfe der Methode `Open` bzw. `CreateObject` gestartet haben, machen Sie die Word-Anwendung sichtbar. Dazu setzen Sie die Eigenschaft `Visible` auf den Wert `True`.

Mithilfe der Methode `Add` fügen Sie jetzt ein neues, noch leeres Dokument ein. Damit Sie später dieses neue Dokument weiter verarbeiten können, speichern Sie es in der Objektvariablen `WordDoc`.

Sie haben jetzt eine Mischung aus Access- und Word-VBA-Befehlen. Unterscheiden können Sie diese, indem Sie immer das anführende Objekt ansehen. `WordObj`, `WordDoc` und `WordTab` beinhalten die Word-VBA-Befehle, das Objekt `dbs` beinhaltet alle Access-VBA-Befehle.

Mithilfe der Word-Eigenschaft `Selection` können Sie einen markierten Bereich oder eine Einfügestelle im Dokument angeben. Über die Methode `TypeText` können Sie an der Einfügestelle einen beliebigen Text einfügen. Das Access-Objekt `CurrentProject` in Verbindung mit der Eigenschaft `Name` gibt Auskunft darüber, woher die Daten stammen, die übertragen werden sollen. Nach dem Einfügen des Textes fügen Sie einen leeren Absatz ein und geben danach das formatierte Tagesdatum aus.

Fügen Sie jetzt eine neue Tabelle in Ihr Dokument ein. Dazu müssen Sie vorher die aktuelle Position im Dokument bestimmen, wo die neue Tabelle eingefügt werden soll. Diese Information speichern Sie in der Objektvariablen `IntPos`. Mithilfe der Methode `Add`, die Sie auf das Objekt `Tables` anwenden, fügen Sie Ihre neue Tabelle in das Dokument ein.

Die Methode `Add` zum Einfügen einer Tabelle hat folgende Syntax:

```
Add(Range, NumRows, NumColumns DefaultTableBehavior, AutoFitBehavior)
```

Unter dem Argument `Range` geben Sie die genaue Einfügeposition der Tabelle bekannt.

Im Argument NumRows legen Sie die Anzahl der Zeilen fest, die in der Tabelle enthalten sein sollen.

Das Argument NumColumns bestimmt die Anzahl der Spalten, die in der Tabelle enthalten sein sollen.

Mithilfe des Arguments DefaultTableBehavior können Sie entscheiden, ob sich die Zellengröße in der Tabelle automatisch ändert, wenn zu viele Zeichen in eine Zelle übertragen werden. Standardmäßig ist diese Einstellung über die Konstante wdWord9TableBehavior aktiviert und kann daher auch weggelassen werden. Möchten Sie, dass sich die Zellgröße nicht ändert, dann verwenden Sie die Konstante wdWord8TableBehaviour.

Im letzten Argument AutoFitBehavior legen Sie die Regeln für das AUTOANPASSEN der Zellen fest, nach denen die Größe von Tabellen in Word geändert wird. Dies kann eine der folgenden WdAutoFitBehavior-Konstanten sein: wdAutoFitContent, wdAutoFitFixed oder wdAutoFitWindow. Wenn DefaultTableBehavior auf wdWord8TableBehavior gesetzt ist, wird dieses Argument ignoriert.

Um die Tabelle elegant ansprechen zu können, bilden Sie die Objektvariable WordTab und speichern in ihr die erste Tabelle in Ihrem Dokument.

In einer anschließenden Schleife wird die Access-Tabelle Artikel durchlaufen und die einzelnen Feldinhalte werden in die dafür vorgesehenen Zellen der Word-Tabelle übertragen. Setzen Sie für diese Aufgabe das Word-Objekt Cell ein. Dieses Objekt erwartet einen Wert, der die jeweilige Zeile identifiziert, sowie einen Wert für die Spalte. Mit dem Objekt Range ist die Zellenfläche der einzelnen Tabellenzelle gemeint, welche Sie über die Eigenschaft Text füllen können. Als Text setzen Sie dabei natürlich die einzelnen Datenfelder Ihrer Access-Tabelle ein. Addieren Sie bei jedem eingelesenen Satz die Zählvariable intz, die für die Zeile steht, um den Wert 1 zu erhöhen.

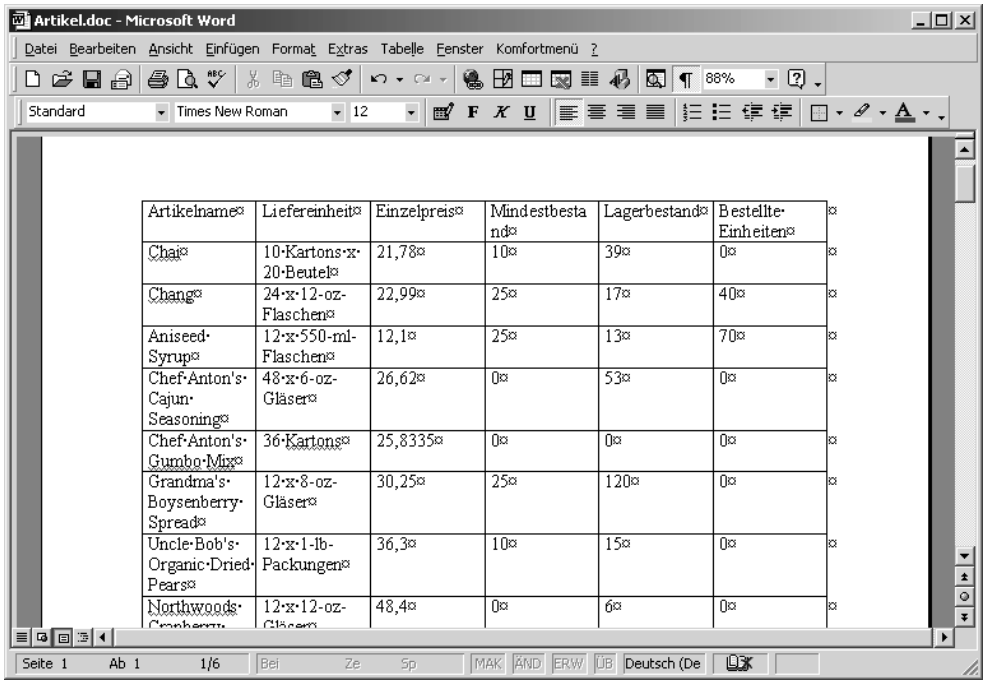
Vergessen Sie nicht, am Ende der Schleife den Zeiger über die Methode MoveNext auf den nächsten Datensatz in der Artikeltabelle zu setzen, da Sie sonst eine Endlosschleife produzieren.

Über die Methode SaveAs speichern Sie das Dokument im angegebenen Verzeichnis. Die Methode Close schließt im Anschluss daran das Dokument.

447 Access-Tabellen transferieren

Eine weitere sehr komfortable Art und Weise, eine Tabelle in ein Word-Dokument zu transferieren, lässt sich über die Methode OutputTo realisieren. Bei dieser Methode wird Ihnen der komplette Datentransfer weitestgehend abgenommen. Dabei wird die komplette Tabelle übertragen.

Im folgenden Beispiel aus Listing 538 wird die Tabelle ARTIKEL als Word-Dokument ausgegeben.



Artikelname	Liefereinheit	Einzelpreis	Mindestbestand	Lagerbestand	Bestellte Einheiten
Chai	10 Kartons x 20 Beutel	21,78	10	39	0
Chang	24 x 12 oz Flaschen	22,99	25	17	40
Aniseed Syrup	12 x 550 ml Flaschen	12,1	25	13	70
Chef Anton's Cajun Seasoning	48 x 6 oz Gläser	26,62	0	53	0
Chef Anton's Gumbo Mix	36 Kartons	25,8335	0	0	0
Grandma's Boysenberry Spread	12 x 8 oz Gläser	30,25	25	120	0
Uncle Bob's Organic Dried Pears	12 x 1 lb Packungen	36,3	10	15	0
Northwoods	12 x 12 oz Gläser	48,4	0	6	0

Abbildung 336: Die Tabelle wurde nach Word übertragen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================

Sub TabelleNachWordTranferieren()
   On Error GoTo fehler
   DoCmd.OutputTo acOutputTable, "Artikel", acFormatRTF, _
      "C:\Eigene Dateien\Artikel2.doc", True
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 538: Eine Access-Tabelle in ein Word-Dokument übertragen – Variante 2

Mit der Methode `OutputTo` können Sie die Daten in einem bestimmten Microsoft Access-Datenbankobjekt (einem Datenblatt, einem Formular, einem Bericht, einem Modul oder einer Datenzugriffsseite) in verschiedenen Formaten ausgeben. Dabei lautet die Syntax dieser Methode wie folgt:

```
OutputTo(ObjectType, ObjectName, OutputFormat, OutputFile, AutoStart, TemplateFile)
```

Über das Argument `ObjectType` legen Sie die Art des Access-Objekts fest, dessen Daten Sie exportieren möchten.

Dabei haben Sie folgende Möglichkeiten.

- `acOutputForm`: Export der Daten eines Formulars
- `acOutputFunction`: Export einer Funktion zur Sicherung
- `acOutputModule`: Export eines kompletten Moduls inklusive aller Funktionen und Makros
- `acOutputQuery`: Export der Ergebnisse einer Abfrage
- `acOutputReport`: Export eines Berichts
- `acOutputServerView`: Export einer Serveransicht
- `acOutputStoredProcedure`: Export einer gespeicherten Prozedur
- `acOutputTable`: Export einer Tabelle

Beim Argument `ObjectName` geben Sie den Namen des Objekts an, das Sie exportieren möchten. Im Makro aus Listing 538 ist dies der Name des Objekts `acOutputTable`.

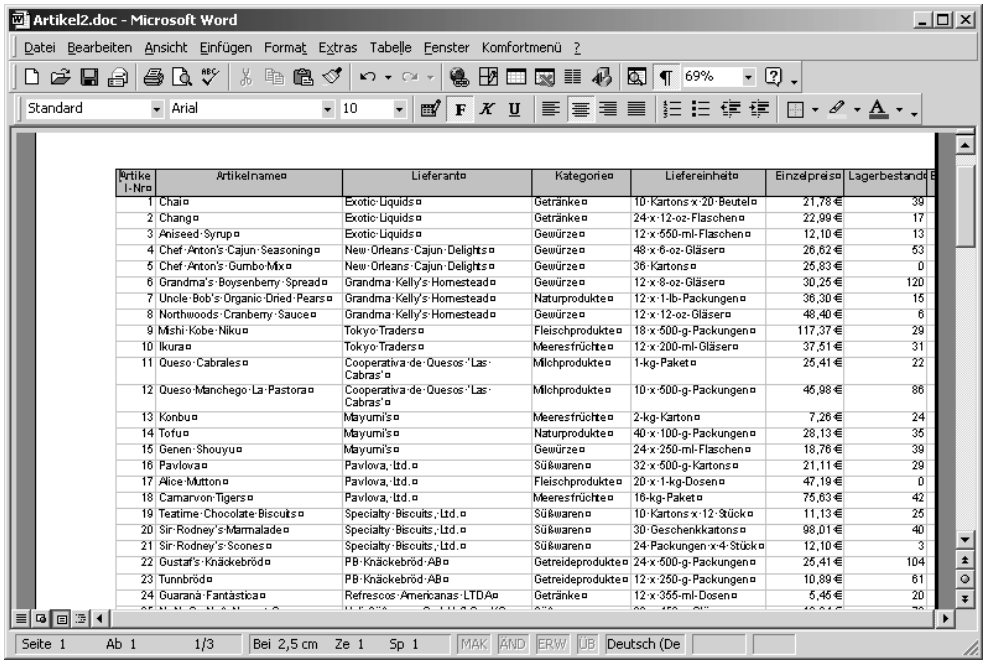
Das Argument `OutPutFormat` legt fest, in welchem Datenformat Sie die Daten transferieren möchten. Die bekanntesten Formate heißen dabei wie folgt:

- `acFormatHTML`: Konvertiert die Daten in das HTML-Format.
- `acFormatRTF`: Konvertiert die Daten in das Rich-Text-Format. Dieses Format kann beispielsweise problemlos in Microsoft Word eingelesen werden.
- `acFormatTXT`: Mit diesem Format ist das Textformat gemeint.
- `acFormatXLS`: Konvertiert die Daten in das Microsoft Excel-Format.

Beim Argument `OutputFile` geben Sie den Pfad sowie den Dateinamen der Datei an, in welche Sie die Daten transferieren möchten. Dabei muss die Datei noch nicht vorhanden sein. Access legt diese bei Bedarf selbst an.

Mithilfe des Arguments `AutoStart` haben Sie die Möglichkeit, die so erstellte Exportdatei gleich zu öffnen. Verwenden Sie den Wert `True`, um die entsprechende auf Windows basierende Anwendung sofort zu starten. Setzen Sie das Argument auf den Wert `False` oder lassen Sie es weg, wenn Sie die Exportdatei nicht öffnen möchten.

Das Argument `TemplateFile` ist dann von Interesse, wenn Sie eine Vorlage beispielsweise für die HTML-Datei verwenden möchten. In diesem Fall ist der komplette Pfad dieser Vorlagendatei anzugeben.



Artikel-Nr	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbestand
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €	39
2	Chang	Exotic Liquids	Getränke	24 x 12 oz Flaschen	22,99 €	17
3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550 ml Flaschen	12,10 €	13
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6 oz Gläser	26,62 €	53
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €	0
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8 oz Gläser	30,25 €	120
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1 lb Packungen	36,30 €	15
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12 oz Gläser	48,40 €	6
9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	18 x 500 g Packungen	117,37 €	29
10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200 ml Gläser	37,51 €	31
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	1 kg Paket	25,41 €	22
12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	10 x 500 g Packungen	46,98 €	86
13	Konbu	Mayumi's	Meeresfrüchte	2 kg Karton	7,26 €	24
14	Tofu	Mayumi's	Naturprodukte	40 x 100 g Packungen	28,13 €	35
15	Genen Shouyu	Mayumi's	Gewürze	24 x 250 ml Flaschen	18,76 €	39
16	Pavlova	Pavlova, Ltd.	Süßwaren	32 x 500 g Kartons	21,11 €	29
17	Alice Mutton	Pavlova, Ltd.	Fleischprodukte	20 x 1 kg Dosen	47,19 €	0
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte	16 kg Paket	75,63 €	42
19	Teatime Chocolate Biscuits	Specialty Biscuits, Ltd.	Süßwaren	10 Kartons x 12 Stück	11,13 €	25
20	Sir Rodney's Marmalade	Specialty Biscuits, Ltd.	Süßwaren	30 Geschenkkartons	98,01 €	40
21	Sir Rodney's Scones	Specialty Biscuits, Ltd.	Süßwaren	24 Packungen x 4 Stück	12,10 €	3
22	Gustaf's Knäckebröd	PB Knäckebröd AB	Getreideprodukte	24 x 500 g Packungen	25,41 €	104
23	Tunnbröd	PB Knäckebröd AB	Getreideprodukte	12 x 250 g Packungen	10,89 €	61
24	Guaraná Fantástica	Refrescos Americanas LTDA	Getränke	12 x 355 ml Dosen	5,46 €	20

Abbildung 337: Die Ausgabe einer Tabelle über OutputTo

448 Word-Makro starten

Über den Einsatz der Methode `Run` können Sie Makros starten. Wenn sich beispielsweise das Makro `Konvertierung` in einem Word-Dokument befindet, das Sie von Access aus starten möchten, dann wenden Sie das Makro aus Listing 539 an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=======================================================

Sub WordMakroStarten()
   Dim DOK As Object
   Set DOK = CreateObject("Word.Application")

   DOK.Documents.Open "C:\Eigene Dateien\Dok1.Doc"
   DOK.Run "Modul1.Konvertierung"
   DOK.Close
   DOK.Quit

   Set DOK = Nothing
End Sub
```

Listing 539: Ein Word-Makro von Access aus starten

Setzen Sie die Methode `Open` ein, um das Word-Dokument zu öffnen, das das zu startende Makro enthält. Führen Sie danach die Methode `Run` aus, der Sie den Namen des Makros bekannt geben. Es kann dabei auch nicht schaden, genau anzugeben, wo das Makro sich in der Entwicklungsumgebung befindet. Schließen Sie nach dem Ablauf des Makros das Dokument über die Methode `Close` und beenden Sie die Word-Anwendung mithilfe der Methode `Quit`. Heben Sie danach den Objektverweis `DOK` wieder auf, um den reservierten Speicher wieder freizugeben.

449 Datenbankzugriff von Word durchführen

Der Datentransfer von Access nach Word verläuft in den meisten Fällen genau umgekehrt. Stellen Sie sich dazu einmal vor, Sie haben ein Word-Dokument und möchten jetzt beispielsweise Kontaktdaten aus einer Access-Tabelle abrufen. Diese Daten werden dann an bestimmten Textmarken in Ihrem Dokument eingefügt. Um eine solche Lösung umzusetzen, müssen Sie in Word anfangen zu programmieren und dann mithilfe von ADO auf eine Kontaktdatenbank zugreifen. Dabei kann diese Aktion völlig im Hintergrund ablaufen, ohne dass der Anwender etwas davon merkt. Sie als Administrator haben dann die Möglichkeit, eine einzige Kontaktdatenbank anzulegen, auf die alle Ihre Anwender zugreifen. Es sollen in Zukunft keine Insellösungen mehr existieren, sondern alles soll über den Access-Datenbestand abgewickelt werden.

Im folgenden Beispiel wird von der Tabelle `Kontakte` ausgegangen, die Sie in Abbildung 339 vorab mal ansehen können.

Kontakte : Tabelle

	ID	Nachname	Vorname	Straße	PLZ	Ort	Mail
	1	Schmitt	Holger	Tetanus-Str 5	70469	Stuttgart	schmittHol@t-online
	2	Meier	Hans	Burgstr. 5	70469	Stuttgart	MeierH@T-Online.de
	3	Müller	Egon	Seestr. 42	70771	Leinfelden	Egon.Mueller@wack
▶	4	Vogel	Holger	Akazienweg 13	70871	Stuttgart	Vogel@vogel.de
	5	Glück	Maria	Hohenzollern 9	70839	Gerlingen	Glueck@hibo.de
	6	Wald	Hubert	Sattlerstr. 89	70868	Stuttgart	Wald@dete.de
	7	Diakow	Vera	Fichtenweg 13	70771	Leinfelden	Diakow@nest.com
*	Vert)						

Datensatz: 4 von 7

Abbildung 338: Die Ausgangstabelle mit den Kontaktdaten

Im nächsten Schritt legen Sie ein Word-Dokument mit dem Namen *Memo.doc* an und fügen einige Textmarken über den Menübefehl EINFÜGEN/TEXTMARKE ein.

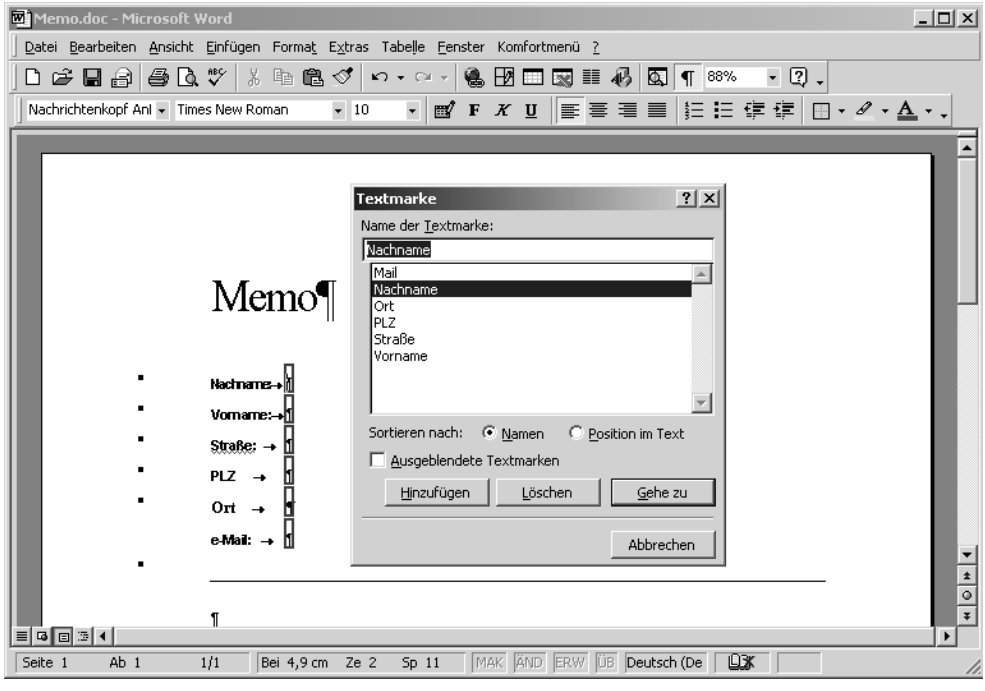


Abbildung 339: Die Vorbereitung im Dokument

Verwenden Sie bei den Textmarken die aus Abbildung 340 vorgeschlagenen Textmarken, damit der anschließende Code auch richtig ohne weitere Anpassung läuft.

Da das Kontaktdaten-Abfrage-Makro von Word aus gestartet werden soll, ist ein Dialog vorgesehen, aus dem der Anwender den Namen der Kontaktperson aus einem Kombinationsfeld auswählen kann. Dazu wechseln Sie über die Tastenkombination Alt + F11 in die Entwicklungsumgebung von Word und fügen über den Menübefehl EINFÜGEN/USERFORM einen neuen, noch leeren Dialog ein. Auf dieser Userform fügen Sie zwei Schaltflächen und ein Kombinationsfeld ein.

Beim Aufruf der Userform soll das Kombinationsfeld mit den Access-Daten `Nachname` und `Vorname` gefüllt werden. Dazu legen Sie das Ereignis aus Listing 540 genau hinter die Userform.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Memo.Doc
' Klasse     Userform1
'=================================================
```

Listing 540: Das Kombinationsfeld beim Aufruf der Userform füllen

```
Private Sub UserForm_Initialize()
   Dim CONN As ADODB.Connection
   Dim DBS As ADODB.Recordset

   Set CONN = New ADODB.Connection
   With CONN
      .Provider = "Microsoft.Jet.OLEDB.4.0"
      .Open ThisDocument.Path & "\Umfeld.mdb"
   End With

   Set DBS = New ADODB.Recordset
   With DBS
      .Open Source:="Kontakte", ActiveConnection:=CONN, _
          CursorType:=adOpenKeyset, LockType:=adLockOptimistic
      Do While Not .EOF
         UserForm1.ComboBox1.AddItem _
            .Fields("Nachname").Value & ", " & .Fields("Vorname").Value
         .MoveNext
      Loop
      .Close
   End With
   CONN.Close
   UserForm1.ComboBox1.ListIndex = 0

   Set DBS = Nothing
End Sub
```

Listing 540: Das Kombinationsfeld beim Aufruf der Userform füllen (Forts.)

Damit das Makro aus Listing 540 korrekt abläuft, binden Sie in der Entwicklungsumgebung von Word die Bibliothek MICROSOFT ACTIVEX DATA OBJECT 2.7 LIBRARY ein, indem Sie diese Bibliothek über den Menübefehl EXTRAS/VERWEISE aus dem Listenfeld VERFÜGBARE VERWEISE aktivieren.

Die Datenbank mit der Tabelle `Kontakte` befindet sich in diesem Beispiel im gleichen Verzeichnis wie das Word-Dokument *Memo.doc*. Daher können Sie dieses einheitliche Verzeichnis über die Eigenschaft `Path`, welche Sie auf das aktuelle Dokument anwenden, abfragen und anschließend mit der Methode `Open` die Datenbank öffnen. Ebenso über die Methode `Open` wird die Tabelle `Kontakte` geöffnet. Danach werden alle darin enthaltenen Datensätze verarbeitet und die Felder `Nachname` und `Vorname` über die Methode `AddItem` in das Kombinationsfeld eingelesen. Damit am Ende der erste Name im Listenfeld angezeigt wird, setzen Sie die Eigenschaft `ListIndex` des Kombinationsfelds auf den Wert `0`.

Das Makro für den Start der Userform entnehmen Sie dem Listing 541. Dieses Makro wird in einem Standardmodul erfasst.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Memo.Doc
' Modul      Modul1
'=================================================

Sub DatenAbfrage()
   UserForm1.Show
End Sub
```

Listing 541: Die Word-Userform starten

Über den Einsatz der Methode `Show` starten Sie die Userform. Kurz darauf wird das Ereignis `Initialize` ausgelöst, welches im Hintergrund auf die Access-Datentabelle KONTAKTE zugreift und das Kombinationsfeld füllt.

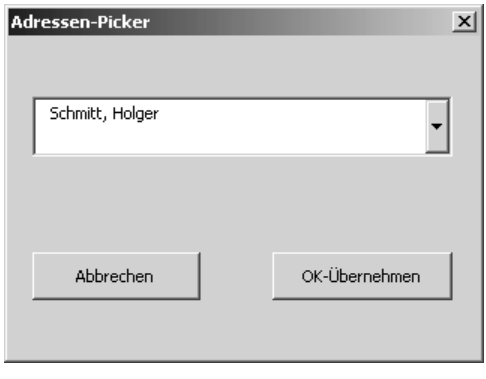


Abbildung 340: Alle Namen stehen im Kombinationsfeld zur Auswahl bereit.

Was jetzt noch fehlt, ist das Makro für die Übernahme der Kontaktdaten. Dabei wird das Makro aus Listing 542 hinter die Schaltfläche OK/ÜBERNEHMEN gelegt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Memo.Doc
' Klasse     Userform1
'=================================================

Private Sub cmd_OK_Click()
   Dim WordObj As Object
   Dim CONN As ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim strName As String
   Dim strNachname As String
```

Listing 542: Kontaktdaten in Access-Tabelle finden und ins Dokument übertragen

```
Dim strVorname As String
Dim strStraße As String
Dim strPLZ As String
Dim strOrt As String
Dim strMail As String

strName = UserForm1.ComboBox1.Value
strName = "Nachname='" & Left(strName, InStr(strName, ",") - 1) & "'"

Set CONN = New ADODB.Connection
With CONN
   .Provider = "Microsoft.Jet.OLEDB.4.0"
   .Open ThisDocument.Path & "\Umfeld.mdb"
End With

Set DBS = New ADODB.Recordset
With DBS
   .Open Source:="Kontakte", ActiveConnection:=CONN, _
      CursorType:=adOpenKeyset, LockType:=adLockOptimistic
   .Find Criteria:=strName, SearchDirection:=adSearchForward
   If Not .EOF Then
      strNachname = .Fields("Nachname").Value
      strVorname = .Fields("Vorname").Value
      strStraße = .Fields("Straße").Value
      strPLZ = .Fields("PLZ").Value
      strOrt = .Fields("Ort").Value
      strMail = .Fields("Mail").Value
   Else
      MsgBox "Datensatz nicht gefunden"
   End If
   .Close
End With

CONN.Close
Set DBS = Nothing

On Error Resume Next
Set WordObj = GetObject(, "Word.Application.11")
With WordObj
   .Selection.GoTo What:=wdGoToBookmark, Name:="Nachname"
   .Selection.TypeText Text:=strNachname
   .Selection.GoTo What:=wdGoToBookmark, Name:="Vorname"
   .Selection.TypeText Text:=strVorname
   .Selection.GoTo What:=wdGoToBookmark, Name:="Straße"
   .Selection.TypeText Text:=strStraße
   .Selection.GoTo What:=wdGoToBookmark, Name:="PLZ"
   .Selection.TypeText Text:=strPLZ
   .Selection.GoTo What:=wdGoToBookmark, Name:="ORT"
   .Selection.TypeText Text:=strOrt
   .Selection.GoTo What:=wdGoToBookmark, Name:="Mail"
   .Selection.TypeText Text:=strMail
```

Listing 542: Kontaktdaten in Access-Tabelle finden und ins Dokument übertragen (Forts.)

```
        '.ActiveDocument.Save
    End With

    Set WordObj = Nothing
    Unload me
End Sub
```

Listing 542: Kontaktdaten in Access-Tabelle finden und ins Dokument übertragen (Forts.)

Definieren Sie im ersten Schritt die Variablen, die Sie für diesen Job brauchen. Dazu gehören unter anderem die String-Variablen, um die Ergebnisse aus der Access-Tabelle zwischenzuspeichern. Des Weiteren benötigen Sie Objekte, um die Anwendungen Access und Word zu steuern und eine Objektvariable vom Typ `Recordset`, um die Access-Tabelle zu verarbeiten.

Öffnen Sie danach die beteiligte Datenbank sowie die Tabelle `Kontakte`. Setzen Sie die Methode `Find` ein und übergeben Sie ihr den Suchstring, den Sie sich in der Variablen `strName` zusammengebastelt haben. War die Suche über die Kundennummer erfolgreich, dann meldet die Eigenschaft `EOF` den Wert `False`. Diese Eigenschaft würde übrigens den Wert `True` melden, wenn die Suche erfolglos gewesen wäre. Dann wird nämlich der letzte Satz in der Tabelle erreicht. Weisen Sie nun den Variablen die gefundenen Werte zu und schließen Sie die Access-Tabelle gleich danach über die Methode `Close`. Im selben Atemzug schließen Sie auch die Datenbank.

Über die Funktion `GetObject` stellen Sie einen Verweis zum geöffnetem Dokument her und speichern diesen Verweis unter der Objektvariablen `WordObj`. Sie haben jetzt Zugriff auf alle Word-VBA-Befehle.

Setzen Sie die Methode `GoTo` ein, um zur angegebenen Textmarke zu springen. Dieser Befehl funktioniert genauso, als würden Sie die Taste F5 drücken, um das Dialogfeld SUCHEN UND ERSETZEN mit der Registerkarte GEHE ZU aufzurufen. Mithilfe der Methode `GoTo` können Sie neben Textmarken unter anderem auch Kommentare, Tabellen, Seiten und Zeilen anspringen. Fügen Sie mit der Methode `TypeText` den Inhalt der Variablen unmittelbar nach den Textmarken ins Dokument ein.

450 Excel bereits gestartet?

Im Zusammenspiel von Access und Excel können Sie sowohl Daten nach Excel exportieren als auch von Excel empfangen. Gerade wenn Sie eine Access-Tabelle haben und Ihr Kunde möglicherweise kein Access zur Verfügung hat, können Sie relativ leicht Ihre Access-Tabelle in eine Excel-Datei umwandeln.

Zu Beginn der Access/Excel-Beispiele prüfen Sie mithilfe der Funktion aus Listing 543, ob Excel bereits gestartet ist. Dabei muss die Excel-Bibliothek unter EXTRAS/VERWEISE als Voraussetzung eingebunden sein.

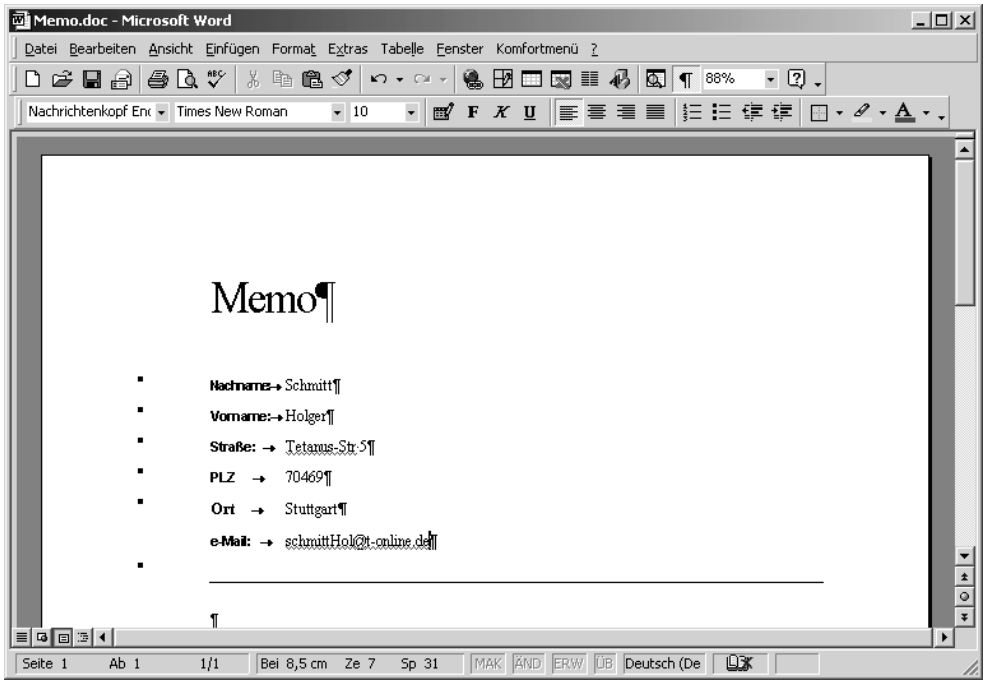


Abbildung 341: Die Daten wurden richtig übertragen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Function ExcelOffen() As Boolean
   Dim ExcelApp As Excel.Application

   On Error Resume Next
   Set ExcelApp = GetObject(, "Excel.Application")

   If Err.Number = 0 Then
      ExcelOffen = True
   Else
      ExcelOffen = False
   End If
End Function
```

Listing 543: Funktion zur Prüfung, ob Excel gestartet ist

Verwenden Sie die `GetObject`-Funktion für den Zugriff auf die Objektbibliothek von Excel und weisen Sie das Objekt einer Objektvariablen zu. Verwenden Sie die `Set`-Anweisung, um das von `GetObject` zurückgegebene Objekt der Objektvariablen zuzuweisen. Ist dieser Vorgang erfolgreich, dann meldet das Objekt `Err` den Wert `0`. In diesem Fall ist Excel bereits

gestartet. Als Rückgabewert geben Sie daher den Wahrheitswert `True` an das aufrufende Makro aus Listing 544 zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub StartExcel()
   Dim ExApp As Object

   If ExcelOffen = False Then
      Set ExApp = CreateObject("Excel.Application")
      ExApp.Workbooks.Add
      ExApp.Visible = True
   Else
      Set ExApp = GetObject(, "Excel.Application")
      ExApp.Visible = True
   End If
End Sub
```

Listing 544: Soll Excel gestartet werden oder nicht?

Im Makro aus Listing 544 wurde das Late-Binding durchgeführt und die Objektvariable `ExApp` deklariert. Meldet die Funktion `ExcelOffen` den Rückgabewert `False`, dann ist Excel noch nicht geöffnet. In diesem Fall wenden Sie die Funktion `CreateObject` an, um ein Excel-Objekt zu erstellen. Danach haben Sie Zugriff auf alle Methoden und Eigenschaften des Objekts. Geben Sie im Anschluss daran über die Methode `Add` des Objekts `Workbooks` an, dass Sie eine neue Arbeitsmappe einfügen wollen. Wenden Sie danach die Eigenschaft `Visible` an, um die Applikation anzuzeigen.

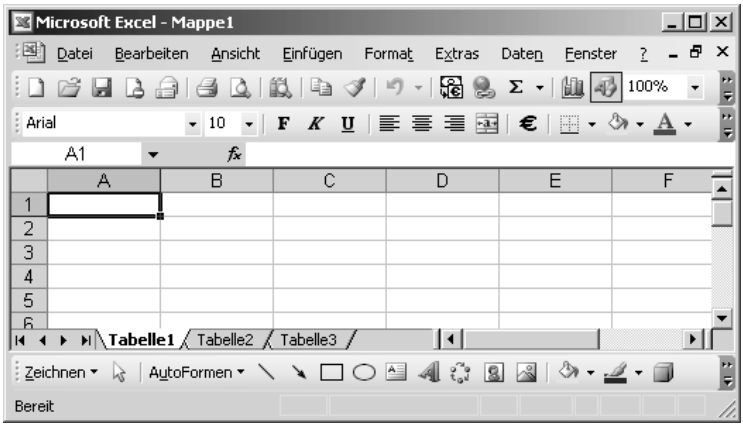


Abbildung 342: Excel wurde mit einer neuen Arbeitsmappe gestartet.

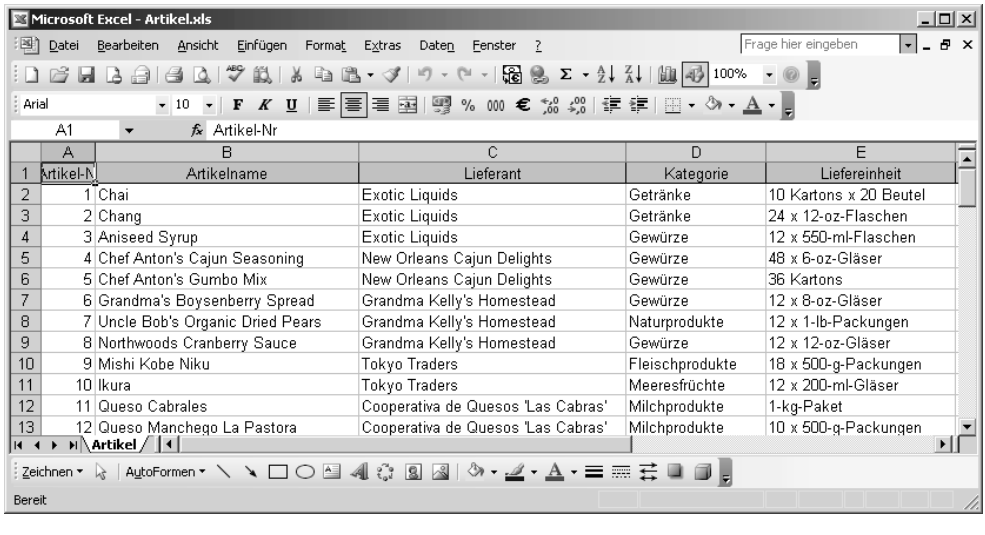
451 Access-Tabelle in eine Excel-Tabelle umwandeln

Im folgenden Beispiel aus Listing 545 wird die Access-Tabelle `Artikel` in eine Excel-Arbeitsmappe *Artikel.xls* umgewandelt. Alles was Sie dafür tun müssen, ist, die Methode `OutputTo` einzusetzen und das gewünschte Format festlegen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub TabelleNachExcelTranferieren()
   DoCmd.OutputTo acOutputTable, "Artikel", acFormatXLS, _
      "C:\Eigene Dateien\Artikel.xls", True
End Sub
```

Listing 545: Access-Tabelle nach Excel transferieren

Sollte die Exportdatei *Artikel.xls* noch nicht vorliegen, dann erstellt Access diese Datei selbstverständlich automatisch für Sie.



	A	B	C	D	E
1	Artikel-N	Artikelname	Lieferant	Kategorie	Liefereinheit
2	1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel
3	2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen
4	3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen
5	4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser
6	5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons
7	6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser
8	7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen
9	8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser
10	9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	18 x 500-g-Packungen
11	10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200-ml-Gläser
12	11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	1-kg-Paket
13	12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	10 x 500-g-Packungen

Abbildung 343: Der schnelle Export über die Methode OutputTo

Die Alternative

Eine weitere Möglichkeit, um Daten von Access nach Excel zu transportieren, bietet die Methode `TransferSpreadsheet`. Diese Methode hat folgende Syntax:

```
TransferSpreadsheet(Transfertyp, Dateiformat, Tabellenname, Dateiname,
BesitztFeldnamen, Bereich)
```

Im Argument `Transfertyp` geben Sie an, welchen Transfer Sie genau durchführen möchten. Sie haben dabei die Auswahl zwischen dem Export (`acExport`), dem Import (`acImport`) oder einer Verknüpfung (`acLink`).

Im Argument `Dateiformat` geben Sie an, in welcher Excel-Version (bzw. Lotus-Version) Sie die Access-Tabelle exportieren möchten.

Im darauf folgenden Argument `Tabellename` geben Sie den Namen der zu exportierenden Access-Tabelle an. Über das Argument `Dateinamen` geben Sie das Ziel für den Datenexport bekannt. Dabei muss die angegebene Datenquelle nicht einmal existieren. Access legt diese neu für Sie an.

Beim Argument `BesitztFeldnamen` verwenden Sie den Wert `True`, um die erste Zeile der Kalkulationstabelle beim Importieren, Exportieren oder Verknüpfen zur Angabe der Feldnamen zu verwenden. Geben Sie hingegen den Wert `False` an, wenn die erste Zeile als normale Datenzeile gelten soll. Wenn Sie dieses Argument nicht angeben, wird der Standardwert `False` verwendet.

Hinweis

Das Argument `Bereich` gilt nur für Importoperationen und darf beim Export nicht angegeben werden. Beim Import werden standardmäßig immer alle Datensätze importiert, sofern dieses Argument nicht gesetzt wird (siehe Abbildung 344).

452 Excel-Daten in eine Access-Tabelle transferieren

Mithilfe der gerade vorgestellten Lösung über die Methode `TransferSpreadsheet` können Sie auch Excel-Tabellen in eine Access-Datenbank importieren. Das Makro für diese Aufgabe lautet:

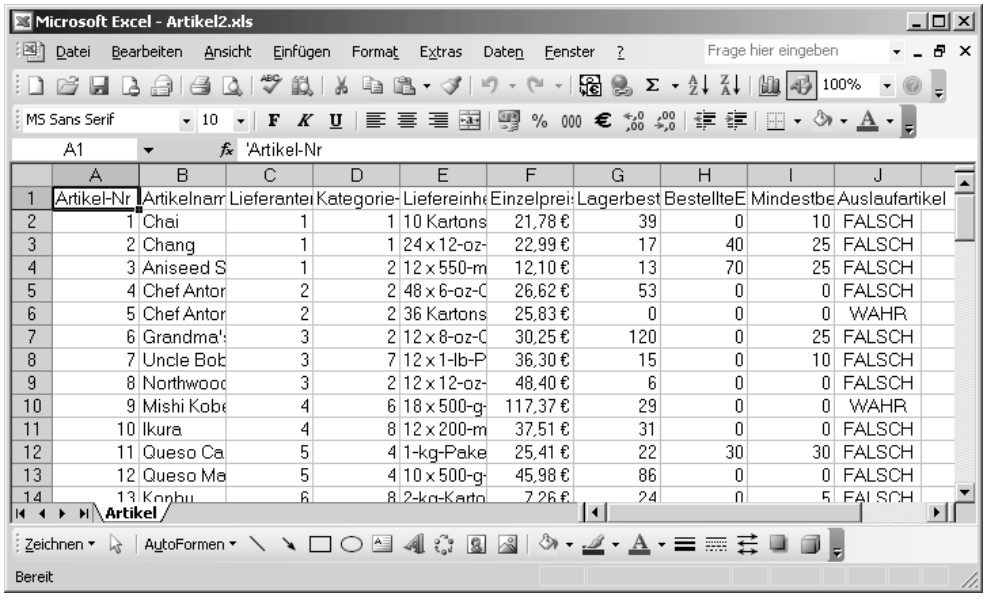
```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=================================================

Sub TabelleVonExcelImportieren()
   DoCmd.TransferSpreadsheet acImport, acSpreadsheetTypeExcel11, _
      "ArtikelAusExcel", "C:\Eigene Dateien\Artikel2.xls", True
End Sub
```

Listing 546: Eine Excel-Tabelle nach Access importieren

Übergeben Sie der Methode `TransferSpreadsheet` die Konstante `acImport`, um mitzuteilen, dass Access einen Import vornehmen soll. Als zweite Konstante geben Sie die Excel-Version an, in welcher die Tabelle vorliegt.

Dabei haben Sie folgende Möglichkeiten:

- `acSpreadsheetTypeExcel3` (Excel 3.0)
- `acSpreadsheetTypeExcel4` (Excel 4.0)
- `acSpreadsheetTypeExcel5` (Excel 5.0)
- `acSpreadsheetTypeExcel7` (Excel 95)
- `acSpreadsheetTypeExcel8` (Excel 97)
- `acSpreadsheetTypeExcel9` (Excel 2000)
- `acSpreadsheetTypeExcel10` (Excel 2002)
- `acSpreadsheetTypeExcel11` (Excel 2003)



	A	B	C	D	E	F	G	H	I	J
1	Artikel-Nr	Artikelnam	Lieferantei	Kategorie-	Liefereinhe	Einzelprei	Lagerbest	BestellteE	Mindestbe	Auslaufartikel
2	1	Chai	1	1	10 Kartons	21,78 €	39	0	10	FALSCH
3	2	Chang	1	1	24 x 12-oz-	22,99 €	17	40	25	FALSCH
4	3	Aniseed S	1	2	12 x 550-m	12,10 €	13	70	25	FALSCH
5	4	Chef Antor	2	2	48 x 6-oz-C	26,62 €	53	0	0	FALSCH
6	5	Chef Antor	2	2	36 Kartons	25,83 €	0	0	0	WAHR
7	6	Grandma's	3	2	12 x 8-oz-C	30,25 €	120	0	25	FALSCH
8	7	Uncle Bob	3	7	12 x 1-lb-P	36,30 €	15	0	10	FALSCH
9	8	Northwooc	3	2	12 x 12-oz-	48,40 €	6	0	0	FALSCH
10	9	Mishi Kobe	4	6	18 x 500-g-	117,37 €	29	0	0	WAHR
11	10	Ikura	4	8	12 x 200-m	37,51 €	31	0	0	FALSCH
12	11	Queso Ca	5	4	1-kg-Pake	25,41 €	22	30	30	FALSCH
13	12	Queso Ma	5	4	10 x 500-g-	45,98 €	86	0	0	FALSCH
14	13	Konbu	6	8	2-kg-Karto	7,26 €	24	0	5	FALSCH

Abbildung 344: Das Ergebnis der Methode TransferSpreadsheet

ArtikelAusExcel : Tabelle

Artikel-Nr	Artikelname	Lieferante	Kategori	Liefereinheit	Einzelpreis	Lagerbestand	BestellteEinheit
1	Chai	1	1	10 Kartons x 20 Beutel	21,78 €	39	0
2	Chang	1	1	24 x 12-oz-Flaschen	22,99 €	17	40
3	Aniseed Syrup	1	2	12 x 550-ml-Flaschen	12,10 €	13	70
4	Chef Anton's Ca	2	2	48 x 6-oz-Gläser	26,62 €	53	0
5	Chef Anton's Gu	2	2	36 Kartons	25,83 €	0	0
6	Grandma's Boy	3	2	12 x 8-oz-Gläser	30,25 €	120	0
7	Uncle Bob's Or	3	7	12 x 1-lb-Packungen	36,30 €	15	0
8	Northwoods Cra	3	2	12 x 12-oz-Gläser	48,40 €	6	0
9	Mishi Kobe Nik	4	6	18 x 500-g-Packungen	117,37 €	29	0
10	Ikura	4	8	12 x 200-ml-Gläser	37,51 €	31	0
11	Queso Cabrales	5	4	1-kg-Paket	25,41 €	22	30
12	Queso Manche	5	4	10 x 500-g-Packungen	45,98 €	86	0
13	Konbu	6	8	2-kg-Karton	7,26 €	24	0
14	Tofu	6	7	40 x 100-g-Packungen	28,13 €	35	0
15	Genen Shouyu	6	2	24 x 250-ml-Flaschen	18,76 €	39	0
16	Pavlova	7	3	32 x 500-g-Kartons	21,11 €	29	0
17	Alice Mutton	7	6	20 x 1-kg-Dosen	47,19 €	0	0
18	Carnarvon Tiger	7	8	16-kg-Paket	75,63 €	42	0

Datensatz: 1 von 77

Abbildung 345: Die Excel-Tabelle wurde erfolgreich in Access importiert.

453 Access als Datenlieferant für Excel

Im folgenden Beispiel erfolgen Zugriffe von Excel auf eine Access-Datenbank. Dabei werden Daten sowohl importiert als auch exportiert.

Um den Zugriff von Excel auf Access zu realisieren, brauchen Sie eine zusätzliche Objektbibliothek, die Sie in der Entwicklungsumgebung einbinden müssen. Dabei wechseln Sie in die Entwicklungsumgebung von Excel und wählen aus dem Menü EXTRAS den Befehl VERWEISE. Im Listenfeld VERFÜGBARE VERWEISE aktivieren Sie die Bibliothek MICROSOFT ACTIVEX DATA OBJECTS 2.7 und bestätigen mit OK. Damit haben Sie Zugriff auf die Datenzugriffsmethode ADO (ACTIVEX DATA OBJECT). Unter anderem zeichnet sich diese Methode durch eine sehr hohe Geschwindigkeit, eine benutzerfreundliche Bedienung sowie einen geringen Verbrauch an Arbeitsspeicher und Festplattenspeicher aus. Über den Einsatz von ADO können Sie auf Access-Tabellen zugreifen, um diese auszulesen bzw. neue Datensätze anzuhängen.

Zu Beginn wurde in Excel eine Userform für die Datensuche erstellt, die Sie in Abbildung 347 sehen können (siehe Abbildung 346).

Diese Datenmaske enthält vier Texteingabefelder, über die sich die diversen Suchen durchführen lassen. Möglich sind folgende Suchoptionen:

- Die Eingabe des Stern-Symbols im Feld `Name` listet im Listenfeld alle Datensätze auf.
- Suche nach dem Namen. Dabei wird lediglich das Feld `Name` ausgefüllt.
- Suche nach Name und Vorname. Beide Felder müssen hierzu ausgefüllt werden.
- Suche über Namen und Wohnort. Hier erfolgt eine Eingabe von Suchbegriffen in den Feldern `Name` und `Wohnort`.
- Suche nach Beruf. Hier darf nur im Feld `Beruf` ein Suchbegriff eingegeben werden.

- Suche nach Beruf und Wohnort. Suchen Sie beispielsweise einen Metzger in Hamburg, füllen Sie dazu beide Felder aus.
- Suche nach Wohnort: Hier werden alle Personen aus einer Stadt gesucht.

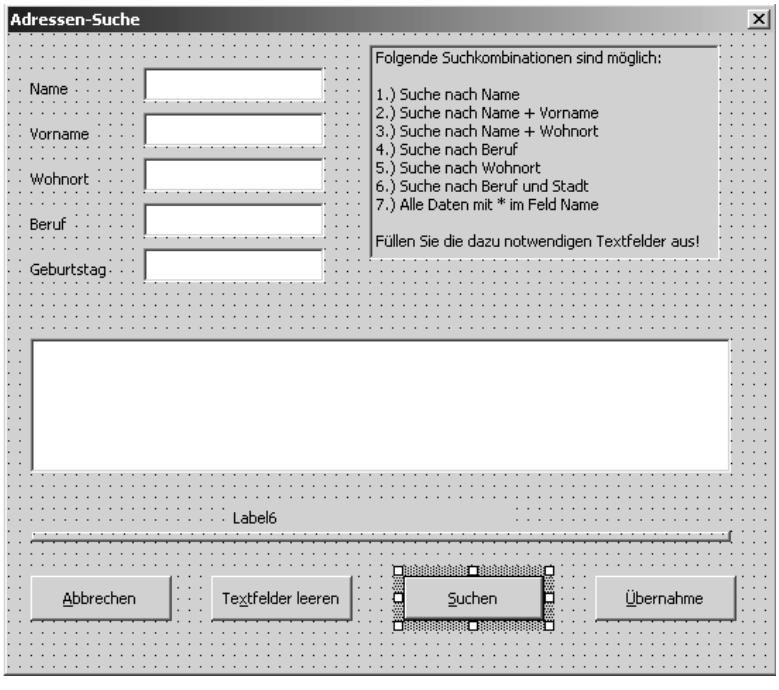


Abbildung 346: Die komfortable Adressensuche über Excel

Im Listenfeld der `Userform1` werden alle gefundenen Personen aufgelistet. Dabei wird das Listenfeld von Beginn an mehrspaltig definiert und die Spaltenbreiten werden definiert. Diese Aufgabe können Sie gleich beim Öffnen der Userform durchführen lassen. Dazu setzen Sie das Userform-Ereignis `Initialize` ein, wie Sie es in Listing 547 sehen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub UserForm_Initialize()
   'Userform initialisieren
   With UserForm1
      .ListBox1.ColumnCount = 5
      .ListBox1.ColumnWidths = "80;80;60;70;70"
      .CommandButton1.Default = True
      .TextBox1.SetFocus
```

Listing 547: Einige Aufgaben bereits beim Starten der Userform ausführen

```
        .Label6.Visible = False
        .Label6.Caption = ""
        .Label6.ForeColor = RGB(0, 0, 255)
        .Label6.Font.Bold = True
    End With
End Sub
```

Listing 547: Einige Aufgaben bereits beim Starten der Userform ausführen (Forts.)

Über die Eigenschaft `ColumnCount` geben Sie an, wie viele Spalten im Listenfeld angelegt werden sollen. Mithilfe der Eigenschaft `ColumnWidth` können Sie die Spaltenbreiten festlegen. Damit die Schaltfläche SUCHEN beim Drücken der Taste Enter automatisch gedrückt wird, setzen Sie die Eigenschaft `Default` dieser Schaltfläche auf den Wert `True`.

Standardmäßig setzen Sie den Mauszeiger gleich beim Starten der Userform in das erste Textfeld `Name`. Über ein Bezeichnungsfeld zeigen Sie später an, wie viele Datensätze gefunden werden konnten. Zu Beginn wird dieses Bezeichnungsfeld jedoch nicht angezeigt und somit der Eigenschaft `Visible` der Wert `False` zugewiesen. Die Beschriftung des Bezeichnungsfelds können Sie über die Eigenschaft `Caption` vornehmen. Die Farbe des Textes geben Sie über die Eigenschaft `ForeColor` und die Funktion `RGB` an. Die Funktion `RGB` mischt sich die Farbe aus den drei Komponenten Rot, Grün und Blau zusammen. Den Schriftschnitt FETT weisen Sie über die Eigenschaft `Bold` zu, die Sie auf `True` setzen.

Bevor Sie mit der eigentlichen Programmierung beginnen, werfen Sie einen Blick auf die Struktur der Tabelle `Adressen`.

Die Suchfunktion

Die diversen Suchfunktionen liegen hinter der Schaltfläche SUCHEN, die Sie übrigens auch über die Tastenkombination Alt + S direkt ausführen können. Hierfür ist die Eigenschaft `Accelerator` verantwortlich. Diese Eigenschaft können Sie direkt im Eigenschaften-Fenster der Entwicklungsumgebung einstellen, indem Sie die Schaltfläche markieren und dann im Eigenschaften-Fenster den Buchstaben eingeben, über den Sie in Verbindung mit der Taste Alt die Schaltfläche per Shortcut ausführen möchten. Danach wird der angegebene Buchstabe in der Schaltflächenbeschriftung automatisch unterstrichen.

Legen Sie jetzt den Code aus Listing 548 direkt hinter die Schaltfläche SUCHEN.

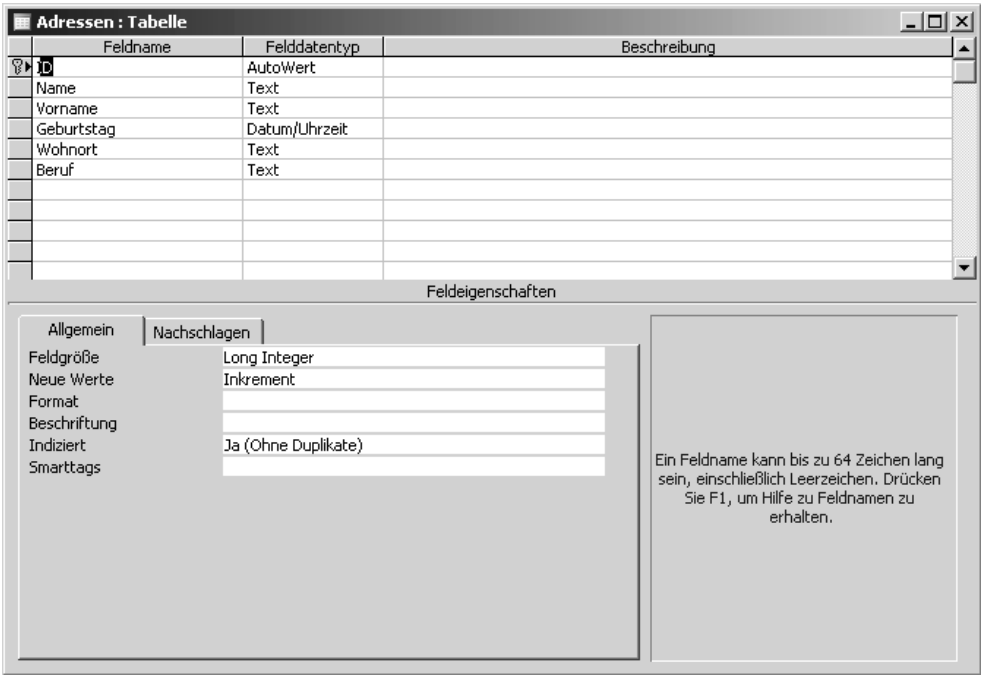


Abbildung 347: Die Datenstruktur der Access-Tabelle Adressen

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub CommandButton1_Click()
   'Daten in Access-Datenbank suchen und übergeben
   Dim AccApp As Object
   Dim conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim strBegriff As String
   Dim strBegriff2 As String
   Dim intZ As Integer
   Dim BSchalter As Boolean

   intZ = 0
   BSchalter = False

   With UserForm1
      'Listenfeld löschen
      .ListBox1.Clear

      'Alle Datensätze
```

Listing 548: Die Suche in der Access-Tabelle durchführen

```
If .TextBox1.Value = "*" Then
   strBegriff = ""
   strBegriff = "Name='" & strBegriff & "'"
   BSchalter = True
   GoTo weiter
End If

'Suche nach Beruf
If .TextBox4.Value <> "" And .TextBox1.Value = "" _
   And .TextBox2.Value = "" And .TextBox3.Value = "" Then
   strBegriff = UserForm1.TextBox4.Value
   strBegriff = "Beruf='" & strBegriff & "'"
End If

'Suche nach Beruf und Stadt
If .TextBox4.Value <> "" And .TextBox1.Value = "" _
   And .TextBox2.Value = "" And .TextBox3.Value <> "" Then
   strBegriff = UserForm1.TextBox4.Value
   strBegriff2 = UserForm1.TextBox3.Value
   strBegriff = "Beruf='" & strBegriff & "' and Wohnort='" & strBegriff2 & "'"
End If

'Suche nach Stadt
If .TextBox4.Value = "" And .TextBox1.Value = "" _
   And .TextBox2.Value = "" And .TextBox3.Value <> "" Then
   strBegriff = UserForm1.TextBox3.Value
   strBegriff = "Wohnort='" & strBegriff & "'"
End If

'Suche nach Name
If .TextBox1.Value <> "" And .TextBox2.Value = "" _
   And .TextBox3.Value = "" And .TextBox4.Value = "" Then
   strBegriff = UserForm1.TextBox1.Value
   strBegriff = "Name='" & strBegriff & "'"
End If

'Suche nach Name und Vorname
If .TextBox1.Value <> "" And .TextBox2.Value <> "" _
   And .TextBox3.Value = "" And .TextBox4.Value = "" Then
   strBegriff = UserForm1.TextBox1.Value
   strBegriff2 = UserForm1.TextBox2.Value
   strBegriff = "Name='" & strBegriff & "' and Vorname='" & strBegriff2 & "'"
End If

'Suche nach Name und Wohnort
If .TextBox1.Value <> "" And .TextBox2.Value = "" _
   And .TextBox3.Value <> "" And .TextBox4.Value = "" Then
   strBegriff = UserForm1.TextBox1.Value
   strBegriff2 = UserForm1.TextBox3.Value
   strBegriff = "Name='" & strBegriff & "' and Wohnort='" & strBegriff2 & "'"
End If
```

Listing 548: Die Suche in der Access-Tabelle durchführen (Forts.)

```
    'Suche nach Beruf
    If .TextBox2.Value <> "" And .TextBox1.Value = "" _
       And .TextBox4.Value = "" And .TextBox3.Value = "" Then
       MsgBox "Diese Suche ist nicht vorgesehen!": Exit Sub
    End If


    'Felder nicht ausgefüllt
    If .TextBox1.Value = "" And .TextBox2.Value = "" _
       And .TextBox3.Value = "" And .TextBox4.Value = "" Then
       MsgBox "Sie müssen die Suchkriterien noch angeben!", vbExclamation
       .TextBox1.SetFocus
       Exit Sub
    End If
End With

weiter:
    Set AccApp = CreateObject("Access.Application")
    'AccApp.Visible = True
    Set DBS = New ADODB.Recordset
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
       "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

    'Tabelle öffnen und abfragen
    If BSchalter = False Then
       DBS.Open _
          "Select * FROM Adressen where (" & strBegriff & ")", _
          conn, adOpenKeyset, adLockOptimistic
    Else
       DBS.Open _
          "Select * FROM Adressen", conn, adOpenKeyset, adLockOptimistic
    End If

    'Gefundene Sätze in mehrspaltiges Listenfeld einfügen
    With UserForm1
       Do Until DBS.EOF
          .ListBox1.AddItem DBS!Name
          .ListBox1.Column(1, intZ) = DBS!Vorname
          .ListBox1.Column(2, intZ) = DBS!Geburtstag
          .ListBox1.Column(3, intZ) = DBS!Wohnort
          .ListBox1.Column(4, intZ) = DBS!Beruf
          intZ = intZ + 1
          DBS.MoveNext
       Loop
       If .ListBox1.ListCount = 0 Then
          .Label6.Caption = "Keine Datensätze gefunden!"
       Else
          .Label6.Caption = .ListBox1.ListCount & " Datensätze gefunden!"
       End If
       .Label6.Visible = True
    End With
```

Listing 548: Die Suche in der Access-Tabelle durchführen (Forts.)

```
    DBS.Close
    AccApp.Quit
    Set DBS = Nothing
    Set conn = Nothing
End Sub
```

Listing 548: Die Suche in der Access-Tabelle durchführen (Forts.)

Deklarieren Sie im ersten Schritt einige Variablen sowie die Objektvariablen für den Zugriff auf die Access-Datentabelle über ADO. Löschen Sie vor jeder neuen Suche das Listenfeld über die Methode `Clear` und setzen Sie die Zählvariable `intZ` auf den Wert 0. Über diese Zählvariable werden Sie später die einzelnen Spalten des mehrspaltigen Listenfelds ansprechen. Des Weiteren setzen Sie die boolesche Variable `BSchalter` standardmäßig auf den Wert `False`. Dieser Schalter wird lediglich dann auf den Wert `True` gesetzt, wenn Sie einen Stern in das Feld `Name` eingeben und danach auf die Schaltfläche SUCHEN klicken. In diesem Fall findet keine Suche statt, sondern es werden alle Datensätze der Access-Datentabelle in das Listenfeld eingelesen.

In den nachfolgenden Zeilen wird ermittelt, welche Suche überhaupt ausgeführt werden soll. Je nachdem, welche Felder Sie mit Suchkriterien ausgefüllt haben, wird entweder ein Suchbegriff bzw. ein Suchbegriff aus zwei einzelnen Feldern zusammengesetzt und später dann an die `SELECT`-Anweisung übergeben. Innerhalb dieses zusammengesetzten Suchbegriffs muss der Feldname des jeweiligen Access-Datentabellenfelds mit angegeben werden.

Mithilfe der Funktion `CreateObject` wird ein neues Access-Objekt erstellt. Die Zeile `AccApp.Visible = True` ist im Listing als Kommentar hinterlegt worden. Wenn Sie diese Zeile aktivieren, dann können Sie erkennen, dass wirklich ein Access-Objekt erstellt wird, d.h., Microsoft Access wird dabei geöffnet und ist im Zustand »sichtbar« (`Visible=true`). Standardmäßig bleibt Microsoft Access aber im Hintergrund (`Visible=False`), was bedeutet, dass der Anwender von dem Zugriff auf die Access-Datentabelle nichts mitbekommt.

Über die Methode `Open` öffnen Sie die Datenbank *Umfeld.mdb*. Gleichzeitig führen Sie eine SQL-Abfrage durch, der Sie den vorher zusammengesetzten Suchbegriff übergeben. Für den Fall, dass Sie vorher einen Stern in das Feld `Name` eingegeben haben, genügt es, den `SELECT`-Befehl ohne weitere Argumente einzusetzen.

Alle gefundenen Adressen werden jetzt im `RecordSet`-Objekt `DBS` zur Verfügung gestellt und können dort direkt abgefragt werden. Übertragen werden diese Informationen in das Listenfeld mithilfe der Methode `AddItem`. Über die Eigenschaft `Column` können Sie jede einzelne Spalte des mehrspaltigen Listenfelds füllen. Mithilfe der Methode `MoveNext` wird jeweils das nächste gefundene Element im `RecordSet` eingestellt. So werden alle Elemente nach und nach ins Listenfeld übertragen, bis die Bedingung `EOF` (»End of File«) zutrifft. Konnten keine Datensätze über die Suche ermittelt werden, dann bleibt das Listenfeld leer. In diesem Fall liefert die Eigenschaft `ListCount` den Wert `0`, was eine Beschriftung des Bezeichnungsfelds »Keine Datensätze gefunden« nach sich zieht. Im anderen Fall wird über dieses noch ausgeblendete Bezeichnungsfeld die Anzahl der gefundenen Datensätze bekannt gegeben. Danach blenden Sie das Bezeichnungsfeld über die Eigenschaft `Visible` ein, indem Sie dieser Eigenschaft den Wert `True` zuweisen.

Schließen Sie im Anschluss daran die Access-Datentabelle über die Methode `Close`. Beenden Sie die Access-Anwendung, indem Sie die Methode `Quit` einsetzen, und heben Sie die Objektverweise über das Schlüsselwort `Nothing` auf, um reservierten Arbeitsspeicher wieder freizugeben (siehe Abbildung 348).

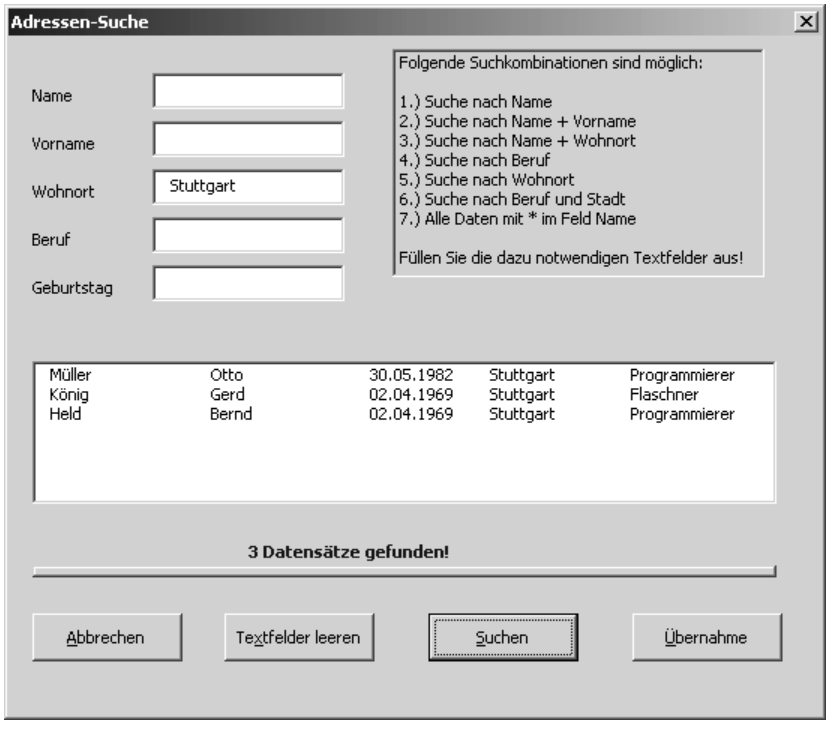


Abbildung 348: Die gefundenen Sätze werden im Listenfeld angezeigt.

Der Klick aufs Listenfeld

Wenn Sie auf einen der gefundenen Datensätze im Listenfeld klicken, dann sollen diese Daten automatisch in die oberen Textfelder übernommen werden (siehe Abbildung 350). Dazu können Sie die Einträge direkt aus dem mehrspaltigen Listenfeld der Userform nehmen und in die Textfelder übertragen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub ListBox1_Click()
   'Daten aus Listenfeld in Textfelder übertragen
   Dim intEintrag As Integer
```

Listing 549: Die Ergebnisse aus dem Listenfeld in die Textfelder übertragen

```
    With UserForm1
       intEintrag = .ListBox1.ListIndex
       .TextBox1.Value = .ListBox1.Column(0, intEintrag)
       .TextBox2.Value = .ListBox1.Column(1, intEintrag)
       .TextBox4.Value = .ListBox1.Column(4, intEintrag)
       .TextBox3.Value = .ListBox1.Column(3, intEintrag)
       .TextBox5.Value = .ListBox1.Column(2, intEintrag)
    End With
End Sub
```

Listing 549: Die Ergebnisse aus dem Listenfeld in die Textfelder übertragen (Forts.)

Mit jedem Klick auf das Listenfeld wird automatisch das Ereignis `Click` ausgelöst. Diesem Ereignis wurden in Listing 549 weitere Befehle zugewiesen, die dann beim Ereigniseintritt mit ausgeführt werden. Über die Eigenschaft `ListIndex` können Sie feststellen, welcher Eintrag im Listenfeld gerade angeklickt wurde. Dabei liefert der erste Eintrag im Listenfeld den Index 0! Mithilfe der Eigenschaft `Column` greifen Sie direkt auf die einzelnen Spalten zu und übertragen die Inhalte über die Eigenschaft `Value` in die dazugehörigen Textfelder.

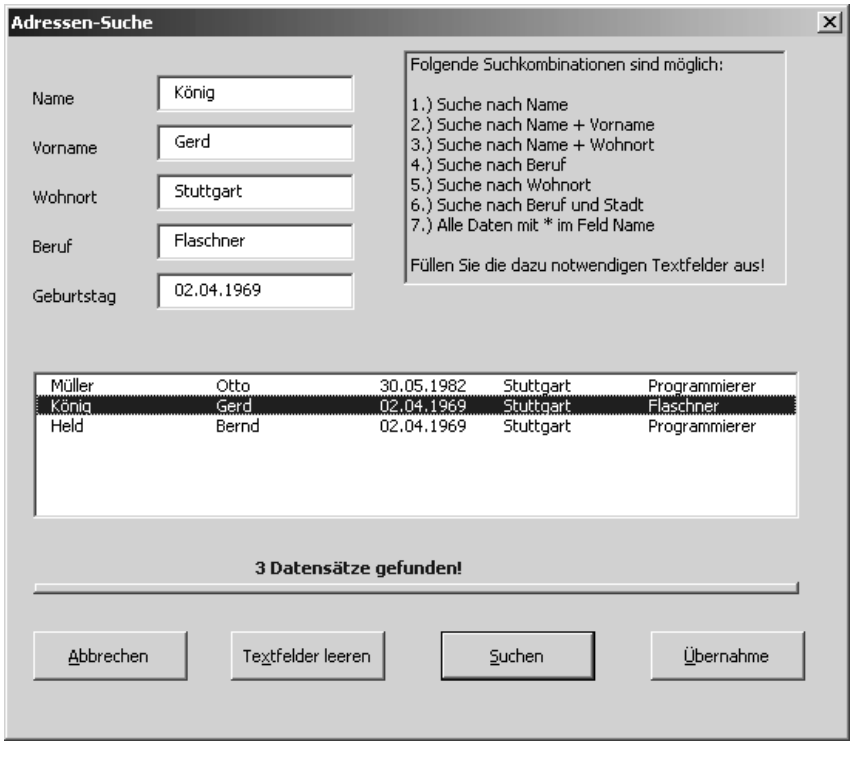


Abbildung 349: Infos mit einem Klick übertragen

Der Optik zuliebe

Beim Verlassen bzw. beim Aktivieren eines Textfelds werden die beteiligten Textfelder dynamisch formatiert. So bekommt ein Textfeld beim Aktivieren eine rote Hintergrundfarbe, die Schriftfarbe wird auf WEISS gesetzt und das Feld wird optisch hervorgehoben. Dazu schreiben Sie zwei Makros, die Sie dem Listing 550 entnehmen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Sub FeldAktiv()
   With UserForm1
      .ActiveControl.BackColor = RGB(255, 0, 0)
      .ActiveControl.ForeColor = RGB(255, 255, 255)
      .ActiveControl.Font.Bold = True
      .ActiveControl.SpecialEffect = fmSpecialEffectRaised
   End With
End Sub


Sub FeldDeaktiv()
   With UserForm1
      .ActiveControl.BackColor = RGB(255, 255, 255)
      .ActiveControl.ForeColor = RGB(0, 0, 0)
      .ActiveControl.Font.Bold = False
      .ActiveControl.SpecialEffect = fmSpecialEffectSunken
   End With
End Sub
```

Listing 550: Felder optisch hervorheben

Über die Eigenschaft `BackColor` lässt sich der Hintergrund eines Textfelds ansprechen. Mithilfe der Funktion `RGB` können Sie die gewünschte Farbe zusammenmischen. Die Farbe Rot hat dabei das Rot-Grün-Blau-Verhältnis `RGB(255, 0, 0)`. Da standardmäßig die schwarze Schriftfarbe eingesetzt wird, die auf rotem Grund etwas schwer zu lesen ist, passen Sie die Schriftfarbe auf Weiß an, indem Sie dem Textfeld die Farbenmischung `RGB(255, 255, 255)` zuweisen. Zusätzlich geben Sie den Text des Felds im Schriftschnitt Fett aus, indem Sie der Eigenschaft `Bold` den Wert `True` zuweisen. Um einen räumlichen Effekt zu erhalten, weisen Sie der Eigenschaft `SpecialEffect` die Konstante `fmSpecialEffectRaised` zu.

Beim Verlassen eines Textfelds müssen diese Einstellungen wieder zurückgesetzt werden. Die beiden Makros aus Listing 550 müssen jetzt den Textfeldereignissen (`Enter` = beim Aktivieren bzw. `Exit` = beim Verlassen) zugewiesen werden. Exemplarisch sehen Sie diese Zuweisung in Listing 551 für die beiden ersten Textfelder.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub TextBox1_Enter()
   FeldAktiv
End Sub

Private Sub TextBox1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
   FeldDeaktiv
End Sub

Private Sub TextBox2_Enter()
   FeldAktiv
End Sub

Private Sub TextBox2_Exit(ByVal Cancel As MSForms.ReturnBoolean)
   FeldDeaktiv
End Sub
```

Listing 551: Den Eintritt und das Verlassen der Textfelder abfangen

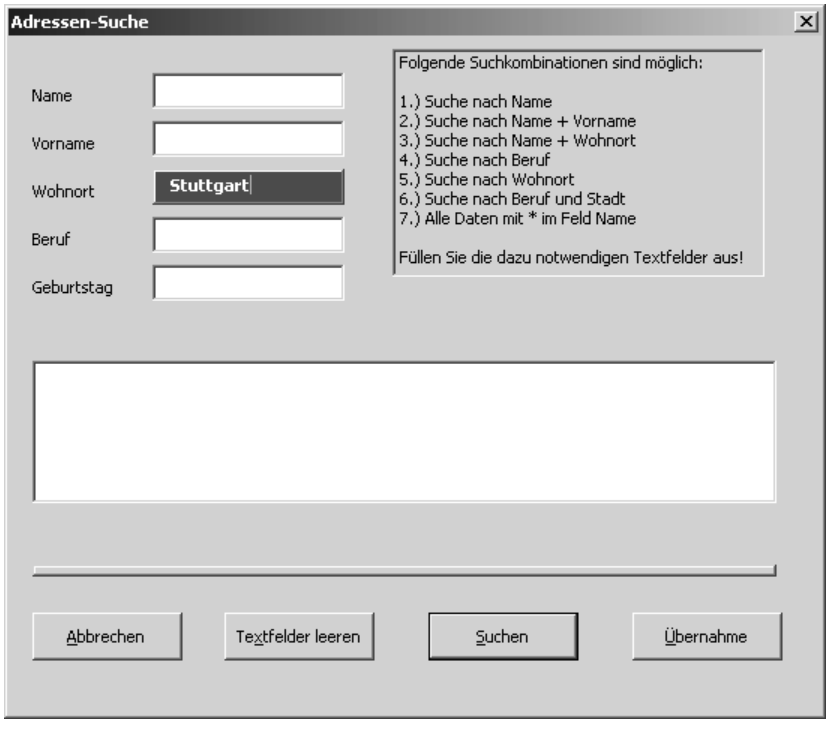


Abbildung 350: Das jeweils aktive Feld wird optisch hervorgehoben.

Textfelder leeren

Zusätzlich wird die Userform mit einer weiteren Funktion bestückt. Über eine Schaltfläche soll es möglich sein, alle Textfelder sowie das Listenfeld von Einträgen zu säubern. Den Code für diese Aufgabe sehen Sie in Listing 552.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=================================================

Private Sub CommandButton4_Click()
   'Textfelder + Listenfeld leeren
   Dim tb As Control

   With UserForm1
      For Each tb In .Controls
         If TypeName(tb) = "TextBox" Then
            tb.Value = ""
         End If
      Next tb
      .ListBox1.Clear
      .Label6.Caption = ""
      .Label6.Visible = False
   End With
End Sub
```

Listing 552: Textfelder leeren

In einer Schleife arbeiten Sie alle einzelnen Steuerelemente der Userform ab. Innerhalb der Schleife prüfen Sie, ob es sich bei dem jeweiligen Element um ein Textfeld handelt. Bei dieser Prüfung verwenden Sie die Funktion `TypeName`, wobei Sie bei der Prüfung auf die korrekte Schreibweise von `TextBox` achten müssen. Diese Funktion unterscheidet zwischen Groß- und Kleinschreibung.

Um das Listenfeld zurückzusetzen, müssen Sie nicht die einzelnen Einträge löschen, sondern Sie können über die Methode `Clear` den kompletten Inhalt des Listenfelds löschen.

Übernahme der Daten in das Excel-Formular

Wurden Daten aus Access gefunden und im Listenfeld eingelesen, dann können Sie die Daten des markierten Datensatzes im Listenfeld in die Excel-Tabelle FORMULAR übertragen. Dazu starten Sie das Makro aus Listing 553.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=================================================
```

Listing 553: Daten aus der Userform in eine Excel-Tabelle übertragen

```
Private Sub CommandButton3_Click()
   'Übernahme der Daten in das Formular
   Dim intEintrag As Integer

   With UserForm1
      If .ListBox1.ListIndex = -1 Then
         MsgBox "Bitte einen Eintrag im Listenfeld markieren!"
      Else
         intEintrag = .ListBox1.ListIndex
         Sheets("Formular").Range("C6").Value = .ListBox1.Column(0, intEintrag)
         Sheets("Formular").Range("C8").Value = .ListBox1.Column(1, intEintrag)
         Sheets("Formular").Range("C10").Value = .ListBox1.Column(2, intEintrag)
         Sheets("Formular").Range("C12").Value = .ListBox1.Column(3, intEintrag)
         Sheets("Formular").Range("C14").Value = .ListBox1.Column(4, intEintrag)
      End If
   End With
End Sub
```

Listing 553: Daten aus der Userform in eine Excel-Tabelle übertragen (Forts.)

Da Sie die Daten in der Excel-Tabelle immer an derselben Position einfügen, können Sie den markierten Listbox-Eintrag direkt in diese Zellen über eine Zuweisung »entleeren«.

Der Datenexport aus Excel

Als nächste Aufgabe führen Sie einen Datenexport aus Excel bzw. einen Datenimport in die Access-Datentabelle durch. Dazu werden Daten über die `Userform2` eingegeben und per Klick direkt in die im Hintergrund befindliche Access-Tabelle geschrieben. Dazu wird die Userform aus Abbildung 352 eingesetzt und folgendes Makro aus Listing 554 ausgeführt.

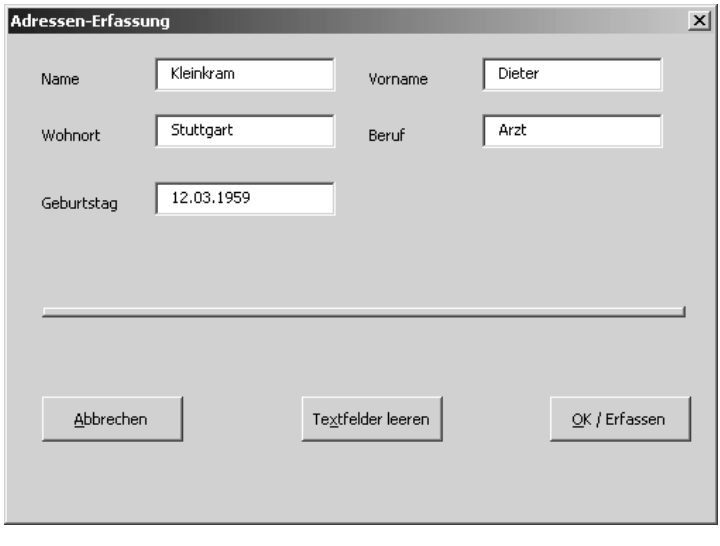


Abbildung 351: Die Userform für die Datenerfassung

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=================================================

Private Sub CommandButton2_Click()
'Daten zurückschreiben
Dim AccApp As Object
Dim conn As New ADODB.Connection
Dim DBS As ADODB.Recordset

Set AccApp = CreateObject("Access.Application")
Set DBS = New ADODB.Recordset
conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
"Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

DBS.Open "Adressen", conn, adOpenKeyset, adLockOptimistic

DBS.AddNew
With UserForm2
  DBS!Name = .TextBox1.Value
  DBS!Vorname = .TextBox2.Value
  DBS!Wohnort = .TextBox3.Value
  DBS!Beruf = .TextBox4.Value
  DBS!Geburtstag = .TextBox5.Value
  DBS.Update
  UserForm2.Label5.Visible = True
  UserForm2.Label5.Caption = "Satz erfasst!"
  DBS.Close
End With

 AccApp.Quit
 Set DBS = Nothing
 Set conn = Nothing
End Sub
```

Listing 554: Einen Datensatz ins Excel-Formular eingeben und per Klick in Access speichern

Auch in diesem Beispiel muss zuerst ein Access-Objekt über die Funktion `CreateObject` erstellt werden. Ebenso müssen die Datenbank sowie die Datentabelle über die Methode `Open` geöffnet werden. Mithilfe der Methode `AddNew` wird ein neuer, noch leerer Satz am Ende der Access-Datentabelle angehängt. Dieser Satz wird danach gefüllt. Dabei werden die Inhalte der Textfelder direkt in die Datenbankfelder geschrieben. Erst durch dem Einsatz der Methode `Update` wird der neue Satz endgültig angelegt. Zum Abschluss werden, wie bereits zuvor beschrieben, die Datentabelle geschlossen, die Access-Applikation beendet sowie die Objektverweise aufgehoben.

Beenden der Userform verhindern

Möchten Sie das Beenden der Userform lediglich über die Schaltfläche ABBRECHEN zulassen, dann erfassen Sie folgendes Ereignis aus Listing 555:

Adressen : Tabelle

	ID	Name	Vorname	Geburtstag	Wohnort	Beruf
▶	1	Günter	Walter	15.12.1970	Mainz	Schlosser
	2	Müller	Werner	06.03.1956	München	Bäcker
	3	Schneider	Stefan	19.10.1982	Berlin	Metzger
	4	Hattemer	Ulrich	23.04.1976	München	Schlosser
	5	Mayer	Michael	13.09.1969	Hamburg	Metzger
	6	Zöller	Peter	17.06.1991	Mainz	Bäcker
	8	Müller	Otto	30.05.1982	Stuttgart	Programmierer
	10	König	Gerd	02.04.1969	Stuttgart	Flaschner
	11	Held	Bernd	02.04.1969	Stuttgart	Programmierer
	12	Kleinkram	Dieter	12.03.1959	Stuttgart	Arzt
*	(AutoWert)					

Datensatz: 1 von 10

Abbildung 352: Der Datensatz wurde erfolgreich in Access gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   If CloseMode <> 1 Then Cancel = True
End Sub
```

Listing 555: Das Beenden einer Userform über das Kreuzsymbol unterbinden

Setzen Sie das Argument `Cancel` auf den Wert `True`, um das Schließen der Userform zu verhindern.

Datenbankabfragen starten

Am Ende des Abschnitts sollen noch zwei Fragen beantwortet werden:

- Wie kann ich die drei ältesten Personen ermitteln?
- Wie kann ich den Namen des »Juniors« abfragen?

Die erste Frage beantwortet das Makro aus Listing 556.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Modul      Modul1
'=====================================================

Sub DieÄltestenFinden()
   'Den Ältesten finden
   Dim AccApp As Object
```

Listing 556: Die drei ältesten Personen werden ermittelt.

```
    Dim conn As New ADODB.Connection
    Dim DBS As ADODB.Recordset
    Dim intZ As Integer
    Dim ArrPersonName(2) As Variant
    Dim ArrPersonVorname(2) As Variant
    Dim ArrDatum(2) As Variant

    Set AccApp = CreateObject("Access.Application")
    Set DBS = New ADODB.Recordset
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
    "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

    DBS.CursorLocation = adUseClient
    DBS.Open "Adressen", conn, adOpenKeyset, adLockOptimistic
    DBS.Sort = "Geburtstag Asc"

    For intZ = 0 To 2
       ArrPersonName(intZ) = DBS!Name
       ArrPersonVorname(intZ) = DBS!Vorname
       ArrDatum(intZ) = DBS!Geburtstag
       DBS.MoveNext
    Next intZ

    MsgBox "Die älteste Person heißt " & ArrPersonVorname(0) & " " & _
       ArrPersonName(0) & " und ist am " & ArrDatum(0) & " geboren!" & vbLf & _
       vbLf & "Die zweit-älteste Person heißt " & ArrPersonVorname(1) & " " & _
       ArrPersonName(1) & " und ist am " & ArrDatum(1) & " geboren!" & vbLf & _
       vbLf & "Die dritt-älteste Person heißt " & ArrPersonVorname(2) & " " & _
       ArrPersonName(2) & " und ist am " & ArrDatum(2) & " geboren!", _
       vbInformation, "Alters-Check"

    DBS.Close
    AccApp.Quit

    Set DBS = Nothing
    Set conn = Nothing
End Sub
```

Listing 556: Die drei ältesten Personen werden ermittelt. (Forts.)

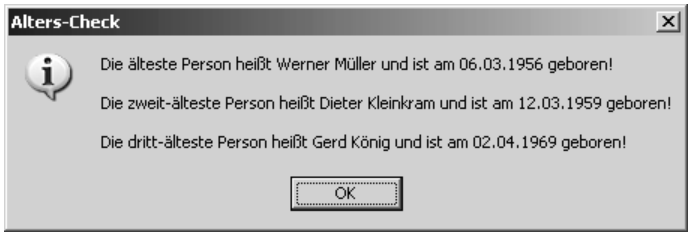


Abbildung 353: Altersauswertung von Excel aus durchführen

Beim Öffnen der Datentabelle wird eine Sortierung eingestellt. Dabei werden im Makro aus Listing 556 die ersten drei Datensätze bzw. im Makro aus Listing 557 der erste Datensatz abgegriffen, in einem Datenfeld gespeichert und am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Abfrage.xls
' Modul      Modul1
'=====================================================

Sub JuniorCheck()
   'Die jüngste Person finden
   Dim AccApp As Object
   Dim conn As New ADODB.Connection
   Dim DBS As ADODB.Recordset
   Dim StrPersonName As String
   Dim StrPersonVorname As String
   Dim DatPerson As Date

   Set AccApp = CreateObject("Access.Application")
   Set DBS = New ADODB.Recordset
   conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
      "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

   DBS.CursorLocation = adUseClient
   DBS.Open "Adressen", conn, adOpenKeyset, adLockOptimistic
   DBS.Sort = "Geburtstag Desc"

   StrPersonName = DBS!Name
   StrPersonVorname = DBS!Vorname
   DatPerson = DBS!Geburtstag

   MsgBox "Die jüngste Person heißt " & StrPersonVorname & " " & _
      StrPersonName & " und ist am " & DatPerson & " geboren!", _
      vbInformation, "Junior-Check"

   DBS.Close
   AccApp.Quit

   Set DBS = Nothing
   Set conn = Nothing
End Sub
```

Listing 557: Die jüngste Person ermitteln

454 Excel-Makro starten

Soll aus Access auf ein Excel-Makro zugegriffen werden, dann wenden Sie die Methode `Run` wie im Listing 558 gezeigt an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub ExcelMakroStarten()
   Dim XL As Object

   On Error GoTo fehler
   Set XL = CreateObject("Excel.Application")

   XL.Workbooks.Open "D:\Eigene Dateien\Mappe1.xls"
   XL.Run "Modul1.MeldungAnzeigen"
   XL.ActiveWorkbook.Close
   XL.Quit

   Set XL = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 558: Excel-Makro aus Access starten

Setzen Sie die Methode `Open` ein, um die Excel-Arbeitsmappe zu öffnen, die das zu startende Makro enthält. Führen Sie danach die Methode `Run` aus, der Sie den Namen des Makros bekannt geben. Es kann dabei auch nicht schaden, genau anzugeben, wo sich das Makro in der Entwicklungsumgebung befindet. Schließen Sie nach dem Ablauf des Makros die Excel-Arbeitsmappe über die Methode `Close` und beenden Sie die Excel-Anwendung mithilfe der Methode `Quit`. Heben Sie danach den Objektverweis `Xl` wieder auf, um den reservierten Speicher wieder freizugeben.

455 Auf Excel-Funktionen zugreifen

Als Worksheet-Funktionen stehen in Excel einige Funktionen zur Verfügung, die in Access so in dieser Form nicht verfügbar sind. In diesem Buch wurde bereits die Funktion `Roman` (`Römisch`) vorgestellt, die über einen Zugriff von Access auf Excel eingesetzt werden konnte.

Im folgenden Beispiel wird exemplarisch die Funktion `Rept` (`Wiederholen`) eingesetzt, um eine Zeichenfolge zu wiederholen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub ExcelFunktionAbfreifen()
   Dim XL As Object

   On Error GoTo fehler
   Set XL = CreateObject("Excel.Application")

   MsgBox XL.WorksheetFunction.Rept("*-*", 10)
   XL.Quit

   Set XL = Nothing
   Exit Sub


   fehler:
      MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 559: Auf Excel-Funktionalität zugreifen

Über die Funktion `CreateObject` wird ein Verweis auf die Excel-Bibliothek eingerichtet. Danach stehen alle Methoden und Eigenschaften dieser Bibliothek zur Verfügung, unter anderem auch die `WorksheetFunction Rept`, über die eine Zeichenfolge beliebig wiederholt werden kann.

Abbildung 354: Eine Zeichenfolge wird wiederholt.

456 Ist PowerPoint bereits gestartet?

Zwischen dem reinen Präsentationsprogramm PowerPoint und der Datenbank Access können Daten ausgetauscht werden. Sogar eine Steuerung von PowerPoint direkt aus Access heraus ist möglich.

Auch bei der Anwendung PowerPoint im Zusammenspiel mit Access wird zuerst einmal geprüft, ob PowerPoint überhaupt gestartet ist. Diese Aufgabe erledigt die Funktion aus Listing 560. Dabei muss die PowerPoint-Bibliothek unter EXTRAS/VERWEISE als Voraussetzung eingebunden sein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlPP
'=====================================================

Function PowerpointOffen() As Boolean
   Dim PowerpointApp As PowerPoint.Application

   On Error Resume Next
   Set PowerpointApp = GetObject(, "Powerpoint.Application")

   If Err.Number = 0 Then
      PowerpointOffen = True
   Else
      PowerpointOffen = False
   End If
End Function
```

Listing 560: Funktion, die prüft, ob PowerPoint geöffnet ist

Verwenden Sie die `GetObject`-Funktion für den Zugriff auf die Objektbibliothek von PowerPoint, und weisen Sie das Objekt einer Objektvariablen zu. Verwenden Sie die `Set`-Anweisung, um das von `GetObject` zurückgegebene Objekt der Objektvariablen zuzuweisen. Ist dieser Vorgang erfolgreich, dann meldet das Objekt `Err` den Wert `0`. In diesem Fall ist PowerPoint bereits gestartet. Als Rückgabewert geben Sie daher den Wahrheitswert `True` an das aufrufende Makro aus Listing 561 zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlPP
'=====================================================

Sub StartPowerpoint()
   Dim PowerpointApp As Object

   If PowerpointOffen = False Then
      Set PowerpointApp = CreateObject("Powerpoint.Application")
      PowerpointApp.Presentations.Add
      PowerpointApp.Visible = True
   End If
End Sub
```

Listing 561: PowerPoint wird je nach Status gestartet oder nicht.

Im Makro aus Listing 561 wurde das Late-Binding durchgeführt und die Objektvariable `PowerpointApp` deklariert. Meldet die Funktion `PowerpointOffen` den Rückgabewert `False`, dann ist PowerPoint noch nicht geöffnet. In diesem Fall wenden Sie die Funktion `CreateObject` an, um ein PowerPoint-Objekt zu erstellen. Danach haben Sie Zugriff auf alle Methoden und

Eigenschaften des Objekts. Geben Sie im Anschluss daran über die Methode `Add` des Objekts `Presentations` an, dass Sie eine neue Präsentation einfügen. Wenden Sie die Eigenschaft `Visible` an, um die Applikation anzuzeigen.

457 PowerPoint-Dateien suchen und öffnen

Für das folgende Beispiel wird zuerst die Objektbibliothek MICROSOFT POWERPOINT in der Entwicklungsumgebung eingebunden. Danach wird ein neues Formular in Access eingefügt und mit einem Listenfeld sowie einer Schaltfläche wie in Abbildung 356 integriert.

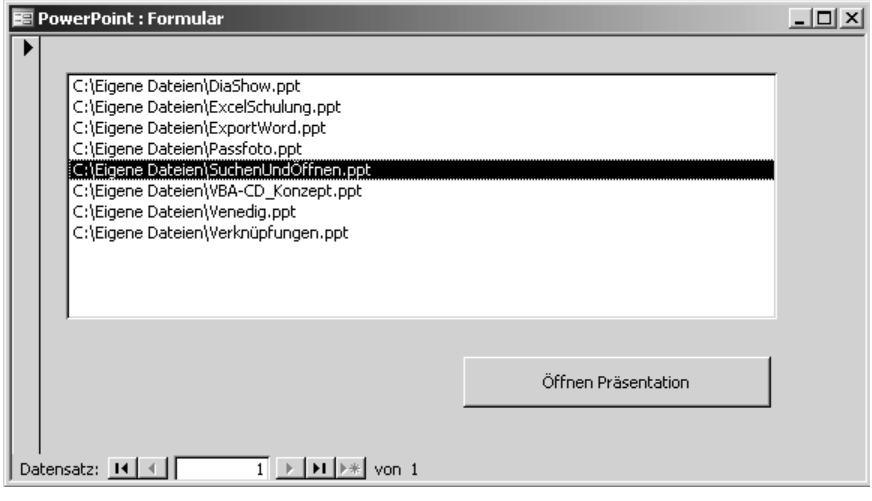


Abbildung 355: Das Formular zum Anzeigen von PowerPoint-Dateien

Beim Öffnen des Formulars sollen alle in einem bestimmten Verzeichnis gefundenen PowerPoint-Präsentationen im Listenfeld angezeigt werden. Für diese Aufgabe stellen Sie das Ereignis `Form_Load` aus Listing 562 ein, das automatisch ausgeführt wird, wenn das Formular geöffnet wird.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Klasse     Form_Powerpoint
'=================================================

Private Sub Form_Load()
   Dim VarPP As Variant

   On Error GoTo fehler
   With Application.FileSearch
      .NewSearch
```

Listing 562: PowerPoint-Präsentationen suchen

```
        .LookIn = "c:\Eigene Dateien\"
        .FileType = msoFileTypePowerPointPresentations
        '.FileName = "*.ppt"
        .SearchSubFolders = True
        If .Execute() > 0 Then
           For Each VarPP In .FoundFiles
              Liste0.AddItem VarPP
           Next VarPP
        End If
    End With
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 562: PowerPoint-Präsentationen suchen (Forts.)

Führen Sie eine Suche über das Objekt `FileSearch` durch und ermitteln Sie dabei alle PowerPoint-Präsentationen eines Verzeichnisses, welches Sie über die Eigenschaft `LookIn` angeben. Für die nähere Spezifikation, welche Dateien gesucht werden sollen, haben Sie zwei Möglichkeiten. Entweder Sie setzen die Eigenschaft `FileType` ein und übergeben der Eigenschaft die Konstante `msoFileTypePowerPointPresentations` oder Sie verwenden die Eigenschaft `FileName` und übergeben dieser Eigenschaft den Suchtext `*.ppt`. Mithilfe der Eigenschaft `SearchSubFolders` können Sie bestimmen, ob auch noch weitere Verzeichnisse, die in der Hierarchie unterhalb des bei der Eigenschaft `LookIn` angegebenen Verzeichnisses liegen, durchsucht werden sollen. Im Beispiel aus Listing 562 wird dies zugelassen.

Über die Methode `Execute` wird die eigentliche Suche gestartet. Dabei wird die Verarbeitung im Makro nur fortgesetzt, wenn überhaupt etwas gefunden wird. Nach einer erfolgreichen Suche stehen die Namen aller gefundenen Dateien im Objekt `FoundFiles`, das Sie nun in einer `For each Next`-Schleife auslesen können. Innerhalb der Schleife wenden Sie die Methode `AddItem` an, um das Listenfeld zu füllen.

Wenn eine PowerPoint-Präsentation im Listenfeld ausgewählt und die Schaltfläche ÖFFNEN PRÄSENTATION angeklickt wird, dann wird das Makro aus Listing 563 gestartet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Klasse     Form_Powerpoint
'=====================================================

Private Sub Befehl2_Click()
   'PowerPoint-Präsentation öffnen
   Dim PPapp As New PowerPoint.Application
```

Listing 563: PowerPoint-Applikation erstellen und Präsentation öffnen

```
    PPapp.Visible = True
    PPapp.Presentations.Open Liste0.Value
End Sub
```

Listing 563: PowerPoint-Applikation erstellen und Präsentation öffnen (Forts.)

Erstellen Sie zunächst ein neues PowerPoint-Objekt über die Deklaration `Dim PPapp as New PowerPoint.Application`. Danach haben Sie automatisch Zugriff auf alle Methoden und Eigenschaften, die dieses Objekt standardmäßig anbietet. Weisen Sie der Eigenschaft `Visible` den Wert `True` zu, um die Anwendung anzuzeigen. Was noch übrig bleibt, ist, die Methode `Open` einzusetzen, um die im Listenfeld ausgewählte Präsentation zu öffnen.

458 Textdateien speichern

In Access haben Sie standardmäßig die Möglichkeit, Tabellen als Textdateien zu speichern. Dazu öffnen Sie die Tabelle, rufen im Menü DATEI den Befehl EXPORTIEREN auf, wählen aus dem Drop-down-Feld DATEITYP den Eintrag TEXTDATEIEN, vergeben einen passenden Dateinamen und bestätigen mit OK. Ebenso können Sie Textdateien in Access einlesen. Dazu rufen Sie im Menü DATEI den Befehl EXTERNE DATEN/IMPORTIEREN auf, stellen im Drop-down-Feld DATEITYP den Eintrag TEXTDATEIEN ein, wählen die gewünschte Textdatei aus und bestätigen mit IMPORTIEREN. Selbstverständlich können Sie beide Aktionen auch per Makro durchführen.

Den Vorgang des Speicherns einer Textdatei können Sie selbstverständlich auch über ein Makro automatisieren. So speichern Sie im folgenden Beispiel die Tabelle `Artikel` in eine Textdatei. Als Trennzeichen verwenden Sie dabei das Semikolon.

Beim Datentransfer aus Listing 564 sollen nur die Datenfelder `Artikelname`, `Liefereinheit`, `Mindestbestand`, `Lagerbestand` und `Bestellte Einheiten` in die Textdatei übertragen werden. Sie brauchen beim folgenden Makro aus Listing 564 die Tabelle `Artikel` vorher nicht zu öffnen. Das Öffnen, Verarbeiten und Transferieren der Daten in die Textdatei wird automatisch vorgenommen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=================================================

Sub TabelleAlsTextdateiSpeichern()
   Dim DBS As New ADODB.Recordset
   Dim strZeile As String
   Dim intz As Integer

   On Error GoTo fehler
   DBS.Open "Artikel", CurrentProject.Connection
```

Listing 564: Eine Tabelle in eine Textdatei schreiben – Variante 1

```
    Open "C:\Eigene Dateien\Artikel.csv" For Output As #1
    intz = 0

    Do Until DBS.EOF
       strZeile = strZeile & DBS!Artikelname & ";" & _
       DBS!Liefereinheit & ";" & DBS!Einzelpreis & ";" & _
       DBS!Mindestbestand & ";" & DBS!Lagerbestand & ";" & _
       DBS!BestellteEinheiten
       DBS.MoveNext
       Print #1, strZeile
       intz = intz + 1
       strZeile = ""
    Loop

    Close #1
    MsgBox "Transfer beendet! Es wurden " & intz _
       & " Sätze übertragen!"
    DBS.Close
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 564: Eine Tabelle in eine Textdatei schreiben – Variante 1 (Forts.)

Definieren Sie im ersten Schritt ein ADO-Objekt vom Typ `Recordset`. Damit haben Sie Zugriff auf alle Tabellen in Access. Definieren Sie danach eine String-Variable, um später die einzelnen Datenfelder in dieser Variablen zwischenzuspeichern. In der folgenden Variablen `intz` sollen die übertragenen Datensätze gezählt und später ausgegeben werden. Öffnen Sie daraufhin mithilfe der Methode `Open` die Tabelle `Artikel` in der aktuell geöffneten Datenbank. Mit der Anweisung `CurrentProject.Connection` fangen Sie den Namen der aktuellen Datenbank ein.

Öffnen Sie dann über die Methode `Open` die Textdatei, in welche die Daten hinein transferiert werden sollen. Dabei muss diese Textdatei keineswegs schon angelegt sein. Der Befehl `Open` erstellt diese neu, sofern sie noch nicht existiert. Die Syntax dieser Methode lautet:

```
Open Pfadname For Modus [Access-Zugriff] [Sperre] As [#]Dateinummer [Len=Satzlänge]
```

Im Argument `Pfadname` geben Sie das Laufwerk sowie den Dateinamen der Textdatei an, die Sie öffnen möchten.

Das Argument `Modus` legt den Zugriffsmodus für die Datei fest. Möglich sind die Modi `Append`, `Binary`, `Input`, `Output` oder `Random`. Wenn kein Modus festgelegt ist, wird die Datei im Zugriffsmodus `Random` geöffnet. Im Beispiel aus Listing 564 wurde die Textdatei im Zugriffsmodus `Output` geöffnet, was darauf schließen lässt, dass Sie Sätze in diese Textdatei transferieren möchten. Würden Sie die Textdatei mit dem Zugriffsmodus `Append` öffnen, würde das Makro bei mehrmaligem Start hintereinander die Daten immer unten an die Textdatei anhängen, was zur Folge hätte, dass die Datensätze mehrfach vorliegen würden.

Das nächste optionale Argument `Access-Zugriff` bestimmt die Operation, die mit der geöffneten Datei ausgeführt werden soll. Dabei wählen Sie entweder `Read`, wenn Sie die Textdatei nur lesen, oder `Write`, wenn Sie etwas in die Textdatei schreiben möchten. Möchten Sie beide Aktionen durchführen, dann setzen Sie die Konstante `Read Write` ein.

Beim optionalen Argument `Sperre` können Sie bestimmen, welche Operationen mit der geöffneten Datei von anderen Anwendungen durchgeführt werden dürfen oder nicht. Dabei können Sie bestimmte Zugriffe wie das Lesen und Schreiben zulassen oder verwehren. Zur Auswahl stehen `Shared`, `Lock Read`, `Lock Write` und `Lock Read Write`.

Mithilfe des Arguments `Dateinummer` vergeben Sie eine gültige Dateinummer im Bereich von 1 bis 511. Dabei haben Sie die Möglichkeit, bei den Modi `Binary`, `Input` und `Random` eine Datei mit einer anderen Dateinummer zu öffnen, ohne sie zuvor schließen zu müssen. In den Modi `Append` und `Output` müssen Sie eine Datei erst schließen, bevor sie mit einer anderen Dateinummer geöffnet werden kann. Wenn Sie dennoch versuchen, in eine bereits so geöffnete Textdatei zu schreiben, werden Sie daran gehindert und Sie erhalten eine Fehlermeldung.

Im letzten optionalen Argument `Satzlänge` können Sie der Textdatei noch eine Satzlänge vorgeben. Dabei handelt es sich um eine Zahl, die kleiner oder gleich 32.767 (Byte) ist. Bei Dateien mit wahlfreiem Zugriff ist dies die Datensatzlänge, bei sequenziellen Dateien die Anzahl der gepufferten Zeichen.

Haben Sie nun sowohl die Tabelle als auch die Textdatei geöffnet, setzen Sie die Zählvariable `intz` auf den Wert 0. Danach setzen Sie eine Schleife auf, die so lange durchlaufen wird, bis der letzte Datensatz in der Tabelle erreicht ist. Diese automatische Überprüfung nimmt die Eigenschaft `EOF` (End Of File) vor. Sie dürfen allerdings nicht vergessen, den Datenzeiger mithilfe der Methode `MoveNext` jeweils einen Satz weiter zu setzen, sobald Sie einen Satz übertragen haben. Sie erzeugen ansonsten eine Endlosschleife. Ebenfalls innerhalb der Schleife basteln Sie sich Ihre Variable `strZeile` zusammen, die die einzelnen Datenfelder aus der Tabelle `Artikel` aufnimmt. Vergessen Sie dabei nicht, nach jeder Feldinformation das Trennzeichen (;) einzusetzen.

Mithilfe der Anweisung `Print` drucken Sie den auf diese Weise zusammengebastelten augenblicklichen Inhalt der Variablen `strZeile` direkt in die geöffnete Textdatei. Dabei müssen Sie dieselbe Dateinummer verwenden, wie sie schon beim Öffnen der Textdatei zum Einsatz kam. Nur so wird gewährleistet, dass der »Ausdruck« auch in die gewünschte Textdatei kommt.

459 Weitere Methoden zum Speichern von Textdateien

Mithilfe der Methode `TransferText` können Sie genau diese Aufgabe ebenso gut ausführen. Der Unterschied zur gerade vorgestellten Lösung besteht darin, dass Sie mit dieser Methode die komplette Tabelle als Textdatei speichern müssen. Sie haben dabei keine Möglichkeit, einzelne Felder zu ignorieren.

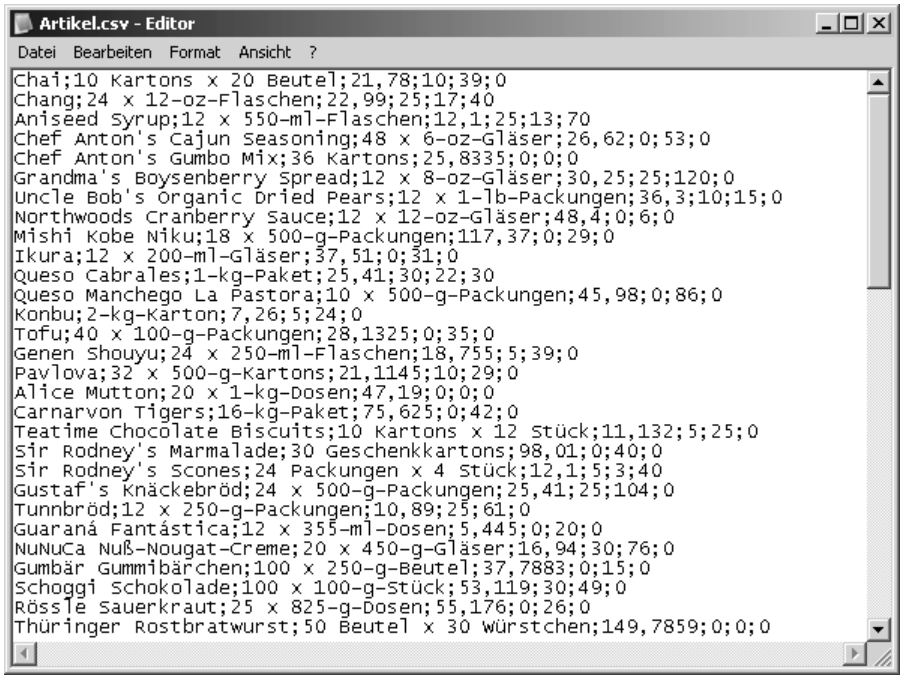


Abbildung 356: Die CSV-Textdatei wurde aus der Tabelle Artikel erzeugt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=====================================================

Sub TabelleAlsTextdateiSpeichernSchnell()
   DoCmd.TransferText acExportDelim, , "Artikel", "C:\Eigene Dateien\Artikel.txt"
End Sub
```

Listing 565: Eine Tabelle in eine Textdatei schreiben – Variante 2

Mithilfe der Methode `TransferText` können Sie sowohl Daten in Textdateien exportieren als auch Daten aus Textdateien in Access-Tabellen einlesen.

Eine weitere Möglichkeit, um Access-Tabellen in Textdateien zu speichern, bietet das `DoCmd`-Objekt. So speichert das Makro aus Listing 566 die Tabelle `Artikel` im Verzeichnis *C:\Eigene Dateien* unter dem Namen *Artikel.txt*.

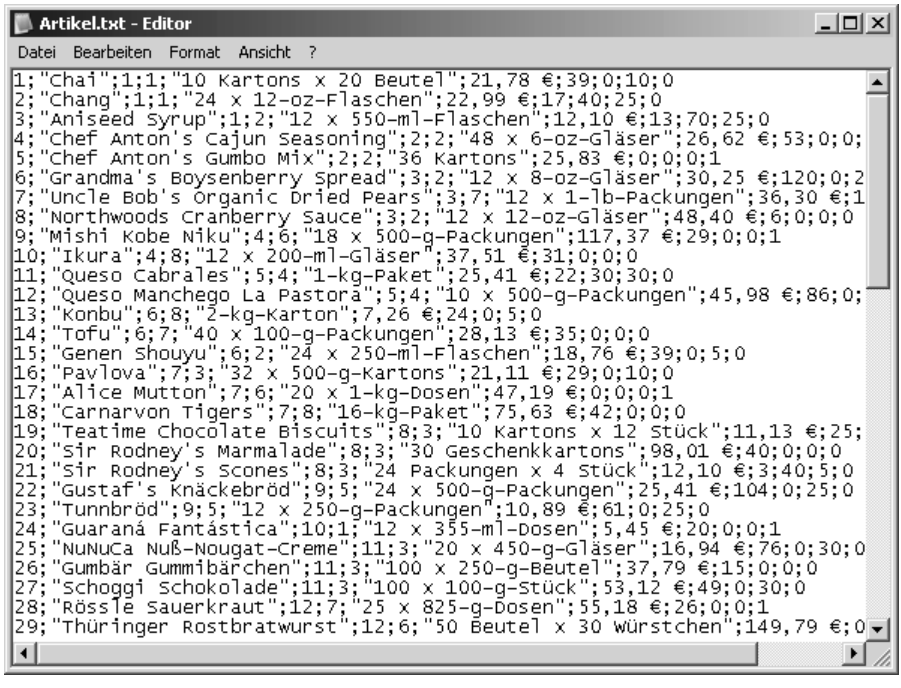


Abbildung 357: Textdatei aus Tabelle erstellen

```
Sub TabelleTransferieren()
   DoCmd.OutputTo acOutputTable, "Artikel", acFormatTXT, _
      "C:\Eigene Dateien\Artikel.txt", True
End Sub
```

Listing 566: Eine Tabelle in eine Textdatei schreiben – Variante 3

Mit der Methode `OutputTo` können Sie die Daten in einem bestimmten Microsoft Access-Datenbankobjekt (Datenblatt, Formular, Bericht, Modul oder Datenzugriffsseite) in verschiedenen Formaten ausgeben.

460 Codes sichern

Die Methode `OutputTo` bietet Ihnen die Möglichkeit, Ihre Programmierung zu sichern. Das Makro aus Listing 567 sichert das Modul MODUL1 in der Textdatei *Code.txt* im Verzeichnis *C:\Eigene Dateien*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=====================================================

Sub ModulTransferieren()
   DoCmd.OutputTo acOutputModule, "Modul1", _
      acFormatTXT, "C:\Eigene Dateien\Code.txt", True
End Sub
```

Listing 567: Codes in einer Textdatei speichern

Indem Sie die Konstante `acOutputModule` verwenden, erkennt Access, dass es dabei aus der Entwicklungsumgebung das MODUL1 exportieren soll. Mit der Konstante `acFormatTXT` legen Sie fest, dass dieser Transfer im Textformat erfolgen soll.

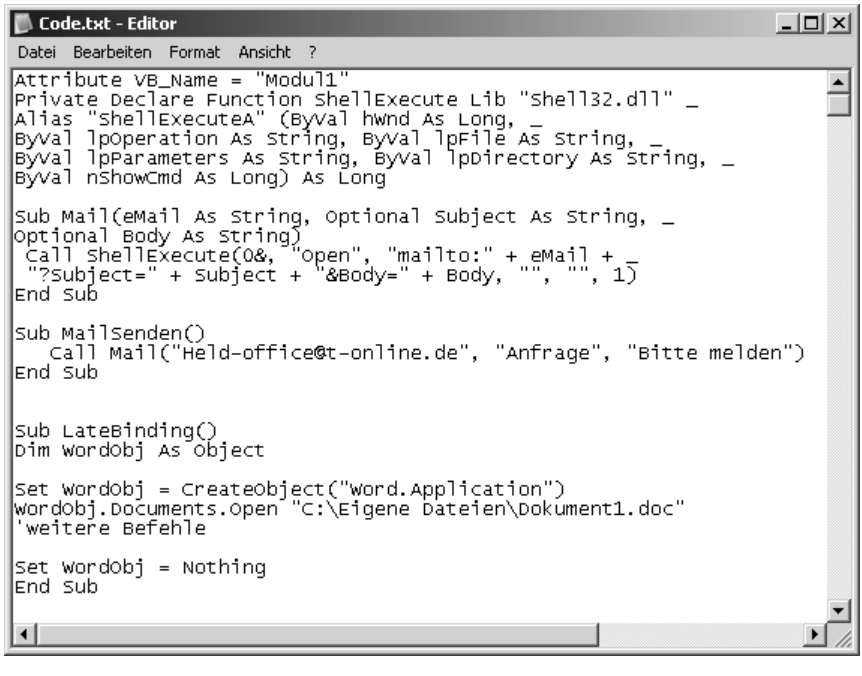


Abbildung 358: Makros wurden in eine Textdatei gesichert.

Haben Sie mehrere Module in Ihrer Datenbank untergebracht und möchten Sie diese alle sichern, dann setzen Sie das Makro Listing 568 ein.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=================================================

Sub AlleModuleSpeichern()
   Dim obj As AccessObject
   Dim dbs As Object
   Dim intz As Integer

   intz = 0
   Set dbs = Application.CurrentProject

   For Each obj In dbs.AllModules
      intz = intz + 1
      DoCmd.OutputTo acOutputModule, obj.Name, _
      acFormatTXT, "C:\Eigene Dateien\Code" & intz & ".txt"
   Next obj
End Sub
```

Listing 568: Alle Makros in mehreren Textdateien speichern

Im Auflistungsobjekt `AllModules` sind alle Module der Datenbank verzeichnet. Sie können damit ganz elegant ein Modul nach dem anderen mithilfe einer Schleife ansprechen. Innerhalb der Schleife wenden Sie die Methode `OutputTo` an, um die einzelnen Module zu sichern. Dabei geben Sie den einzelnen Textdateien einen fortlaufenden Namen, der sich aus dem Text `Code` und der Zählvariablen `intz` zusammensetzt.

461 Textdatei importieren

Für den Import von Textdateien haben Sie in Access die Möglichkeit, eine eigene Importspezifikation zu erstellen und auf Basis dieser Spezifikation Textdateien einzulesen. Wie das genau funktioniert, können Sie in Kapitel 3 dieses Buchs nachlesen.

Eine alternative Möglichkeit ist, die Textdatei selbst über ein Makro einzulesen. Im folgenden Beispiel aus Listing 569 werden aus der Textdatei *Artikel.csv* die Informationen `Artikelname`, `Liefereinheit` und `Einzelpreis` in die Tabelle `Artikel2` eingelesen.

```
'=================================================
' Auf CD     Buchdaten\Beispiele\Kap10
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=================================================

Sub TextdateiInTabelleEinlesen()
   Dim dbs As New ADODB.Recordset
   Dim strZeile As String
   Dim intPos1 As Integer
```

Listing 569: Teile einer Textdatei einlesen – Variante 1

```
    On Error GoTo fehler
    dbs.Open "Artikel2", CurrentProject.Connection, , adLockOptimistic
    Open "C:\Eigene Dateien\Artikel.csv" For Input As #1

    Do While Not EOF(1)
       dbs.AddNew
       Line Input #1, strZeile
       dbs!Artikelname = Left(strZeile, InStr(strZeile, ";") - 1)
       intPos1 = InStr(strZeile, ";") + 1
       strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
       dbs!Liefereinheit = Left(strZeile, InStr(strZeile, ";") - 1)
       intPos1 = InStr(strZeile, ";") + 1
       strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
       dbs!Einzelpreis = Left(strZeile, InStr(strZeile, ";") - 1)
       dbs.Update
    Loop
    Close #1
    dbs.Close
    Exit Sub


    fehler:
       MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 569: Teile einer Textdatei einlesen – Variante 1 (Forts.)

Öffnen Sie zuerst die Access-Tabelle `Artikel2` über die Methode `Open`. Damit die Tabelle editiert werden kann, verwenden Sie die Konstante `adLockOptimistic`. Öffnen Sie danach ebenfalls über die Methode `Open` die Textdatei. Da Sie diese Datei lesen möchten, verwenden Sie dabei das Argument `For Input`.

In einer Schleife arbeiten Sie alle Datensätze der Textdatei ab. Über den Einsatz der Methode `AddNew` fügen Sie einen neuen, noch leeren Datensatz in die Tabelle ein, den Sie direkt im Anschluss mit dem Inhalt aus der Textdatei füllen. Dabei arbeiten Sie sich Position für Position vor. Zu Beginn steht in der Variablen `StrZeile` noch der komplette Datensatz aus der Textdatei, den Sie über die Anweisung `Line Input` füllen. Danach springen Sie von Semikolon zu Semikolon, wobei Sie den String jeweils kürzen.

Haben Sie die Datenfelder gefüllt, dann setzen Sie die Methode `Update` ein, um den Datensatz zu sichern. Schließen Sie am Ende des Makros die Datentabelle sowie die Textdatei über die Methode `Close`.

Artikel2 : Tabelle

Artikel-Nr	Artikelname	Liefereinheit	Einzelpreis
489	Chai	10 Kartons x 20 Beutel	21,78 €
490	Chang	24 x 12-oz-Flaschen	22,99 €
491	Aniseed Syrup	12 x 550-ml-Flaschen	12,10 €
492	Chef Anton's Cajun Seasoning	48 x 6-oz-Gläser	26,62 €
493	Chef Anton's Gumbo Mix	36 Kartons	25,83 €
494	Grandma's Boysenberry Spread	12 x 8-oz-Gläser	30,25 €
495	Uncle Bob's Organic Dried Pears	12 x 1-lb-Packungen	36,30 €
496	Northwoods Cranberry Sauce	12 x 12-oz-Gläser	48,40 €
497	Mishi Kobe Niku	18 x 500-g-Packungen	117,37 €
498	Ikura	12 x 200-ml-Gläser	37,51 €
499	Queso Cabrales	1-kg-Paket	25,41 €
500	Queso Manchego La Pastora	10 x 500-g-Packungen	45,98 €
501	Konbu	2-kg-Karton	7,26 €
502	Tofu	40 x 100-g-Packungen	28,13 €
503	Genen Shouyu	24 x 250-ml-Flaschen	18,76 €

Datensatz: 1 von 77

Abbildung 359: Die Daten wurden erfolgreich eingelesen.

Hinweis

Wenn Sie sich das Makro aus Listing 569 noch einmal ansehen, dann sehen Sie, dass einige Zeilen darin redundant sind. Wie kann dieses Makro umgeschrieben werden, damit es kürzer und schneller abgearbeitet werden kann?

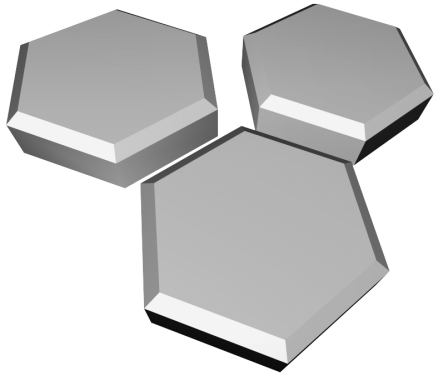
Nun, das stückweise Verkürzen der String-Variablen `StrZeile` kann in eine Schleife gepackt werden. Dadurch müssen die Datenbankfelder variabel angesprochen werden, wie es im Listing 570 gemacht wird.

```
'=====================================================
' Auf CD Buchdaten\Beispiele\Kap10
' Dateiname Umfeld.mdb
' Modul mdlTXT
'=====================================================

Sub TextdateiInTabelleEinlesenSchneller()
Dim dbs As New ADODB.Recordset
Dim strZeile As String
Dim intz As Integer
Dim intPos1 As Integer

On Error GoTo fehler
dbs.Open "Artikel3", CurrentProject.Connection, , adLockOptimistic
Open "C:\Eigene Dateien\Artikel.csv" For Input As #1

Do While Not EOF(1)
 Line Input #1, strZeile
 dbs.AddNew
 For intz = 1 To 3
 dbs.Fields(intz) = Left(strZeile, InStr(strZeile, ";") - 1)
 intPos1 = InStr(strZeile, ";") + 1
 strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
 Next intz
 dbs.Update
Loop
Close #1
dbs.Close
Exit Sub


fehler:
 MsgBox Err.Number & " " & Err.Description
End Sub
```

Listing 570: Teile einer Textdatei einlesen – Variante 2

Über die Zählvariable `intz` können Sie in einer Schleife die Datenbankfelder `Artikelname` (erstes Feld) bis `Einzelpreis` (drittes Feld) ansprechen. Das erste Datenfeld der Tabelle (`Artikel-Nr`) wird durch den Index selbst geschrieben. Da die Funktion `Instr` immer nur das erste Semikolon im String finden würde, wird der String bei jedem Schleifendurchlauf gekürzt, indem die bereits übertragenen Informationen aus dem String entfernt werden. Sie arbeiten sich dabei also von links nach rechts in der Variablen `strZeile` vor.

Referenz



VBA-Funktionen

Datums- und Zeitfunktionen

Funktion	Kurzbeschreibung
`CDate`	Wandelt eine Zeichenfolge in einen Datumswert um.
`Date`	Gibt das aktuelle Systemdatum aus.
`DateAdd`	Liefert einen Wert zurück, der ein Datum enthält, zu dem ein bestimmtes Zeitintervall addiert wurde.
`DateDiff`	Liefert einen Wert zurück, der die Anzahl der Zeitintervalle zwischen zwei bestimmten Terminen angibt.
`DatePart`	Liefert einen Wert zurück, der einen bestimmten Teil eines angegebenen Datums enthält.
`DateSerial`	Liefert einen Wert zurück, der die angegebene Jahres-, Monats- und Tageszahl enthält.
`DateValue`	Wandelt eine Zeichenfolge in einen gültigen Datumswert um.
`Day`	Extrahiert den Tag als ganzzahligen Wert (1–31) aus einem Datumswert.
`FileDateTime`	Liefert das Erstellungsdatum bzw. das letzte Änderungsdatum einer Datei.
`FormatDateTime`	Gibt einen als Datum oder Uhrzeit formatierten Ausdruck zurück.
`Hour`	Liefert die Stunde aus einem Datums-/Zeitwert als ganzzahligen Wert (0–23).
`Minute`	Liefert die Minute aus einem Datums-/Zeitwert als ganzzahligen Wert (0–59).
`Month`	Extrahiert den Monat aus einem Datumswert als ganzzahligen Wert (1–12).
`Monthname`	Gibt eine Zeichenfolge zurück, die den festgelegten Monat angibt.
`Now`	Liefert das Systemdatum inklusive der Uhrzeit.
`Second`	Liefert die Sekunde aus einem Datums-/Zeitwert als ganzzahligen Wert (0–59).
`Time`	Gibt die aktuelle Systemzeit aus.
`Timer`	Gibt einen Wert vom Typ `Single` zurück, der die Anzahl der seit Mitternacht vergangenen Sekunden angibt.
`TimeSerial`	Setzt einen Datums-/Zeitwert aus Ganzzahlwerten (Sekunden, Minuten und Stunden) zusammen.
`TimeValue`	Wandelt eine Zeichenfolge in einen gültigen Zeitwert um.

Tabelle 159: Die Funktionen der Kategorie Datum und Zeit

Funktion	Kurzbeschreibung
`WeekDay`	Gibt den Wochentag aus einem Datumswert zurück (1–7).
`WeekDayName`	Gibt eine Zeichenfolge zurück, die den festgelegten Wochentag angibt.
`Year`	Extrahiert die Jahresinformation aus einem Datumswert.

Tabelle 159: Die Funktionen der Kategorie Datum und Zeit (Forts.)

Textfunktionen

Funktion	Kurzbeschreibung
`Asc`	Gibt einen Wert vom Typ `Integer` zurück, der den Zeichencode entsprechend dem ersten Buchstaben in einer Zeichenfolge darstellt.
`Choose`	Wählt einen Wert aus einer Liste von Argumenten aus und gibt ihn zurück.
`Chr`	Gibt einen Wert vom Typ `String` zurück, der das Zeichen enthält, das dem angegebenen Zeichencode zugeordnet ist.
`InStr`	Gibt einen Wert vom Typ `Variant` (`Long`) zurück, der die Position des ersten Auftretens einer Zeichenfolge innerhalb einer anderen Zeichenfolge angibt.
`InStrRev`	Gibt die Position eines Vorkommnisses einer Zeichenfolge in einer anderen Zeichenfolge vom Ende der Zeichenfolge gesehen an.
`Join`	Gibt eine Zeichenfolge zurück, die sich aus der Kombination einer Reihe von untergeordneten Zeichenfolgen, die in einem Datenfeld enthalten sind, ergibt.
`LCase`	Wandelt Großbuchstaben in Kleinbuchstaben um.
`Left`	Gibt einen Wert vom Typ `Variant` (`String`) zurück, der eine bestimmte Anzahl von Zeichen ab dem ersten (linken) Zeichen einer Zeichenfolge enthält.
`Len`	Gibt einen Wert vom Typ `Long` zurück, der die Anzahl der Zeichen in einer Zeichenfolge oder die zum Speichern einer Variablen erforderlichen Byte enthält.
`Mid`	Gibt einen Wert vom Typ `Variant` (`String`) zurück, der eine bestimmte Anzahl von Zeichen aus einer Zeichenfolge enthält.
`Replace`	Gibt eine Zeichenfolge zurück, in der eine festgelegte, untergeordnete Zeichenfolge mit einer festgelegten Häufigkeit durch eine andere untergeordnete Zeichenfolge ersetzt wurde.
`Right`	Gibt einen Wert vom Typ `Variant` (`String`) zurück, der eine bestimmte Anzahl von Zeichen von der rechten Seite (dem Ende) einer Zeichenfolge enthält.

Tabelle 160: Die Textfunktionen

Funktion	Kurzbeschreibung
Space	Gibt eine Zeichenfolge vom Typ Variant (String) zurück, die aus einer angegebenen Anzahl von Leerzeichen besteht.
Spc	Fügt in einer Textdatei eine bestimmte Anzahl von Leerzeichen ein.
Split	Gibt ein nullbasiertes, eindimensionales Datenfeld zurück, das eine festgelegte Anzahl an untergeordneten Zeichenfolgen enthält.
Str	Gibt einen Wert vom Typ Variant (String) zurück, der eine Zahl darstellt.
StrComp	Gibt einen Wert vom Typ Variant (Integer) zurück, der das Ergebnis eines Zeichenfolgenvergleichs anzeigt.
StrConv	Gibt einen Wert vom Typ Variant (String) zurück, der wie angegeben umgewandelt wurde.
StrReverse	Gibt eine Zeichenfolge zurück, in der die Reihenfolge der Zeichen einer bestimmten Zeichenfolge umgekehrt wurde.
String	Gibt eine Zeichenfolge vom Typ Variant (String) zurück, die ein sich wiederholendes Zeichen der angegebenen Länge enthält.
Switch	Wertet eine Liste von Ausdrücken aus und gibt einen Wert vom Typ Variant oder einen Ausdruck zurück, der dem ersten Ausdruck in der Liste zugeordnet ist, der True ergibt.
Trim	Gibt einen Wert vom Typ Variant (String) zurück, der eine Kopie einer bestimmten Zeichenfolge enthält, die keine führenden Leerzeichen (LTrim), keine nachgestellten Leerzeichen (RTrim) sowie keine Kombination aus führenden und nachgestellten Leerzeichen (Trim) enthält.
UCase	Wandelt Kleinbuchstaben in Großbuchstaben um.

Tabelle 160: Die Textfunktionen (Forts.)

Dateifunktionen

Funktion/Anweisung	Kurzbeschreibung
ChDir (A)	Wechselt das aktuelle Verzeichnis oder den aktuellen Ordner.
ChDrive (A)	Wechselt das aktuelle Laufwerk.
Close (A)	Beendet das Lesen aus und das Schreiben in eine Datei, die mit der Open-Anweisung geöffnet wurde.
CurDir (A)	Gibt einen Wert vom Typ Variant (String) zurück, der den aktuellen Pfad darstellt.
Dir (F)	Gibt eine Zeichenfolge (String) zurück, die den Namen einer Datei, eines Verzeichnisses oder eines Ordners darstellt, der mit einem bestimmten Suchmuster, einem Dateiattribut oder mit der angegebenen Datenträger- bzw. Laufwerksbezeichnung übereinstimmt.

Tabelle 161: Die Funktionen und Anweisungen der Kategorie Dateifunktionen und -anweisungen

Funktion/Anweisung	Kurzbeschreibung
`Environ (F)`	Gibt die mit einer Betriebssystem-Umgebungsvariablen verbundene Zeichenfolge (`String`) zurück.
`EOF (F)`	Gibt einen Wert vom Typ `Integer` zurück, der den `Boolean`-Wert `True` enthält, wenn das Ende einer Datei, die im Zugriffsmodus `Random` oder `Input` geöffnet wurde, erreicht worden ist.
`FileAttr (F)`	Gibt einen Wert vom Typ `Long` zurück, der den Zugriffsmodus für mit der `Open`-Anweisung geöffnete Dateien darstellt.
`FileCopy (A)`	Kopiert eine Datei.
`FileDateTime (F)`	Gibt einen Wert vom Typ `Variant` (`Date`) zurück, der den Tag und die Uhrzeit der Erstellung bzw. der letzten Änderung der Datei anzeigt.
`FileLen (F)`	Gibt einen Wert vom Typ `Long` zurück, der die Länge einer Datei in Byte angibt.
`FreeFile (F)`	Gibt einen Wert vom Typ `Integer` zurück, der die nächste verfügbare Dateinummer darstellt, die die `Open`-Anweisung zum Öffnen einer Datei verwenden kann.
`Get (A)`	Liest Daten aus einer geöffneten Datenträgerdatei in eine Variable ein.
`GetAttr (F)`	Gibt einen Wert vom Typ `Integer` zurück, der die Attribute einer Datei, eines Verzeichnisses darstellt.
`Input (F)`	Gibt einen Wert vom Typ `String` zurück, der Zeichen aus einer im Modus `Input` oder `Binary` geöffneten Datei enthält.
`Kill (A)`	Löscht eine Datei ohne Rückfrage.
`Line Input (A)`	Liest eine einzelne Zeile aus einer geöffneten sequentiellen Datei und weist sie einer Variablen vom Typ `String` zu.
`Loc (F)`	Gibt einen Wert vom Typ `Long` zurück, der die aktuelle Schreib-/Leseposition innerhalb einer geöffneten Datei angibt.
`Lock (A)` `Unlock (A)`	Regelt die Zugriffsmöglichkeiten anderer Prozesse auf eine Datei (oder auf Teile einer Datei), die mit der `Open`-Anweisung geöffnet wurde.
`LOF (F)`	Gibt einen Wert vom Typ `Long` zurück, der die Größe einer mit der `Open`-Anweisung geöffneten Datei in Byte angibt.
`LSet (A)`	Richtet eine Zeichenfolge innerhalb einer Zeichenfolgenvariablen links aus oder kopiert eine Variable eines benutzerdefinierten Datentyps in eine Variable eines anderen benutzerdefinierten Datentyps.
`MkDir (A)`	Erstellt ein neues Verzeichnis.
`Name (A)`	Benennt eine Datei, ein Verzeichnis oder einen Ordner um.
`Open (A)`	Öffnet eine Datei für die Ein- bzw. Ausgabe.
`Print (A)`	Schreibt Daten, die für die Ausgabe formatiert sind, in eine sequentielle Datei.

Tabelle 161: Die Funktionen und Anweisungen der Kategorie Dateifunktionen und -anweisungen (Forts.)

Funktion/Anweisung	Kurzbeschreibung
`Put (A)`	Schreibt Daten aus einer Variablen in eine Datenträgerdatei.
`Reset (A)`	Schließt alle Datenträgerdateien, die mit der `Open`-Anweisung geöffnet wurden.
`RmDir (A)`	Löscht ein Verzeichnis.
`Seek (F)`	Gibt einen Wert vom Typ `Long` zurück, der die aktuelle Schreib-/Leseposition in einer Datei festlegt, die mit der `Open`-Anweisung geöffnet wurde.
`SetAttr (A)`	Legt die Dateiattribute fest.
`Shell (F)`	Führt ein ausführbares Programm aus. Falls erfolgreich, wird ein Wert vom Typ `Variant` (`Double`) zurückgegeben, der die Task-ID des Programms darstellt. Andernfalls wird Null zurückgegeben.
`Write (A)`	Schreibt Daten in eine sequentielle Datei.

Tabelle 161: Die Funktionen und Anweisungen der Kategorie Dateifunktionen und -anweisungen (Forts.)

Mathematische Funktionen

Funktion	Kurzbeschreibung
`Abs`	Gibt den Absolutwert einer Zahl zurück. Das Vorzeichen wird dabei ignoriert.
`Atn`	Gibt den Arcustangens einer Zahl als Winkel im Bogenmaß zurück.
`Cos`	Gibt den Cosinus zu einem Winkel im Bogenmaß zurück.
`DDB`	Errechnet den Abschreibungswert bei geometrisch degressiver Abschreibung.
`Exp`	Gibt den Wert der Exponentialfunktion zu einer Zahl zurück.
`Fix`	Gibt den Vorkommawert einer Zahl zurück.
`FV`	Errechnet den zukünftigen Wert einer Annuität bei konstanter Zahlung, festem Zins und fester Laufzeit.
`Int`	Gibt den ganzzahligen Wert einer Zahl mit Dezimalen zurück.
`IPmt`	Errechnet den Zinsanteil für eine bestimmte Periode bei Ansparung bzw. Abzahlung in konstanten Raten und mit festem Zinssatz.
`IRR`	Gibt den internen Ertragssatz für eine Folge regelmäßiger Ein- und Auszahlungen zurück.
`Log`	Liefert den Logarithmus einer Zahl.
`MIRR`	Liefert den modifizierten internen Ertragssatz für eine Folge regelmäßiger, mit unterschiedlichen Zinssätzen ausgestatteten Ein- und Auszahlungen.
`NPer`	Errechnet die Anzahl der Zeiträume für eine Annuität bei konstanter Zahlung und festem Zinssatz.

Tabelle 162: Die Funktionen der Kategorie Mathematik

Funktion	Kurzbeschreibung
NPV	Errechnet den Nettobarwert einer Investition bei regelmäßigen Ein- und Auszahlungen und festem Diskontsatz.
Pmt	Errechnet den Auszahlungswert für eine Annuität bei festen Zeiträumen, konstanten Zahlungen und festem Zinssatz.
PPmt	Errechnet den Kapitelanteil eines Zeitraums für eine Annuität bei einer festen Anzahl von Zeiträumen, konstanten Zahlungen und festem Zinssatz.
PV	Errechnet den Barwert für eine Annuität bei konstanter Zahlung, festem Zinssatz und einer festen Laufzeit.
Randomize	Initialisiert den Zufallsgenerator.
Rate	Errechnet den Zinssatz zu einer Annuität bei fester Anzahl von Zeiträumen, konstanten Zahlungen und festem Zinssatz.
Rnd	Liefert eine Zufallszahl mit Dezimalstellen zwischen 0 und 1.
Round	Rundet einen Zahlenwert auf Basis einer Dezimalstelle.
Sgn	Gibt das Vorzeichen eines Werts als Faktor (–10 oder 1) zurück.
Sin	Gibt den Sinus zu einem Winkel im Bogenmaß zurück.
SLN	Errechnet den periodischen Abschreibungswert bei linearer Abschreibung über einen bestimmten Zeitraum.
Sqr	Errechnet die Quadratwurzel einer Zahl.
SYD	Errechnet den Abschreibungswert für eine Periode nach dem Modell der Jahressummengewichtung.
Tan	Liefert den Tangens zu einem Winkel im Bogenmaß.

Tabelle 162: Die Funktionen der Kategorie Mathematik (Forts.)

Prüffunktionen

Funktion	Beschreibung
IsEmpty	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob eine Variable initialisiert wurde.
IsArray	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob eine Variable ein Datenfeld ist.
IsDate	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob ein Ausdruck in ein Datum umgewandelt werden kann.
IsError	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob ein Ausdruck ein Fehlerwert ist.
IsNull	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob ein Ausdruck keine gültigen Daten (Null) enthält.

Tabelle 163: Die Prüffunktionen von Access

Funktion	Beschreibung
IsNumeric	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob ein Ausdruck als Zahl ausgewertet werden kann.
IsObject	Gibt einen Wert vom Typ Boolean zurück, der angibt, ob ein Bezeichner eine Objektvariable darstellt.

Tabelle 163: Die Prüffunktionen von Access (Forts.)

Umwandlungsfunktionen

Funktion	Rückgabetyp	Bereich des Arguments/Ausdruck
CBool	Boolean	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck.
Cbyte	Byte	0 bis 255.
Ccur	Currency	–922.337.203.685.477,5808 bis 922.337.203.685.477,5807.
Cdate	Date	Ein beliebiger gültiger Datumsausdruck.
CDbl	Double	–4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
CDec	Decimal	Für skalierte Ganzzahlen, d.h. Zahlen ohne Dezimalstellen. Für Zahlen mit 28 Dezimalstellen gilt der Bereich: +/–7,9228162514264337593543950335
CInt	Integer	–32.768 bis 32.767; Nachkommastellen werden gerundet.
CLng	Long	–2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet.
CSng	Single	–3,402823E38 bis –1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Cvar	Variant	Numerische Werte im Bereich des Typs Double. Nichtnumerische Werte im Bereich des Typs String.
CStr	String	Rückgabe für CStr hängt vom Argument/Ausdruck ab.

Tabelle 164: Typumwandlungsfunktionen

Farbfunktionen

Für den Einsatz von Farben können Sie in Access die Funktionen `RGB` oder `QBColor` einsetzen.

Mithilfe der Funktion `RGB` können Sie sich eine Farbe zusammenmischen. Als Argumente müssen Sie hierbei drei Werte angeben. Der erste Wert steht für den Rotanteil der Farbe, der zweite für den grünen Anteil und der dritte für den blauen Anteil der gewählten Farbe. Die folgende Tabelle listet einige typische Farbmischungen auf.

Farbe	Rotwert	Grünwert	Blauwert
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 165: Die Standardfarben über die Funktion RGB

Farbnummer	Farbe
0	Schwarz
1	Blau
2	Grün
3	Cyan
4	Rot
5	Magenta
6	Gelb
7	Weiß
8	Grau
9	Hellblau
10	Hellgrün
11	Hellcyan
12	Hellrot
13	Hellmagenta
14	Hellgelb
15	Leuchtendes Weiß

Tabelle 166: Die Standardfarben über die Funktion BQColor

Suchfunktionen

Über die folgenden Methoden können Sie Ihre Datenbestände abfragen.

Methode	Beschreibung
`DFirst`	Ermittelt den ersten Datensatz einer Tabelle/Abfrage, der einem bestimmten Kriterium entspricht.
`DLookup`	Ermittelt den Wert eines Felds einer Tabelle/Abfrage, der einem bestimmten Kriterium entspricht.
`DMax`	Ermittelt den größten Wert eines Felds in einer Tabelle/Abfrage.
`DMin`	Ermittelt den kleinsten Wert eines Felds in einer Tabelle/Abfrage.
`DAvg`	Ermittelt den Mittelwert der Datensätze einer Datensatzgruppe.
`DCount`	Ermittelt die Anzahl der Datensätze einer Datensatzgruppe.
`DSum`	Ermittelt die Summe der Datensätze einer Datensatzgruppe.
`DStDev`	Ermittelt die Standardabweichung einer Gruppe von Werten in einer Datensatzgruppe.
`DVar`	Ermittelt die Varianz einer Gruppe von Werten in einer Datensatzgruppe.

Tabelle 167: Die Suchmethoden von Access

Übersicht der Konstanten

Dieser Teil des Anhangs enthält eine Auswahl der wichtigsten Konstanten von Access sowie Konstanten allgemein einsetzbarer Funktionen.

FileType-Konstanten

Die `FileType`-Konstanten sind dann von Interesse, wenn Sie über das Objekt `FileSearch` bestimmte Dateien suchen möchten.

Konstante	Beschreibung
`msoFileTypeAllFiles`	Alle Dateien
`msoFileTypeCalendarItem`	Kalendereinträge
`msoFileTypeContactItem`	Kontaktordnereinträge
`msoFileTypeCustom`	Benutzerdefinierte Dateien
`msoFileTypeDatabases`	Datenbankdateien
`msoFileTypeDesignerFiles`	Designer-Dateien
`msoFileTypeDocumentImagingFiles`	Bilddokumentdateien
`msoFileTypeExcelWorkbooks`	Excel-Arbeitsmappen
`msoFileTypeJournalItem`	Journalbucheinträge (Outlook)
`msoFileTypeMailItem`	Mails
`msoFileTypeNoteItem`	Notizen aus Outlook
`msoFileTypeOfficeFiles`	Office-Dateien aller Art
`msoFileTypeOutlookItems`	Outlook-Objekte
`msoFileTypePhotoDrawFiles`	Dateien von PhotoDraw
`msoFileTypePowerPointPresentations`	PowerPoint-Präsentationen
`msoFileTypeProjectFiles`	Projektdateien aus MS Project
`msoFileTypePublisherFiles`	Dateien von MS Publisher
`msoFileTypeTaskItem`	Aufgaben aus Outlook
`msoFileTypeTemplates`	Vorlagedateien
`msoFileTypeVisioFiles`	Dateien aus MS Visio
`msoFileTypeWebPages`	Webseiten-Dateien
`msoFileTypeWordDocuments`	Word-Dokumente

Tabelle 168: Die Konstanten der Eigenschaft FileType (FileSearch)

RGB-Konstanten

Über den Einsatz der Funktion `RGB` können Sie sich eine Farbe zusammenmischen. Als Argumente müssen Sie hierbei drei Werte angeben. Der erste Wert steht für den Rotanteil der Farbe, der zweite für den grünen Anteil und der dritte für den blauen Anteil der gewählten Farbe. Die folgende Tabelle listet einige typische Farbmischungen auf.

Farbe	Rotwert	Grünwert	Blauwert
Schwarz	0	0	0
Blau	0	0	255
Grün	0	255	0
Cyan	0	255	255
Rot	255	0	0
Magenta	255	0	255
Gelb	255	255	0
Weiß	255	255	255

Tabelle 169: Die Standardfarben über die Funktion RGB

RunCommand-Konstanten

Die Methode `RunCommand` führt einen eingebauten Menü- oder Symbolleistenbefehl aus. Dabei werden folgende Konstanten verwendet. Der Befehl kann entweder über seine Konstante oder einen festgelegten Wert angewendet werden.

Konstante	Beschreibung	Wert
`acCmdAboutMicrosoftAccess`	Zeigt die Access-Info-Box an.	35
`acCmdAnalyzeTable`	Zeigt den Tabelle-Analyse-Dialog an.	284
`acCmdAnswerWizard`	Zeigt die Direkthilfe an.	235
`acCmdApplyFilterSort`	Wendet den Filter an.	93
`acCmdAppMaximize`	Maximiert das Access-Fenster.	10
`acCmdAppMinimize`	Minimiert das Access-Fenster.	11
`acCmdAppMove`	Verschiebt das Access-Fenster.	12
`acCmdAppRestore`	Stellt das Access-Fenster wieder her.	9
`acCmdAppSize`	Passt das Access-Fenster in der Größe an.	13
`acCmdArrangeIconsAuto`	Ordnet die Symbole automatisch an.	218
`acCmdArrangeIconsByCreated`	Ordnet die Symbole nach dem Erstellungsdatum an.	216

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand

Konstante	Beschreibung	Wert
acCmdArrangeIconsByModified	Ordnet die Symbole nach der letzten Änderung an.	217
acCmdArrangeIconsByName	Ordnet die Symbole nach dem Namen an.	214
acCmdArrangeIconsByType	Ordnet die Symbole nach dem Dateityp an.	215
acCmdAutoKorrect	Zeigt den Autokorrektur-Dialog an.	261
acCmdClearAll	Löscht alle Tabellen aus dem Beziehungsfenster.	146
acCmdClearAllBreakPoints	Löscht alle Haltepunkte.	132
acCmdClose	Schließt das Fenster mit dem aktuellen Fokus.	58
acCmdCloseWindow	Schließt das aktuelle Fenster.	186
acCmdColumnWidth	Zeigt den Dialog zur Spaltenbreitendefinition an.	117
acCmdCompactDatabase	Datenbank komprimieren.	4
acCmdControlWizardToggle	Steuerelement-Assistent ein- und ausschalten.	197
acCmdCopy	Kopiert den markierten Bereich.	190
acCmdCreateRelationship	Erstellt Beziehung.	150
acCmdCreateShortcut	Erstellt Verknüpfung.	219
acCmdCut	Schneidet den markierten Bereich aus.	189
acCmdDataAccessPageBrowse	Listet die Datenzugriffsseiten auf.	344
acCmdDatabaseProperties	Zeigt die Datenbankeigenschaften an.	256
acCmdDataEntry	Schaltet auf die Dateneingabe um.	78
acCmdDatasheetView	Wechselt in die Datenblattansicht.	282
acCmdDateAndTime	Zeigt den Datums- und Zeitdialog an.	226
acCmdDebugWindow	Zeigt das Direktfenster an.	123
acCmdDelete	Löscht das ausgewählte Objekt.	337
acCmdDeleteQueryColumn	Löscht Spalten in einer Abfrage.	81
acCmdDeleteRecord	Löscht Datensatz.	223
acCmdDeleteRows	Löscht Zeilen.	188
acCmdDeleteTableColumn	Löscht Spalten in einer Tabelle.	271
acCmdDesignView	Zeigt die Entwurfsansicht an.	183
acCmdDocMaximize	Maximiert das Datenbankfenster.	15

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdDocMinimize	Minimiert das Datenbankfenster.	60
acCmdDocMove	Verschiebt das Datenbankfenster.	16
acCmdDocSize	Legt die Größe des Fensters fest.	17
acCmdDuplicate	Dupliziert das markierte Objekt.	34
acCmdEditHyperlink	Bearbeitet einen Hyperlink.	325
acCmdEditRelationship	Bearbeitet eine Beziehung.	151
acCmdEnd	Unterbricht die aktuelle Aktion.	198
acCmdExit	Beendet Access.	3
acCmdFavoritesAddTo	Fügt einen Eintrag zu den Favoriten hinzu.	299
acCmdFavoritesOpen	Öffnet die Favoriten.	298
acCmdFieldList	Zeigt bzw. blendet die Feldliste an/aus.	42
acCmdFilterByForm	Stellt einen formularbasierten Filter ein.	207
acCmdFilterBySelection	Stellt einen auswahlbasierten Filter ein.	208
acCmdFind	Zeigt das Suchen-Fenster an.	30
acCmdFindNext	Sucht das nächste Vorkommen.	341
acCmdFindPrevious	Sucht das vorhergehende Vorkommen.	120
acCmdFitToWindow	Passt die Fensteransicht an.	245
acCmdFont	Zeigt den Schriften-Dialog an.	19
acCmdFormHdrFtr	Zeigt den Formularkopf und -fuß an bzw. blendet ihn aus.	36
acCmdFormView	Wechselt in die Formularansicht.	281
acCmdFreezeColumn	Fixiert Spalten.	105
acCmdGoBack	Zurück zur vorherigen Seite.	294
acCmdGoContinue	Führt den Code fort.	127
acCmdGoForward	Vor zur nächsten Seite.	295
acCmdHideColumns	Blendet Spalten aus.	79
acCmdHideTable	Blendet Tabellen aus.	147
acCmdImport	Importiert externe Daten.	257
acCmdInsertActiveXControl	ActiveX-Control einfügen.	258
acCmdInsertChart	Fügt ein Diagramm ein.	293
acCmdInsertFileIntoModule	Fügt eine Datei in ein Modul ein.	118
acCmdInsertHyperlink	Fügt einen Hyperlink ein.	259
acCmdInsertObject	Fügt ein Objekt ein.	33

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdInsertPicture	Fügt eine Grafik ein.	222
acCmdInsertProcedure	Fügt eine Prozedur ein.	262
acCmdInsertQueryColumn	Fügt eine Spalte in eine Abfrage ein.	82
acCmdInsertRows	Fügt eine Zeile ein.	187
acCmdInsertTableColumn	Fügt eine Tabellenspalte ein.	272
acCmdJoinProperties	Zeigt den Verknüpfungseigenschaften-Dialog an.	72
acCmdLastPosition	Steuert die letzte Position in einem Modul an.	339
acCmdLayoutPreview	Zeigt die Layout-Vorschau an.	141
acCmdLineUpIcons	Ordnet die Symbole neu an.	213
acCmdLinkTables	Verknüpft Tabellen.	102
acCmdListConstants	Zeigt die Konstanten an.	303
acCmdMicrosoftAccessHelpTopics	Zeigt das Access-Hilfefenster an.	100
acCmdNewDatabase	Legt eine neue Datenbank an.	26
acCmdNewObjectAutoForm	Erstellt ein AutoFormular.	193
acCmdNewObjectAutoReport	Erstellt einen AutoBericht.	194
acCmdNewObjectClassModule	Erstellt ein neues Klassenmodul.	140
acCmdNewObjectDataAccessPage	Erstellt eine neue Datenzugriffsseite.	346
acCmdNewObjectDiagramm	Erstellt ein neues Diagramm in der Entwurfsansicht.	352
acCmdNewObjectForm	Startet den Formularassistenten.	136
acCmdNewObjectModule	Erstellt ein neues Modul.	139
acCmdNewObjectQuery	Startet den Abfrage-Assistenten.	135
acCmdNewObjectReport	Startet den Berichts-Assistenten.	137
acCmdNewObjectTable	Startet den Tabellen-Assistenten.	134
acCmdObjectBrowser	Öffnet den Objektdialog.	200
acCmdOpenDatabase	Zeigt den Datenbank-Öffnen Dialog an.	25
acCmdOpenHyperlink	Öffnet einen Hyperlink.	326
acCmdOpenNewHyperlink	Zeigt den Hyperlink in einem neuen Fenster an.	327
acCmdOpenTable	Öffnet eine Tabelle.	221
acCmdOptions	Zeigt das Optionsfenster an.	49
acCmdOutputToExcel	Exportiert ein Objekt als Excel-Tabelle.	175
acCmdOutputToText	Exportiert ein Modul in einer Textdatei.	177

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdPageSetup	Zeigt den Dialog SEITE EINRICHTEN an.	32
acCmdPaste	Fügt Inhalt aus der Zwischenablage ein.	191
acCmdPasteSpecial	Zeigt den Dialog INHALTE EINFÜGEN an.	64
acCmdPreviewOnePage	Zeigt die Seitenansicht in einer Seite an.	246
acCmdPreviewTwoPages	Zeigt die Seitenansicht in zwei Seiten an.	247
acCmdPrintPreview	Zeigt die Seitenansicht an.	54
acCmdPrint	Zeigt den Dialog DRUCKEN an.	340
acCmdProperties	Zeigt das Eigenschaftsfenster an.	287
acCmdQuickPrint	Druckt direkt, ohne den Dialog DRUCKEN anzuzeigen.	278
acCmdRecordsGoToFirst	Gehe zu – erster Satz.	67
acCmdRecordsGoToLast	Gehe zu – letzter Satz.	68
acCmdRecordsGoToNew	Gehe zu – neuem Satz.	28
acCmdRecordsGoToNext	Gehe zu – nächstem Satz.	65
acCmdRecordsGotoPrevious	Gehe zu – vorherigem Satz.	66
acCmdRedo	Letzten Befehl wiederholen.	199
acCmdReferences	Zeigt den Dialog VERWEISE an.	260
acCmdRefresh	Die Anzeige wird aktualisiert.	18
acCmdRelationships	Öffnet das Fenster BEZIEHUNGEN.	133
acCmdRemoveFilterSort	Entfernt Filter und Sortierung.	144
acCmdRemoveTable	Entfernt Tabelle.	84
acCmdRename	Benennt ein Objekt um.	143
acCmdRepairDatabase	Repariert Datenbank.	6
acCmdReplace	Öffnet den Dialog ERSETZEN.	29
acCmdReset	Setzt den laufenden Code zurück.	124
acCmdRowHeight	Zeigt den Dialog ZEILENHÖHE an.	116
acCmdRun	Abfrage ausführen.	181
acCmdRunMacro	Der Dialog MAKRO AUSFÜHREN wird angezeigt.	31
acCmdRunOpenMacro	Makro wird ausgeführt.	338
acCmdSave	Speichert das aktuelle Objekt.	20
acCmdSaveAs	Der Dialog SPEICHERN UNTER/ EXPORTIEREN wird angezeigt.	21
acCmdSaveAsASP	Objekt wird als ASP-Datei gespeichert.	323

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdSaveAsReport	Der Dialog ALS BERICHT SPEICHERN wird angezeigt.	142
acCmdSaveModuleAsText	Der Dialog SPEICHERN UNTER wird für das Modul angezeigt.	119
acCmdSaveRecord	Datensatz wird gespeichert.	97
acCmdSelectAll	Alles wird ausgewählt.	333
acCmdSelectAllRecords	Alle Datensätze werden ausgewählt.	109
acCmdSelectDataAccessPage	Datenzugriffsseite wird ausgewählt.	347
acCmdSelectForm	Formular wird ausgewählt.	40
acCmdSelectRecord	Datensatz wird markiert.	50
acCmdSelectReport	Bericht wird ausgewählt.	319
acCmdSend	Sendet das markierte Objekt per E-Mail.	173
acCmdSetDatabasePasswort	Legt das Datenbankkennwort fest.	275
acCmdShowAllRelationships	Zeigt alle Beziehungen an.	149
acCmdShowTable	Zeigt den Dialog TABELLE HINZUFÜGEN an.	185
acCmdSingleStep	Einzelschrittmodus in einem Makro durchführen.	88
acCmdSizeToFit	Größe an Textgröße anpassen.	59
acCmdSizeToGrid	Größe an Raster anpassen.	48
acCmdSizeToNarrowest	Größe wird auf Basis des schmalsten Elements angepasst.	155
acCmdSizeToShortest	Größe wird auf Basis des niedrigsten Elements angepasst.	260
acCmdSnapToGrid	Elemente werden am Raster neu ausgerichtet.	62
acCmdSortAscending	Sortierung wird aufsteigend durchgeführt.	163
acCmdSortDescending	Sortierung wird absteigend durchgeführt.	164
acCmdSortingAndGrouping	Dialog SORTIEREN UND GRUPPIEREN wird angezeigt.	51
acCmdSQLView	Wechselt zur SQL-Ansicht.	184
acCmdStartupProperties	Zeigt den Dialog START an.	224
acCmdStepInto	Führt einen Einzelschritt in die nächste Prozedurzeile durch.	342

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdStepOver	Lässt einen Prozedurschritt aus.	128
acCmdStepToCursor	Führt eine Prozedur bis zur Cursorposition aus.	204
acCmdToggleBreakpoint	Schaltet einen Haltepunkt ein/aus.	131
acCmdToggleFilter	Wendet einen Filter an bzw. hebt ihn auf.	220
acCmdToolbarsCustomize	Zeigt den Dialog ANPASSEN an.	165
acCmdUndo	Widerruft die letzte Aktion.	292
acCmdUserAndGroupAccounts	Zeigt den Benutzer- und Gruppendialog an.	104
acCmdViewCode	Öffnet das Code-Fenster.	170
acCmdViewDataAccessPages	Zeigt die Datenzugriffsseiten an.	349
acCmdViewDetails	Zeigt Details im Datenbankfenster an.	210
acCmdViewGrid	Blendet das Raster ein bzw. aus.	63
acCmdForms	Zeigt die Formulare im Datenbankfenster an.	112
acCmdViewLargeIcons	Zeigt große Symbole im Datenbankfenster an.	209
acCmdViewList	Zeigt die Objekte als Liste im Datenbankfenster an.	212
acCmdViewMacros	Zeigt die Makros im Datenbankfenster an.	114
acCmdViewModules	Zeigt die Module im Datenbankfenster an.	115
acCmdViewQueries	Zeigt die Abfragen im Datenbankfenster an.	111
acCmdViewReports	Zeigt die Berichte im Datenbankfenster an.	113
acCmdViewRuler	Blendet das Lineal ein bzw. aus.	61
acCmdViewShowPaneGrid	Zeigt eine Tabelle in der Entwurfsansicht an.	359
acCmdViewSmallIcons	Zeigt kleine Symbole im Datenbankfenster an.	211
acCmdViewStoredProcedures	Zeigt die gespeicherten Prozeduren im Datenbankfenster an.	355
acCmdViewTables	Zeigt die Tabellen in der Datenbankansicht an.	110
acCmdViewToolbox	Blendet die Toolbox ein bzw. aus.	85

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdWindowsArrangeIcons	Symbole werden angeordnet.	24
acCmdWindowCascade	Fenster werden überlappend angezeigt.	22
acCmdWindowHide	Aktives Fenster wird ausgeblendet.	2
acCmdWindowUnhide	Der Dialog FENSTER EINBLENDEN wird angezeigt.	1
acCmdZoom10	Der Zoom wird auf 10% eingestellt.	244
acCmdZoom100	Der Zoom wird auf 100% (1:1) eingestellt.	240
acCmdZoomBox	Der Dialog ZOOM wird angezeigt.	179

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Dir-Konstanten

Sicher kennen Sie noch den alten DOS-Befehl `Dir`, über den Sie sich damals, aber auch noch heute, den Inhalt Ihrer Festplatte anzeigen lassen konnten. Mit diesem Befehl lassen sich bestimmte Konstanten einsetzen, die Sie in der nachfolgenden Tabelle einsehen können.

Konstante	Wert	Beschreibung
VbNormal	0	(Voreinstellung) Dateien ohne Attribute.
VbReadOnly	1	Schreibgeschützte Dateien, zusätzlich zu Dateien ohne Attribute.
VbHidden	2	Versteckte Dateien, zusätzlich zu Dateien ohne Attribute.
VbSystem	4	Systemdatei, zusätzlich zu Dateien ohne Attribute. Beim Macintosh nicht verfügbar.
VbVolume	8	Datenträgerbezeichnung. Falls andere Attribute angegeben wurden, wird `VbVolume` ignoriert. Beim Macintosh nicht verfügbar.
VbDirectory	16	Verzeichnis oder Ordner, zusätzlich zu Dateien ohne Attribute.

Tabelle 171: Die Konstanten der Funktion Dir

Shell-Konstanten

Mithilfe der Funktion `Shell` können Sie jedes ausführbare Programm aufrufen. Über die Shell-Konstanten legen Sie dabei fest, ob das Programm im Hintergrund bleiben bzw. minimiert oder maximiert angezeigt werden soll.

Die Funktion `Shell` hat folgende Syntax:

```
Shell(pathname[,windowstyle])
```

Im Argument `pathname` geben Sie den Pfad sowie den Dateinamen der ausführbaren Datei an, die Sie starten möchten.

Das Argument `WindowsStyle` legt den Stil des Fensters fest, in dem das Programm ausgeführt werden soll. Dabei stehen Ihnen die Konstanten aus der folgenden Tabelle zur Verfügung.

Konstante	Wert	Beschreibung
`VbHide`	0	Das Fenster ist ausgeblendet und das ausgeblendete Fenster erhält den Fokus.
`VbNormalFocus`	1	Das Fenster hat den Fokus und die ursprüngliche Größe und Position werden wiederhergestellt.
`VbMinimizedFocus`	2	Das Fenster wird als Symbol mit Fokus angezeigt.
`VbMaximizedFocus`	3	Das Fenster wird maximiert mit Fokus angezeigt.
`VbNormalNoFocus`	4	Die zuletzt verwendete Größe und Position des Fensters wird wiederhergestellt. Das momentan aktive Fenster bleibt aktiv.
`VbMinimizedNoFocus`	6	Das Fenster wird als Symbol angezeigt. Das momentan aktive Fenster bleibt aktiv.

Tabelle 172: Die Konstanten der Funktion Shell

StrConv-Konstanten

Mithilfe der Funktion `StrConv` und der dazugehörigen Konstanten können Sie Texte automatisch anpassen. Sie haben dabei unter anderem die Möglichkeit, Texte in Groß- bzw. Kleinbuchstaben umzuwandeln.

Entnehmen Sie der folgenden Tabelle die gängigsten Konstanten der Funktion `StrConv`.

Konstante	Wert	Beschreibung
`VbUpperCase`	1	Wandelt die Zeichenfolge in Großbuchstaben um.
`VbLowerCase`	2	Wandelt die Zeichenfolge in Kleinbuchstaben um.
`VbProperCase`	3	Wandelt den ersten Buchstaben jedes Worts innerhalb der Zeichenfolge in einen Großbuchstaben um.

Tabelle 173: Die Konstanten der Funktion StrConv

Var-Type-Konstanten

Über die Funktion `VarType` und deren Konstanten können Sie den Datentyp einer Variablen prüfen. Dies ist mitunter wichtig, wenn Sie später Variablentypen umwandeln müssen.

Die Funktion VarType hat folgende Syntax:

```
VarType(VarName)
```

Das Argument VarName liefert einen Wert, dessen Bedeutung Sie der folgenden Tabelle entnehmen können.

Konstante	Wert	Beschreibung
VbEmpty	0	Empty (nicht initialisiert)
VbNull	1	Null (keine gültigen Daten)
VbInteger	2	Ganzzahl (Integer)
VbLong	3	Ganzzahl (Long)
VbSingle	4	Fließkommazahl einfacher Genauigkeit
VbDouble	5	Fließkommazahl doppelter Genauigkeit
VbCurrency	6	Währungsbetrag (Currency)
VbDate	7	Datumswert (Date)
VbString	8	Zeichenfolge (String)
vbObject	9	Objekt
vbError	10	Fehlerwert
vbBoolean	11	Boolescher Wert (Boolean)
vbVariant	12	Variant (nur bei Datenfeldern mit Variant-Werten)
vbDataObject	13	Ein Datenzugriffsobjekt
VbDecimal	14	Dezimalwert
VbByte	17	Byte-Wert
VbArray	8192	Datenfeld (Array)

Tabelle 174: Die Konstanten der Funktion VarType

File Input/Output-Konstanten

Mithilfe dieser Konstanten bestimmen Sie den Zugriff auf eine Datei. Dabei können Sie eine Datei mit Leserecht öffnen. Sie haben dann keine Chance, diese Datei zu verändern. Eine weitere Möglichkeit ist es, eine Datei zu öffnen und dafür zu sorgen, dass diese neu angelegt wird, sofern sie schon existiert. Die dritte Möglichkeit ist, in eine Datei hineinzuschreiben und dabei die bisherigen Eingaben beizubehalten, also den neuen Text einfach unten anzuhängen. Diese File Input/Output-Konstanten werden bevorzugt in der Verarbeitung von Textdateien angewendet.

Entnehmen Sie die einzelnen Möglichkeiten des Dateizugriffs der folgenden Tabelle.

Konstante	Wert	Beschreibung
ForReading	1	Öffnet eine Datei, die nur gelesen werden kann. Sie können nicht in diese Datei schreiben.
ForWriting	2	Öffnet eine Datei zum Schreiben. Wenn es eine Datei mit dem gleichen Namen gibt, wird der frühere Inhalt überschrieben.
ForAppending	8	Öffnet eine Datei und schreibt an das Ende dieser.

Tabelle 175: Die Konstanten für den Dateizugriff bei Textdateien

Datumsformat-Konstanten

Für das Anzeigen von Datums- und Zeitangaben stehen Ihnen fertige Systemkonstanten zur Verfügung, die die Formatierung des Datums bzw. des Zeitwerts für Sie übernehmen. Diese Datums-/Zeitkonstanten werden beispielsweise im Zusammenspiel mit der Funktion `FormatDatTime` verwendet.

```
FormatDateTime(Datum[,BenanntesFormat])
```

Im Argument `Datum` übergeben Sie der Funktion einen Datumswert.

Im Argument `BenanntesFormat` wählen Sie eine der folgenden Datums-/Zeitkonstanten (siehe Tabelle 176).

Konstante	Wert	Beschreibung
vbGeneralDate	0	Zeigt ein Datum und/oder eine Uhrzeit an. Wenn es ein Datum gibt, wird es in Kurzform angezeigt. Wenn es eine Uhrzeit gibt, wird sie im langen Format angezeigt. Falls vorhanden, werden beide Teile angezeigt.
vbLongDate	1	Zeigt ein Datum im langen Datumsformat an, das in den Ländereinstellungen des Computers festgelegt ist.
VbShortDate	2	Zeigt ein Datum im kurzen Datumsformat an, das in den Ländereinstellungen des Computers festgelegt ist.
vbLongTime	3	Zeigt eine Uhrzeit in dem Zeitformat an, das in den Ländereinstellungen des Computers festgelegt ist.
vbShortTime	4	Zeigt eine Uhrzeit im 24-Stunden-Format (hh:mm) an.

Tabelle 176: Die Datumskonstanten

Schaltflächen auswerten

Die Methode `Msgbox` sowie die Methode `Inputbox` haben folgende Schaltflächenkonstanten:

Konstante oder Wert	Beschreibung
`vbOK` oder `1`	Die Schaltfläche OK wurde angeklickt.
`vbCancel` oder `2`	Die Schaltfläche ABBRECHEN wurde angeklickt.
`vbAbort` oder `3`	Die Schaltfläche ABBRUCH wurde angeklickt.
`vbRetry` oder `4`	Die Schaltfläche WIEDERHOLEN wurde angeklickt.
`vbIgnore` oder `5`	Die Schaltfläche IGNORIEREN wurde angeklickt.
`vbYes` oder `6`	Die Schaltfläche JA wurde angeklickt.
`vbNo` oder `7`	Die Schaltfläche NEIN wurde angeklickt.

Tabelle 177: Rückgabewerte für Schaltflächen

Stichwortverzeichnis

D

S