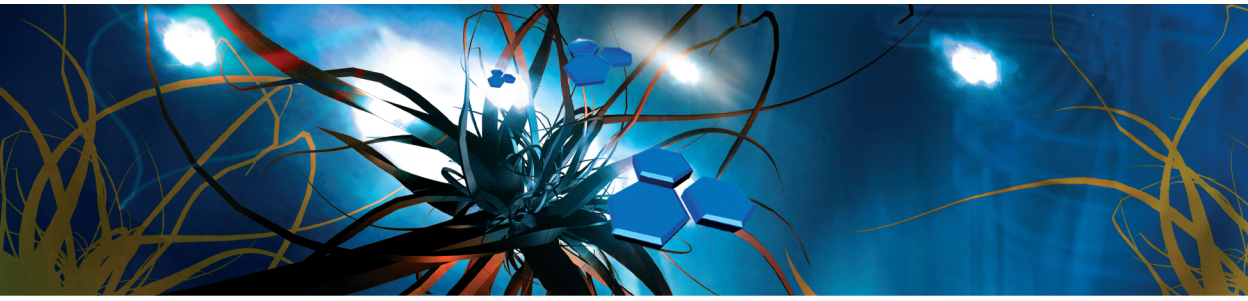
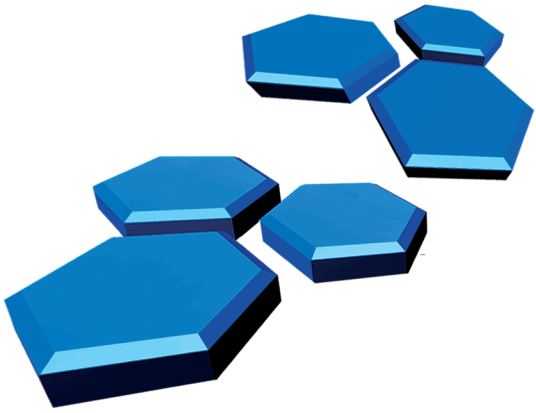
Premium Codebook

BERND HELD

Das Access-VBA Codebook





ADDISON-WESLEY

Das Access-VBA Codebook

Unser Online-Tipp
für noch mehr Wissen …

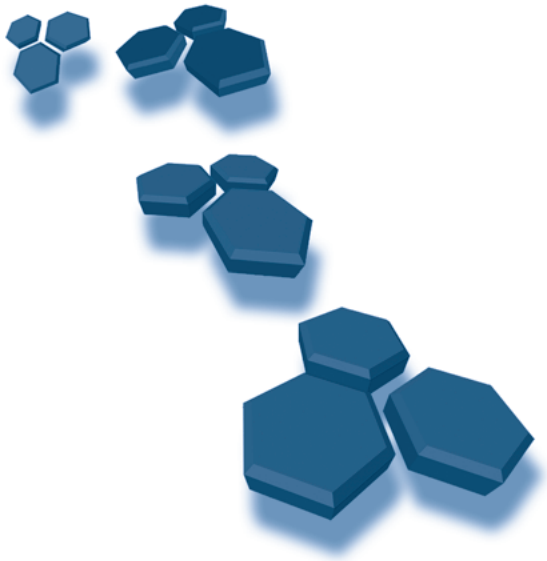
informit.de

Aktuelles Fachwissen rund um die Uhr
– zum Probelesen, Downloaden oder
auch auf Papier.

www.informit.de

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.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt. Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ®-Symbol in diesem Buch nicht verwendet.

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt. Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

10 09 08

ISBN 978-3-8273-2394-1

© 2008 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten

Korrektorat: Martina Gradias
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Fachlektorat: Stefan Leibing
Herstellung: Elisabeth Prümm, epruemm@pearson.de
Satz: Kösel, Krugzell (www.KoeselBuch.de)
Umschlaggestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Druck und Verarbeitung: Kösel, Krugzell (www.KoeselBuch.de)

Printed in Germany

Inhaltsverzeichnis

Vorwort 19

Über den Autor 21

Teil I Rezepte 23

Allgemeine VBA-Funktionen 25

Die Datums- und Zeitfunktionen 25
1 Zeichenfolge in gültiges Datum umwandeln 25
2 Systemdatum abrufen 26
3 Endtermine errechnen 27
4 Kalenderwoche ermitteln 29
5 Datumsdifferenzen errechnen 31
6 Einen Kalender erstellen 32
7 Datumswerte erkennen und umsetzen 34
8 Zeitwerte erkennen und umsetzen 34
9 Datum zerlegen 35
10 Die richtige Datumstabelle öffnen 37
11 Änderungsdatum einer Datenbank ermitteln 37
12 Datumsangaben formatiert ausgeben 41
13 Zeit zerlegen 42
14 Monatsnamen ermitteln 43
15 Wochentag ermitteln 45

Die Textfunktionen 47
16 Buchstaben aus Zeichenfolgen entfernen 47
17 Zahlen aus Zeichenfolgen entfernen 48
18 Zahlungsart definieren 49
19 Aus Zahlen Buchstaben machen 50
20 Zeichenfolgen vergleichen 51
21 Den Pfad- und Dateinamen einer Datenbank zerlegen 54
22 Zeichenfolgen splitten und zusammenführen 56
23 In Groß- und Kleinschreibung umwandeln 58
24 Access-Version feststellen 60
25 Text Zeichen für Zeichen zerlegen 62
26 Datei-Endung prüfen 63
27 Zeichen ersetzen 65
28 Zeichenfolge mit Leerzeichen auffüllen 66
29 Zahlen in Texte umwandeln 67
30 Zeichenfolgen miteinander vergleichen 68
31 Zeichenfolgen spiegeln 69
32 Texte vervollständigen 70
33 Zuordnungen treffen 71
34 Führende Leerzeichen entfernen 72

Die Dateifunktionen **74**
35 Aktuelles Verzeichnis wechseln 74
36 Aktuelles Laufwerk wechseln 75
37 Aktuelles Verzeichnis ermitteln 75
38 Dateien und Verzeichnisse auflisten 76
39 Verzeichnis anlegen 78
40 Verzeichnis löschen 79
41 Datei löschen 79
42 Datei kopieren 80
43 Bilder umbenennen und nummerieren 81
44 Datei umbenennen 84
45 Dateiexistenz prüfen 85
46 Betriebssystem-Umgebungsvariablen auslesen 86
47 Textdatei lesen 88
48 Erste Zeile aus Textdatei lesen 89
49 Nur wenige Zeichen aus Textdatei lesen 90
50 Textdatei Zeile für Zeile auslesen 91
51 Textdatei schreiben 92
52 Textdateien im Batch erstellen 95
53 Zugriffsmodus einer Datei feststellen und setzen 96
54 Externes Programm starten 98

Mathematische Funktionen **102**
55 Absolutwert ausgeben 102
56 Vorzeichen auslesen 103
57 Abschreibung errechnen (degressiv) 104
58 Abschreibung errechnen (linear) 105
59 Ganzzahligen Wert ermitteln 106
60 Zahlen runden 107
61 Zinssatz errechnen 109
62 Ansparungsbetrag errechnen 110
63 Zufallszahlen erzeugen 112

Prüffunktionen **114**
64 Eingabe prüfen 114
65 Datenfeld-Check 115
66 Tabellencheck vornehmen 116

Umwandlungsfunktionen **117**
67 Punkte entfernen 117
68 Datumswerte umwandeln 118
69 Nachkommastellen entfernen 119
70 Zahl aus Zeichenfolge extrahieren 120
71 Zahlen in Texte umwandeln 121

Weitere Funktionen **123**

API-Funktionen **123**
72 Ermittlung des CD-ROM-Laufwerks 124
73 Den Namen des Anwenders ermitteln 126
74 Den Computernamen ermitteln 127

75 Bedienung des CD-ROM-Laufwerks 128
76 Die Bildschirmauflösung ermitteln 129
77 Ist ein externes Programm gestartet? 130
78 Externes Programm aufrufen 131
79 Wie lange läuft ein externes Programm? 132
80 Eine Wartezeit erzeugen 134
81 Verzeichnisse erstellen 134
82 Verzeichnis löschen 135
83 Verzeichnisbaum anzeigen und auswerten 136
84 Windows-Version ermitteln 138
85 Windows-Verzeichnis ermitteln 140
86 Windows-Systemverzeichnis ermitteln 141
87 Das temporäre Verzeichnis ermitteln 142
88 Das aktuelle Verzeichnis ermitteln 143
89 Windows-Info-Dialog anzeigen 144
90 Access-Verzeichnis ermitteln 145
91 Standardverzeichnis festlegen 145
92 Dateityp und Anwendung ermitteln 146
93 Kurze Pfadnamen ermitteln 147
94 Texte mit API-Funktionen konvertieren 148
95 Zwischenablage löschen 149
96 Soundkarte checken 149
97 Sounds per API-Funktion ausgeben 150
98 PC piepsen lassen 151
99 Tasten abfangen 151
100 Dateien suchen 152
101 Dateiinformationen auslesen 153
102 Internetverbindung aktiv? 155
103 Cursorposition in Pixel angeben 156

Eigene Funktionen schreiben 157
104 Stückzahl pro Stunde errechnen 157
105 Stunden in Minuten umrechnen 158
106 Minuten in Stunden umrechnen 159
107 Normalzeit in Industriezeit umrechnen 159
108 Industriezeit in Normalzeit umwandeln 160
109 Schaltjahr oder nicht? 160
110 Stundengeschwindigkeit errechnen 161
111 Durchschnittlichen Benzinverbrauch ermitteln 161
112 Das Idealgewicht bestimmen 162
113 Fahrleistung pro Tag errechnen 163
114 Berechnung des Bremswegs 164
115 Verzeichnis überprüfen 165
116 Datenbank überprüfen 166
117 Datenbankstatus überprüfen 167
118 Objektstatus erkennen 168
119 Dateien zählen 169
120 Initialen erstellen 171
121 Position der ersten Zahl erkennen 172

122 Position des ersten Buchstabens erkennen 173
123 Sonderzeichen entfernen 174
124 Numerische Zeichen in Zeichenfolge zählen 175
125 Buchstaben in Zeichenfolge zählen 176
126 Römische Zahlen umwandeln 177
127 Arabische Zahlen umwandeln 179
128 Auf Dokumenteigenschaften zugreifen 180
129 Den letzten Tag im Monat ermitteln 181

Die Access-Objekte 183

Das AccessObject-Objekt 183
130 Alle Module auflisten 183
131 Alle Tabellen auflisten 184
132 Alle Abfragen auflisten 185

Die Application-Objekte 186
133 Name der aktuellen Datenbank abfragen 186
134 Pfad der aktuellen Datenbank abfragen 187
135 Pfad der Anwendung abfragen 188
136 Access-Version feststellen 188
137 Aktuellen Anwendernamen abfragen 190
138 Aktuelle Datenbank schließen 190
139 Applikation beenden 190
140 Drucker auflisten 191

Das Control-Objekt 192

Das DoCmd-Objekt 193
141 Filter in Tabellen setzen 193
142 Objekte umbenennen 195
143 Objekte kopieren 196
144 Objekte löschen 197
145 Objekte exportieren 198
146 Objekte drucken 200
147 Objekte versenden 201
148 Feld im Formular aktivieren 205
149 Datensatz in Tabelle aktivieren 206
150 Menübefehle ausführen 208
151 Symbolleisten ein- und ausblenden 209
152 SQL-Abfragen ausführen 210
153 Fenstergröße festlegen 211
154 Datenbankinhalte transferieren 212
155 Tabellen transferieren 215
156 Texte exportieren und importieren 216

Das Reference-Objekt 224
157 Bibliotheken auslesen 224

Das Balloon-Objekt 225
158 Office-Assistenten einsetzen 226

Das FileSystem-Objekt **231**
159 Datei kopieren 231
160 Datei verschieben 232
161 Datei löschen 234
162 Dateiexistenz prüfen 235
163 Dateiinfo abfragen 236
164 Verzeichnis anlegen 237
165 Verzeichnis löschen 237
166 Ordnerexistenz prüfen 238
167 Ordnergröße ermitteln 239
168 Ordner auflisten 240
169 Dateien auflisten 242
170 Laufwerke auslesen 243
171 Weitere Laufwerkseigenschaften 245
172 Textdatei einlesen 247
173 Textdatei schreiben 248

Tabellen programmieren **251**

Einfache Aufgaben mit DoCmd **251**
174 Tabellen öffnen 251
175 Datensatz finden 252
176 Datensatz auswählen 254
177 Tabellen verknüpfen 256
178 Tabellen im HTML-/XML-Format speichern 257

Komplexere Aufgaben mit ADO/DAO & SQL **260**
179 Datenbank öffnen und schließen (ADO) 260
180 Tabelle öffnen und Dump ziehen (ADO) 262
181 Eine bestimmte Anzahl von Sätzen extrahieren 264
182 Nur bestimmte Felder einer Tabelle ausgeben (ADO) 265
183 Tabelle öffnen und Daten ausgeben (DAO) 267
184 Mehrere Datensätze in ein Array einlesen (ADO) 268
185 Datensätze in Datenfeld einlesen (DAO) 270
186 SQL-Abfragen ausführen (ADO) 271
187 Unikatliste aus Tabelle erstellen (ADO) 273
188 Unikatliste aus Tabelle erstellen (DAO) 275
189 Datensätze sortieren (ADO) 277
190 Datensätze sortieren (DAO) 279
191 Tabellen zusammenfassen (ADO) 281
192 Verknüpfte Tabellen abfragen (ADO) 282
193 Verknüpfte Tabellen abfragen (DAO) 285
194 Minimal- und Maximalwerte ermitteln (ADO) 286
195 Minimal- und Maximalwerte ermitteln (DAO) 293
196 Daten gruppieren (ADO) 294
197 Datensätze in einer Tabelle finden (ADO) 296
198 Datensätze in einer Tabelle finden (DAO) 300
199 Datensätze verändern (ADO) 304
200 Datensätze verändern (DAO) 306
201 Währungsumrechnungen vornehmen (ADO) 308

202 Währungsumrechnungen vornehmen (DAO) 310
203 Datensätze filtern (ADO) 311
204 Datensätze filtern (DAO) 313
205 Datensätze zählen (ADO) 315
206 Datensätze zählen (DAO) 318
207 Datensätze summieren 320
208 Datensätze clonen (ADO) 321
209 Datensätze clonen (DAO) 323
210 Lesezeichen setzen (ADO) 324
211 Datensätze löschen (ADO) 326
212 Datensätze löschen (DAO) 327
213 Datensätze hinzufügen (ADO) 329
214 Datensätze hinzufügen (DAO) 333

Der Einsatz von ADOX und DAO 334
215 Tabellennamen ermitteln 335
216 Tabelleneigenschaften auslesen 336
217 Tabellenstrukturen auslesen 337
218 Tabelleneigenschaften abfragen 339
219 Tabellen erstellen 341
220 Tabellen verknüpfen 345
221 Tabellen löschen 347
222 Tabellenfeld anhängen (DAO) 347

Verlinken und Verknüpfen (DAO) 349
223 Verknüpfte Tabellen aktualisieren 350
224 Verknüpfungsadresse anpassen 351

Abfragen programmieren 353

Die Abfragetypen von Access 353

Abfragen mithilfe von DoCmd 354
225 Abfragen starten mit OpenQuery 354
226 Abfragen erstellen mit RunSQL 356
227 UPDATE (Aktualisierungsabfrage) 356
228 INSERT INTO (Anfügeabfrage) 358
229 DELETE (Löschabfrage) 361
230 SELECT INTO (Tabellenerstellungsabfrage) 363
231 INNER JOIN (Vergleichsabfrage) 364
232 UNION (Auswahlabfrage) 368
233 Abfragen mit Berechnungen ausführen 371
234 ALTER (Tabellenänderungsabfrage) 373
235 CREATE TABLE (Tabelle erstellen) 375
236 CREATE INDEX (Index definieren) 377
237 DROP INDEX (Index entfernen) 378
238 DROP TABLE (Tabelle löschen) 379

Eigene Abfragen 380
239 Abfrage erstellen und speichern (ADO) 380
240 Abfrage erstellen und speichern (DAO) 381

241 Abfragen über das Katalog-Objekt realisieren 383
242 Normale Abfragen erstellen 383
243 Parameterabfragen erstellen (ADOX) 387
244 Parameterabfragen erstellen (DAO) 390

Steuerelemente, Dialoge und Formulare 397

Das Meldungsfenster MsgBox 397

Der Eingabedialog Inputbox 400

Die FileDialog Standarddialoge 401

Sonstige Dialoge 405

Formulare 406
245 Die Steuerelemente für Formulare 409
246 Formulare auslesen 412
247 Formulare öffnen und schließen 416
248 Aktivierreihenfolge anpassen 419
249 Formularhintergrund einstellen 420
250 Formular ohne Navigationsschaltflächen aufrufen 421

Textfelder 423
251 Hintergrund und Schriftfarbe anpassen 423
252 Startposition des Cursors festhalten 425
253 Auswahl aufheben 425
254 Alte Werte festhalten 426
255 QuickInfo hinzufügen 428
256 Textfelder sperren 429
257 Textfelder ein- und ausblenden 430
258 Textfelder begrenzen 432
259 Textfelder überwachen 432
260 Textfelder initialisieren 434
261 Textfelder prüfen 435

Listenfelder 435
262 Listenfeld füllen 436
263 Listenfeldauswahl verarbeiten 438
264 Listenfeldeinträge zählen 439
265 Aktuellen Listenfeldeintrag löschen 439
266 Alle markierten Listenfeldeinträge löschen 441
267 Alle Listenfeldeinträge aus-/abwählen 442
268 Mehrspaltiges Listenfeld füllen 443
269 Auf einzelne Spalten zugreifen 444

Kombinationsfelder 446
270 Kombinationsfeld füllen – Variante 1 446
271 Kombinationsfeld füllen – Variante 2 449
272 Editieren zulassen 450

Kontrollkästchen **453**
273 Kontrollkästchen auswerten 453
274 Kontrollkästchen initialisieren 454

Optionsfelder **455**

Schaltflächen **457**
275 Eine blinkende Schaltfläche erstellen 461

Weitere Steuerelemente **463**
276 Kalender-Steuerelement 463
277 Der Fortschrittsbalken 467
278 Der Webbrowser 469
279 Zugriff auf Outlook-Postfach 470
280 Das TreeView-Steuerelement 471
281 Das Media Player-Steuerelement 472
282 Diagramme in Formulare integrieren 475
283 Diagramme in Formulare integrieren in Access 2007 478
284 Das SpreadSheet-Steuerelement 483
285 Das Slider-Steuerelement 486
286 Das Spinbutton-Steuerelement 488

Berichte & Datenzugriffsseiten **491**

Berichtstypen **491**

Der Aufbau eines Berichts **491**

Berichte mit DoCmd steuern **492**
287 Berichte öffnen 493
288 Berichtsfilter setzen 494
289 Bericht schließen 495
290 Berichte drucken 496
291 Berichte kopieren 497
292 Berichte umbenennen 499
293 Berichte exportieren 500

Sonstige Lösungen für Berichte **501**
294 Berichte formatieren 501
295 Berichte listen 505
296 Berichtselemente identifizieren 506
297 Berichtsteil ein- und ausblenden 510
298 Eigene Berichte erstellen 511
299 Berichtselemente einfügen 513

Datenzugriffsseiten **517**
300 Datenzugriffsseiten auslesen 520
301 Speicherort der Datenzugriffsseiten ermitteln 521
302 Datenzugriffsseiten anzeigen 522
303 Datenzugriffsseite anpassen 523

Die Ereignisprogrammierung 525

Die Ereignisse Form_Open bzw. Report_Open **525**
304 Zugangsverwaltung in Access (Formular) 525
305 Zugangsverwaltung in Access (Bericht) 527
306 Bestimmtes Feld fokussieren 528
307 Sicherheitscheck durchführen 529

Die Ereignisse Form_Close und Report_Close **530**
308 Alle beteiligten Elemente schließen 530
309 Nach fünf Sekunden automatisch schließen 531
310 Zugriff dokumentieren 532

Das Ereignis Form_Load **534**
311 Listenfeld beim Laden des Formulars füllen 534

Das Ereignis Form_Unload **535**
312 Schließen verhindern 535
313 Rechnungsfälligkeiten überwachen 537

Das Ereignis Form_Current **538**
314 Information beim letzten Satz 538
315 Dynamisches Ein- und Ausblenden von Feldern 539
316 Variablen Titel im Formular einstellen 541
317 QuickInfos definieren 542
318 Eingaben limitieren 543

Das Ereignis Form_AfterInsert **544**
319 Neue Mitarbeiter dokumentieren 544

Das Ereignis Form_BeforeInsert **545**
320 Felder automatisch vorbelegen 546

Das Ereignis Form_BeforeUpdate **548**
321 Rückfrage vor dem Speichern einholen 548
322 Eingabe-Prüfung vornehmen 549
323 Muss-Felder definieren 551
324 Änderungen in Textdatei festhalten 551
325 Neuanlage von Datensätzen verhindern 553
326 Überhaupt keine Änderungen zulassen 553

Das Ereignis Form_AfterUpdate **554**
327 Letzte Änderung festhalten 554

Das Ereignis Form_Delete **555**
328 Löschen von Datensätzen verhindern 555

Das Ereignis BeforeDelConfirm **556**
329 Eine eigene Löschabfrage einsetzen 557

Das Ereignis Form_Activate **557**
330 Den letzten Datensatz einstellen 557
331 Den zuletzt geänderten Datensatz aufrufen 558

Das Ereignis Form_Resize **559**
332 Variable Feldanpassung vornehmen 559

Das Ereignis Form_DblClick **561**
333 Mit Doppelklick Felder ein- und ausblenden 561

Das Ereignis Form_Click **562**
334 Lagerbestand per Mausklick erhöhen 562

Die Ereignisse Form_MouseDown und Form_MouseUp **563**
335 Was wurde angeklickt? 563
336 Schaltfläche beim Klicken verändern 564
337 Kontextmenü deaktivieren 565

Das Ereignis MouseMove **566**
338 Spezialeffekt einsetzen 566
339 Ins Internet verzweigen 568

Die Ereignisse GotFocus und LostFocus **570**
340 Textfelder optisch hervorheben 570

Die Key-Ereignisse **572**
341 Tasten erkennen 572
342 Tastenkombinationen einrichten 573
343 Aktuellen Datensatz ans Ende kopieren 575

Das Ereignis Steuerelement_BeforeUpdate **576**
344 Doppelte Eingaben verhindern 576
345 Eingaben vorwegnehmen 577
346 Feldlöschung verhindern 578

Das Ereignis Steuerelement_Enter **579**
347 QuickInfos anzeigen 579

Das Ereignis Steuerelement_Exit **580**

Die Reihenfolge der Ereignisse **581**
348 Reihenfolge beim Öffnen und Schließen eines Formulars 581
349 Aktivierreihenfolge bei Steuerelementen 582
350 Reihenfolge der Aktualisierungsereignisse 583
351 Eine komplette Kette von Ereignissen 583

VBE & Security **585**

Quellcode & Datenbanken schützen **585**
352 Kennwort für die Anzeige des Quellcodes anlegen 585
353 Datenbank ohne Quellcode speichern 586
354 Datenbanken verschlüsseln 587
355 Access-Lösung mithilfe von Startparametern absichern 588
356 Schützen einer Datenbank über ein Kennwort 590
357 Datenbankkennwort ändern 597

Arbeitsgruppen in Access **598**
358 Arbeitsgruppe anlegen 599
359 Arbeitsgruppe entfernen (DAO) 600

360 Benutzer anlegen 601
361 Benutzer entfernen 603
362 Benutzer auslesen 604
363 Berechtigungen zuweisen 605

Der Zugriff auf die VBE **607**
364 Die VBE-Bibliothek einbinden 608
365 Verweise auslesen 615
366 Auf der Suche nach fehlerhaften Verweisen 616
367 VB-Komponenten identifizieren 617
368 Die VBE aufrufen 618
369 Neue Module einfügen 619
370 Modul löschen 621
371 Alle Module löschen 621
372 Modulcheck durchführen 622
373 Module drucken 623
374 Prozedur löschen 624
375 Prozedur-Check durchführen 626
376 Formular-Ereignisse löschen 627
377 Einzelne Texte/Befehle im Quellcode finden 629
378 Texte ersetzen 631
379 Prozeduren eines Moduls auflisten 634
380 Prozeduren einfügen 637
381 Formularcode importieren 640
382 Formular-Ereignisprozedur einstellen 640
383 Prozeduren exportieren 642
384 Formularcodes sichern 643

Access und … **647**

Early- und Late-Binding **647**
385 Early-Binding 647
386 Late-Binding 647

Access & Outlook **648**
387 Ist Outlook bereits gestartet? 648
388 E-Mails verschicken 651
389 E-Mail mit Anhang versenden 654
390 E-Mail-Adressen aus Outlook herauslesen (Early-Binding) 655
391 Kontaktdaten aus Outlook herauslesen (Late-Binding) 656
392 Kontaktdaten nach Outlook übertragen 658
393 Kontaktdaten von Outlook importieren 661
394 Termine aus dem Outlook-Kalender abfragen 665
395 Aufgaben abfragen 667
396 E-Mails in Access-Tabelle importieren 670
397 Sammel-E-Mails versenden 673
398 E-Mail-Anhänge speichern 675

Access & Internet Explorer **676**
399 Eine Diashow programmieren 677
400 URLs auslesen 678

Access & Word **679**
401 Ist Word bereits gestartet? 680
402 Access-Tabelle übertragen 681
403 Access-Tabellen transferieren 685
404 Word-Prozeduren starten 687
405 Datenbankzugriff mit Word durchführen 687

Access & Excel **693**
406 Ist Excel bereits gestartet? 693
407 Access-Tabelle in eine Excel-Tabelle umwandeln 695
408 Excel-Daten in eine Access-Tabelle transferieren 697
409 Access als Datenlieferant für Excel 699
410 Excel-Prozedur starten 716
411 Auf Excel-Funktionen zugreifen 717

Access & PowerPoint **718**
412 Ist PowerPoint bereits gestartet? 718
413 PowerPoint-Dateien suchen und öffnen 719

Access mit Textdateien **721**
414 Textdateien speichern 721
415 Eine weitere Methode Textdateien zu speichern 724
416 Codes sichern 725
417 Textdatei importieren 726

Symbol- und Menüleisten 731

Symbol- und Menüleisten bis Access 2003 **731**
418 Leisten identifizieren 731
419 Symbolleiste ein- und ausblenden 733
420 Alle Symbolleisten ein- und ausblenden 734
421 Symbolleisten-IDs ermitteln 735
422 Leisten und Befehle ermitteln 736
423 Symbolschaltflächen (de)aktivieren 737
424 Neue Symbolleiste erstellen 738
425 Symbolleiste löschen 739
426 Symbolschaltflächen-FaceIDs ermitteln 739
427 Symbolschaltflächen-IDs ermitteln 741
428 Symbolschaltflächen einfügen 742
429 Symbolleisten schützen 744
430 Symbolschaltflächen (de)aktivieren 746
431 Adaptive Menüs ausschalten 747
432 Neues Menü einfügen 748
433 Menüleiste zurücksetzen 750
434 Menü löschen 751
435 Menübefehle einfügen 751
436 Menübefehle gruppieren 753
437 Menü auslesen 753
438 Menübefehle auslesen 754
439 Menüs (de)aktivieren 755
440 Menübefehle (de)aktivieren 757

Die Multifunktionsleiste programmieren **758**
441 Programmierung über ein XML-Markup 759
442 Anlage der Tabelle USysRibbons 760
443 Ereignis-Prozedur erstellen 762
444 Neue Menüleiste auswählen 762
445 Dynamisches Laden ohne USysRibbon 764

Weitere wichtige Quellen und Hilfen **766**

Teil II Anhang 769

VBA-Funktionen 771

Datums- und Zeitfunktionen **771**
Textfunktionen **772**
Dateifunktionen **773**
Mathematische Funktionen **775**
Prüffunktionen **776**
Umwandlungsfunktionen **777**
Farbfunktionen **778**
Suchfunktionen **779**

Übersicht der Konstanten 781

Die RGB-Konstanten **781**
Die RunCommand-Konstanten **781**
Dir-Konstanten **787**
Die Shell-Konstanten **788**
StrConv-Konstanten **789**
Var-Type-Konstanten **789**
File Input-/Output-Konstanten **790**
Datumsformat-Konstanten **790**
Schaltflächen auswerten **791**

Stichwortverzeichnis 793

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 mit ausgelieferten CD-ROM in den einzelnen Demodateien abrufbar.

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

Im *Kapitel Allgemeine VBA-Funktionen* können Sie typische Lösungen rund um das Thema Funktionen nachschlagen. Dabei handelt es sich um Access-spezifische sowie VBA-Standardfunktionen.

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

Das *Kapitel Die Access-Objekte* steigt in die Welt der wichtigsten Access-Objekte ein. Hier können Sie Objekte von der Applikation selbst über Menü- und Symbolleisten bis hin zu Objekten für die Datei- und Verzeichnisverwaltung einsetzen.

Im *Kapitel Tabellen programmieren* liegen Lösungen zur Tabellenprogrammierung vor. Dabei gibt es in diesem Kapitel jeweils Lösungen mit ADO, DAO und ADOX.

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

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

Das *Kapitel Berichte & Datenzugriffsseiten* beschäftigt sich mit dem Thema der Berichtsprogrammierung. Sie finden in diesem Kapitel Lösungen vor, wie Sie Berichte beispielsweise ansprechen, formatieren und erstellen können.

Im *Kapitel Die Ereignisprogrammierung* 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 Steuerelementebene bis hin zu Tasten- und Mausereignissen.

Das *Kapitel VBE & Security* geht der Frage nach, wie man Quellcode und Datenbanken schützen kann. Des Weiteren können Sie in diesem Kapitel nachschlagen, wie Sie Arbeitsgruppen anlegen, Benutzer einfügen und löschen können. Ein weiterer Bestandteil dieses Kapitels ist die VBE-Programmierung. Dabei werden beispielsweise

Module, Funktionen, Prozeduren und Ereignisse eingefügt bzw. gelöscht, Quellcodes importiert und exportiert und vieles mehr.

Im *Kapitel Access und ...* dieses Buches 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 dieses Kapitels ist die Bearbeitung von Textdateien in Access.

Im letzten *Kapitel Symbol- und Menüleisten* werden Lösungen zu Menü- und Symbolleisten vorgestellt. Auch die neuen Ribbons aus Access 2007 sind ein Thema dieses Kapitels.

Bei Nachfragen und allgemeinem Feedback zu meinem Buch erreichen Sie mich über meine Excel-Homepage *http://held-office.de* oder über info@pearson.de. So erreichen Sie gleichzeitig auch den Verlag.

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

Bernd Held

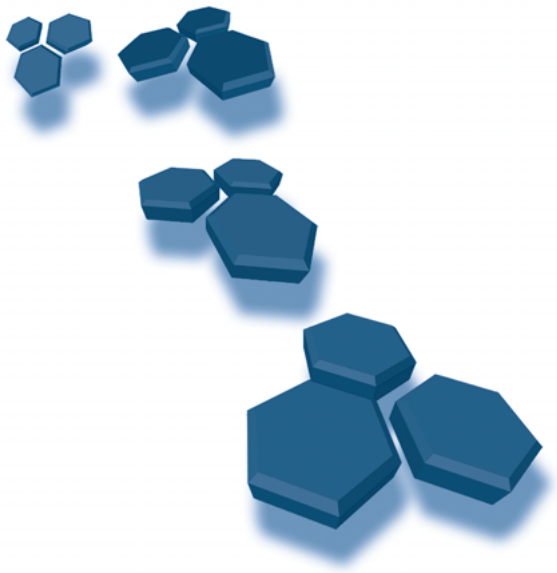
Über den Autor

Bernd Held

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, Access und die Office-VBA-Programmierung spezialisiert. Aber auch über Microsoft Works, FrontPage, Windows und diverse andere Themen hat er schon viele Bücher geschrieben.

Bernd Held wurde acht Mal 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.

Teil I Rezepte



Allgemeine VBA-Funktionen

Weitere Funktionen

Die Access-Objekte

Tabellen programmieren

Abfragen programmieren

Steuerelemente, Dialoge und Formulare

Berichte & Datenzugriffsseiten

Die Ereignis-programmierung

VBE & Security

Access und …

Symbol- und Menüleisten

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 können in verschiedene Gruppen eingeteilt werden. So gibt es beispielsweise Textfunktionen, 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

Die Datums- und Zeitfunktionen

Wie schon der Name verrät finden Sie in dieser Kategorie Funktionen, über die Sie Datums- und Zeitrechnungen in VBA vornehmen können.

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
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=======================================================

Sub DatumWandeln()
  Dim strDatum As String
  Dim dtmDatum As Date
  strDatum = "17. Mai 2008"
  dtmDatum = 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
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

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

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

2 Systemdatum abrufen

Zugriff auf das Systemdatum, das in Windows hinterlegt ist, gewinnen 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
' 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

Sollte das Datum beziehungsweise 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

In der folgenden Prozedur aus *Listing 4* wird ausgehend von einem Bestelldatum der Liefertermin errechnet und im Direktfenster der Entwicklungsumgebung ausgegeben.

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

Listing 4: Das Enddatum mit der Funktion DateAdd ausrechnen

```
Sub EndTerminErrechnen()
  Dim dtmBestellung As Date
  Dim strIntervall As String
  Dim intZahl As Integer

  dtmBestellung = "20.07.2008"
  Debug.Print "Bestelldatum: " & dtmBestellung
  strIntervall = "m"
  intZahl = 3
  Debug.Print "Lieferdatum: " & _
  DateAdd(strIntervall, intZahl, dtmBestellung)
End Sub
```

Listing 4: Das Enddatum mit der Funktion DateAdd ausrechnen (Forts.)

Über die Variable `strIntervall` geben Sie bekannt, in welcher Einheit Sie rechnen möchten. Die dafür infrage 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.

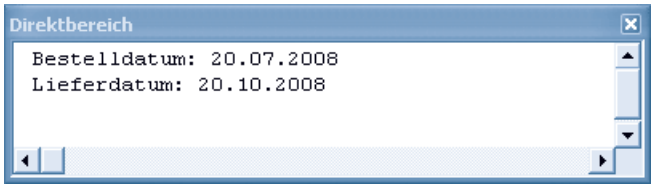


Abbildung 2: Ein Lieferdatum in der Zukunft errechnen

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 erstem 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. Die einzelnen Belegungen der Konstanten sehen Sie in *Tabelle 3*.

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

Beim 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
' 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

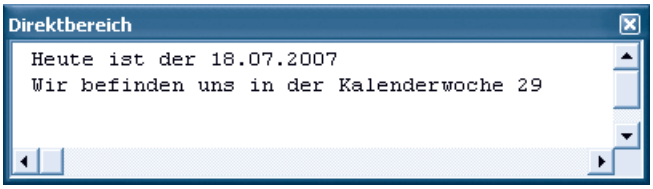


Abbildung 3: Aus einem Datum die dazugehörige KW errechnen

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

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

Listing 6: Die Kalenderwoche über eine Funktion abrufen

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

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

Listing 6: Die Kalenderwoche über eine Funktion abrufen (Forts.)

Übergeben Sie der Funktion `DINKw` das aktuelle Datum, das 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, die zwischen zwei bestimmten Terminen liegen, angeben. Die Syntax für diese Funktion lautet:

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

Da die Argumente der Funktion `DateDiff` und der Funktion `DatePart` fast gleich sind, lesen Sie weiter vorn im Kapitel nach.

In der folgenden Prozedur aus *Listing 7* werden die Tage bis zum Jahreswechsel errechnet und im Direktfenster ausgegeben.

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

Sub TageBisZumJahreswechsel()
  Dim dtmStart As Date
  Dim dtmEnde As Date

  dtmStart = Date
  dtmEnde = "31.12.2007"
  Debug.Print "Anzahl Tage von heute " & _
    dtmStart & vbLf & "bis zum Jahresende: " & _
  DateDiff("d", dtmStart, dtmEnde)
End Sub
```

Listing 7: Verbleibende Tage bis zum Jahreswechsel ermitteln

Deklarieren Sie zu Beginn der Prozedur zwei Variable vom Typ `Date`. Danach füllen Sie die Variable `DateStart` mit dem aktuellen Systemdatum. Die Variable `DateEnde`

füllen Sie mit dem letzten Tag des Jahres. Über die Funktion `DateDiff` ermitteln Sie den Zeitraum, der zwischen diesen beiden Zeitpunkten liegt. Damit die Funktion weiß, in welcher Einheit sie die Berechnung durchführen soll, übergeben Sie als erstes Argument das Kürzel `d`, das für die Einheit Tage steht.

Abbildung 4: Es sind noch genau 166 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

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

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

Sub TageSchreiben()
  Dim intZ As Integer

  For intZ = 29 To 1 Step -1
    Debug.Print Format(DateSerial(2008, 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 bei jedem Schleifendurchlauf den Wert 1. Die Schrittweite müssen Sie bei dieser Prozedur 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 die Prozedur aus *Listing 9*.

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

Sub TageSchreibenVorwärts()
  Dim intZ As Integer

  For intZ = 1 To 29
    Debug.Print Format(DateSerial(2008, 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 1. Februar an und enden beim 29. Februar. Die Schrittweite müssen Sie bei dieser Schleife nicht angeben, da diese Schleife standardmäßig in Einserschritten arbeitet.

Hinweis

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

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

Beim folgenden Beispiel aus *Listing 10* wird ein etwas ungewöhnliches Datum in ein gängiges Datumsformat umgesetzt:

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

Sub EingabeWandeln()
   Dim dtm As Date

   dtm = DateValue("1/1/08")
   MsgBox dtm
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 für 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 Beispiel aus *Listing 11* wird der Funktion `TimeValue` ein Zeitwert übergeben, der als Text vorliegt.

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

Sub ZeitenErkennenUndWandeln()
  Dim dtmZeit As Date

  Debug.Print "Vorher: " & "3:21:59 PM"
  dtmZeit = TimeValue("3:21:59 PM")
  Debug.Print "Nachher: " & dtmZeit
End Sub
```

Listing 11: Eine Uhrzeit erkennen und umwandeln

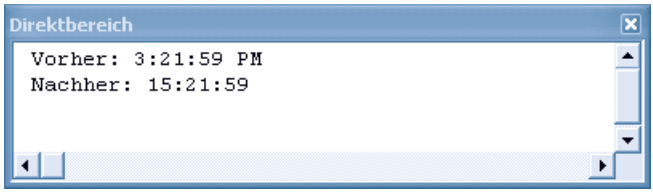


Abbildung 6: Die Uhrzeit wurde erkannt und umgewandelt.

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
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=======================================================

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

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

  Debug.Print "Aus:  " & dtmBeginn
  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 der Prozedur vier Variable. In der ersten Variable, dtmBeginn, vom Typ Date speichern Sie das aktuelle Systemdatum, das Sie über die Funktion Date ermitteln. Danach wird diese Variable mithilfe der Funktionen Day, Month und Year in die drei Variable intTag, intMonat und intJahr zerlegt.

Geben Sie dann das Datum zuerst unverändert im Direktfenster der Entwicklungsumgebung über die Anweisung Debug.Print aus. Danach setzen Sie die Einzelteile des Datums, die nun in den Integer-Variablen vorliegen, über den Einsatz des Verkettungsoperators & in einer anderen Reihenfolge zusammen. Dabei geben Sie mithilfe der Funktion Format an, dass die Tages- und Monatsangaben zweistellig sein müssen, da Access sonst die führenden Nullen bei einstelligen Tagen sowie Monaten entfernen würde. So wird der 1. Januar nicht als »1.1.2008«, sondern als »01.01.2008« ausgegeben.

Abbildung 7: Datum zerlegen und wieder zusammensetzen

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 12 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
' 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, verhindert die Anweisung `On Error GoTo Fehler` einen Prozedurabsturz.

11 Änderungsdatum einer Datenbank ermitteln

Die Funktion `FileDateTime` liefert das Datum der letzten Änderung 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.

Beim folgenden Beispiel aus *Listing 14* wird das Änderungsdatum der aktiven Datenbank ermittelt und am Bildschirm ausgegeben.

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

Sub ÄnderungsdatumAbfragen()
  MsgBox "Datei zuletzt geändert am " & vbLf & _
    FileDateTime(Application.CurrentDb.Name), vbInformation
End Sub
```

Listing 14: Das Änderungsdatum 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 Änderungsdatum der aktuellen Datenbank abfragen

Im folgenden Beispiel aus *Listing 15* wird auf eine externe, nicht geöffnete Datenbank zugegriffen und das Datum der letzten Änderung der Datei ermittelt.

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

Sub ÄnderungsdatumAbfragenExterneDB()
  On Error GoTo Fehler
  MsgBox "Datei erstellt am " & vbLf & _
  FileDateTime("C:\Eigene Dateien\Kundendienst1.mdb"), vbInformation
```

Listing 15: Das Erstellungsdatum einer nicht geöffneten Datenbank ermitteln

```
  Exit Sub

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

Listing 15: Das Erstellungsdatum einer nicht geöffneten Datenbank ermitteln (Forts.)

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

Hinweis

Das Erstellungsdatum, das Sie über die Methode `FileDateTime` erhalten, 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
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=====================================================

Sub ÄnderungsdatumAlleDateienErmitteln()
  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 "strDateiname: " & strDatei
    Debug.Print "Geändert am: " & FileDateTime(strOrdner & strDatei) & vbLf
    strDatei = Dir
    intDatei = intDatei + 1
  Loop
End Sub
```

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

Deklarieren Sie zu Beginn der Prozedur aus *Listing 16* einige Variablen. In der Variablen `strOrdner` geben Sie das Verzeichnis an, indem Sie die Dateien auslesen möchten.

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

In der Variablen `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 Änderungsdatums 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.

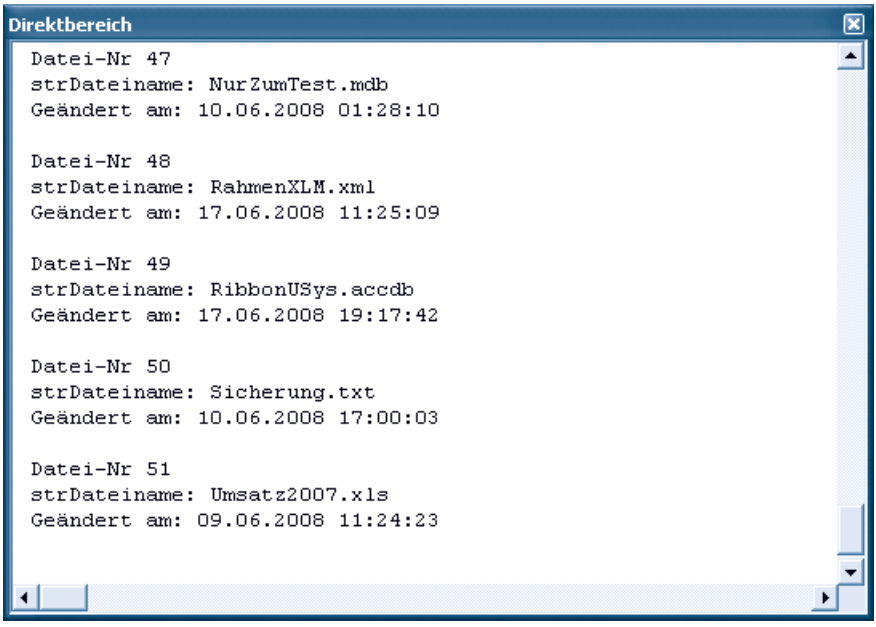


Abbildung 9: Die Namen und das Erstellungsdatum der Dateien wurden dokumentiert.

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
' Dateiname  Funktionen.mdb
' Modul      MdlDate
'=======================================================

Sub DatumFormatiertAusgeben()
  Dim dtmBeginn As Date

  dtmBeginn = Now
  Debug.Print FormatDateTime(dtmBeginn, vbGeneralDate)
  Debug.Print FormatDateTime(dtmBeginn, vbLongDate)
  Debug.Print FormatDateTime(dtmBeginn, vbShortDate)
```

Listing 17: Ein Tagesdatum verschiedenartig formatieren

```
  Debug.Print FormatDateTime(dtmBeginn, vbLongTime)
  Debug.Print FormatDateTime(dtmBeginn, vbShortTime)
End Sub
```

Listing 17: Ein Tagesdatum verschiedenartig formatieren (Forts.)

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

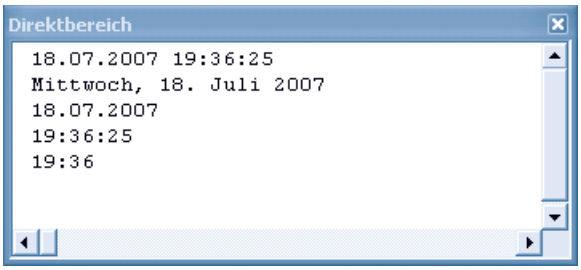


Abbildung 10: Verschiedene Möglichkeiten ein Datum auszugeben

13 Zeit zerlegen

Neben den Datumsfunktionen `Year`, `Month` und `Day` gibt es auch Funktionen, um einen Zeitwert in seine Einzelteile zu zerlegen. Es gibt 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* werden zur Demonstration aus dem aktuellen Zeitpunkt, bestehend aus Datum und Uhrzeit, die aktuelle Stunde, die aktuelle Minute und die aktuelle Sekunde extrahiert und als Text ausgegeben. Anschließend werden diese Zeitwerte über die `TimeSerial`-Funktion wieder zu einer Zeit zusammengefügt, die Sie dann als formatierten Zeitwert ausgeben.

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

Sub ZeitZerlegen()

  Dim dtmJetzt As Date
  Dim intStunde As Integer
  Dim intMinute As Integer
  Dim intSekunde As Integer

  'Holen von aktuellem Datum und Zeit
  dtmJetzt = Now()

  intStunde = Hour(dtmJetzt)
  intMinute = Minute(dtmJetzt)
  intSekunde = Second(dtmJetzt)

  Debug.Print "Aktueller Zeitpunkt: " & dtmJetzt
  Debug.Print "Die aktuelle Stunde ist: " & intStunde
  Debug.Print "Die aktuelle Minute ist: " & intMinute
  Debug.Print "Die aktuelle Sekunde ist: " & intSekunde

  Debug.Print "TimeSerial liefert Ihnen: " & TimeSerial(intStunde, ↵
  intMinute, intSekunde)
End Sub
```

Listing 18: Die Zeit zerlegen und wieder zusammenfügen

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
' 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–DEZ und in normaler Form JANUAR–DEZEMBER.

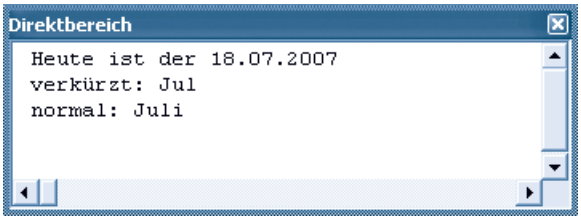


Abbildung 11: Den Monatsnamen aus einem Datum extrahieren

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 von MONTAG–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 Wochentages. Der numerische Wert der einzelnen Tage hängt von der Einstellung für ErsterWochentag ab.
abkürzen	Optional. Boolescher Wert, der angibt, ob der Name des Wochentages abgekürzt wird. Wird er ausgelassen, ist die Standardeinstellung False, d. h. der Name des Wochentages 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* wird ausgehend von einem vorgegebenen Datum der dazugehörige Wochentag sowie das Quartal ermittelt.

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

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

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

Sub WochenTagUndQuartal()
  Debug.Print "Dieses Datum liegt im " & _
    Quartal("20.07.2008") & " .Quartal!"
  Debug.Print "Der Wochentag ist ein " & _
    Wochentag("20.07.2008")
End Sub
```

Listing 20: Quartal und Wochentag 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.

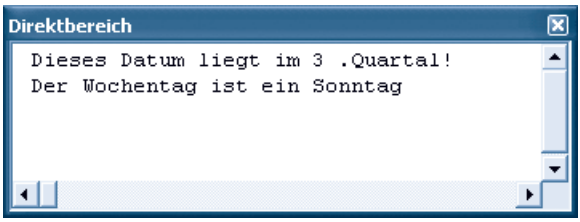


Abbildung 12: Der Wochentag sowie das Quartal wurden ausgelesen.

Die Textfunktionen

Oft liegen Daten nicht so vor, wie man sie eigentlich brauchen würde. Die Textfunktionen helfen dabei, Daten in ein gewünschtes Format zu bringen.

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 `Zeichenfolge`-Argument 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, sodass nur noch Zahlenwerte zurückbleiben.

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

Listing 21: Buchstaben aus Zeichenfolgen entfernen

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

Listing 21: Buchstaben aus Zeichenfolgen entfernen (Forts.)

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 der Prozedur `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 65–90 Kleinbuchstaben und die Werte 97–122 den Großbuchstaben und die restlichen Werte den Sonderzeichen.

An die aufrufende Prozedur werden von der Funktion nur noch numerische Zeichen zurückgegeben.

Abbildung 13: Alle Buchstaben in der Zeichenfolge wurden entfernt.

17 Zahlen aus Zeichenfolgen entfernen

Analog zu der Lösung aus *Listing 21* können Sie ebenso numerische Werte aus einer Zeichenfolge entfernen, indem Sie die Prozedur aus *Listing 22* starten.

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

Listing 22: Zahlen aus Zeichenfolge entfernen

```
      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 (Forts.)

Als Erweiterung der Prozedur aus *Listing 21* wurde in der Prozedur 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. Dazu 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 oder 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
' 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 Argumentsfolge 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.

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

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.

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
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=======================================================

Sub AusZahlenBuchstabenErzeugen()
  Dim lng As Long

  For lng = 65 To 90
    Debug.Print Chr(lng)
  Next lng
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
Zeichenfolge2 ist nicht vorhanden	0
Zeichenfolge2 ist in Zeichenfolge1 enthalten	Position, an der Übereinstimmung beginnt
Start > Zeichenfolge2	0

Tabelle 14: Die Rückgabewerte der Funktion InSt

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 @ darin vorkommt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 `strUrl` gefunden, dann meldet die Funktion einen Rückgabewert größer als 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 ungültige E-Mail-Adresse.

Soll der E-Mail-Adressen-Check ein wenig erweitert werden und beispielsweise sichergestellt werden, dass keine Umlaute wie ä, ö und ü eingesetzt werden, dann starten Sie die Prozedur aus *Listing 26*.

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

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
```

Listing 26: Die erweiterte E-Mail-Adressenprüfung

```
    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 auf Umlaute 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

Beim 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 vom rechten Rand einer Zeichenfolge beginnend 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.
`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

Im folgenden Beispiel aus *Listing 27* wird der komplette Pfad, der Pfad- sowie der Dateiname der aktiven Datenbank ermittelt.

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

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

```
Sub PfadUndDateiname()
  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) (Forts.)

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

So erhalten 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.

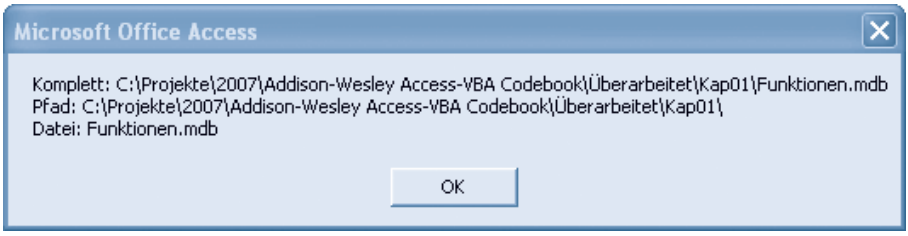


Abbildung 17: Pfad und Dateiname wurden separiert.

Anwender der Version Access 97 haben bei der Prozedur 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.

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

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

  intPos = 1
  Do Until intPos = 0
    intAltPos = intPos
    intPos = InStr(intPos + 1, _
      Application.CurrentDb.Name, "\")
  Loop
  strPfad = Left(Application.CurrentDb.Name, intAltPos)
  strName = Mid(Application.CurrentDb.Name, _
    intAltPos + 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 Pfadnamen 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 `intAltPos` verzeichnet sind.

Den Namen der Datei bekommen Sie, indem Sie zu der letzten Fundstelle des Zeichens »\«, die in der Variablen `intAltPos` 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, die in einem Datenfeld enthalten sind, ergibt. 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

Beim 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
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================
```

Listing 29: Texte zerlegen, aufbereiten und wieder zusammensetzen

```
Sub JoinSplitt()
  Dim strText As String
  Dim var As Variant
  Dim intZ As Integer
  Dim intGesamt As Integer

  strText = "Vorname;Nachame;Straße;PLZ;Ort"
  Debug.Print "Vor Konvertierung: " & strText
  var = Split(strText, ";")
  intGesamt = UBound(var)
  For intZ = 0 To intGesamt
    var(intZ) = UCase(var(intZ))
  Next intZ
  var = Join(var, "-")
  Debug.Print "Nach Konvertierung: " & var
End Sub
```

Listing 29: Texte zerlegen, aufbereiten und wieder zusammensetzen (Forts.)

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 Datenfeldes 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.

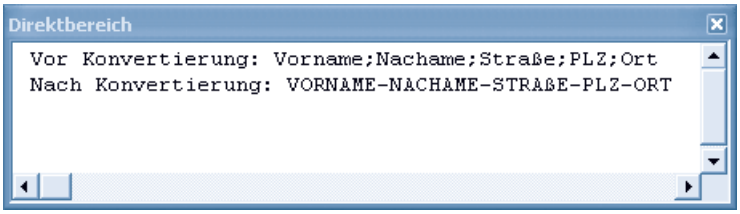


Abbildung 18: Text wurde zerlegt, aufbereitet und wieder zusammengesetzt.

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
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=======================================================

Sub TextKleinMachen()
  Dim strText As String

  strText = "TEST"
  MsgBox LCase(strText)
End Sub
```

Listing 30: Text in Kleinschreibweise umwandeln

Eine weitere Möglichkeit, Texte in Groß- bzw. Kleinschreibweise 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`. Gibt den Typ der auszuführenden Umwandlung an.
`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.

Tabelle 19: Einstellungen für das Argument conversion

Konstante	Wert	Beschreibung
vbWide*	4	Wandelt schmale Zeichen (Einzel-Byte) innerhalb der Zeichenfolge in breite Zeichen (Doppel-Byte) um.
vbNarrow	8	Wandelt breite Zeichen (Doppel-Byte) innerhalb der Zeichenfolge in schmale Zeichen (Einzel-Byte) 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 (Forts.)

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

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 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 Objektes `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
- Access 2003 12.0

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 machen gerade ältere Access-Versionen bei der Programmierung Kummer, weil bestimmte VBA-Befehle noch nicht im Befehlswortschatz sind und daher Prozedurabstürze verursachen.

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

Sub AccessVersionErmitteln()
  MsgBox Application.Version
  Select Case Left(Application.Version, 2)
    Case "5"
      MsgBox "Sie haben Access 5 im Einsatz"
```

Listing 32: Die Access-Version ermitteln und auswerten

```
    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 oder höher im Einsatz!"
  End Select
End Sub
```

Listing 32: Die Access-Version ermitteln und auswerten (Forts.)

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

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

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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` die Gesamtlänge des Vorgabetextes in der Variablen `strText`. Diese Gesamtlänge wird als Ende-Kriterium 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.

Abbildung 19: Text zerlegen und ausgeben

26 Datei-Endung 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.

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 (rechts stehenden) 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 Dateiname in eine Inputbox eingegeben werden. Danach wird geprüft, ob es sich um einen Access-Dateinamen handelt. Dies können Sie an der Dateierweiterung .MDB oder .ACCDB erkennen, wobei es sich bei Letzterem um eine Access-Datenbank im neueren Format handelt, das mit Access 2007 eingeführt wurde.

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

Sub DateiTypChecken()
  Dim strDatei As String
  strDatei = InputBox("Bitte strDateinamen eingeben", _
    "strDateinamen:")

  If Len(strDatei) <> 0 Then
    If UCase(Right(strDatei, 4)) = ".MDB" Or _
      UCase(Right(strDatei, 6)) = ".ACCDB" Then
      MsgBox "Es handelt sich um eine Access-Datenbank!"
    Else
      MsgBox "Es handelt sich um keine Access-Datenbank!"
    End If
  End If
End Sub
```

Listing 34: Den Dateityp über die Endung prüfen

Extrahieren Sie am Dateinamen zum einen die letzten vier Zeichen über die Funktion `Right` und überprüfen Sie, ob das Ergebnis dem String `.MDB` entspricht. Indem Sie in einem zweiten Schritt die letzten sechs Ziffern extrahieren und überprüfen, ob das Ergebnis dem String `.ACCDB` entspricht, detektieren Sie auch das neue Access 2007-Datenbank-Format.

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 und
- aus ü wird ue.

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

Sub UmlauteErsetzen()
  Dim strText As String

  strText = "Sie können sich darüber ärgern oder nicht!"
  Debug.Print "Aus " & strText
  strText = Replace(strText, "ü", "ue")
  strText = Replace(strText, "ä", "ae")
  strText = Replace(strText, "ö", "oe")
  Debug.Print "wird " & 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.

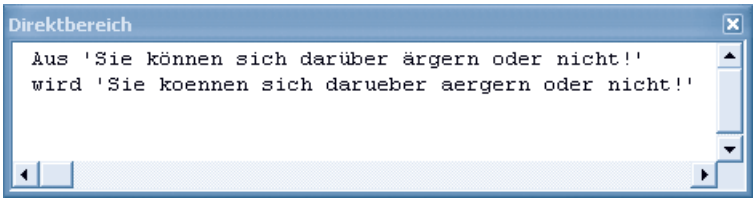


Abbildung 20: Zeichenfolgen schnell austauschen

28 Zeichenfolge mit Leerzeichen auffüllen

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

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 10 Zeichen bestehen. Liegen im Text also beispielsweise nur 4 Zeichen vor, dann werden noch 6 Leerzeichen im Text integriert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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, dann 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 das erste Zeichen 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
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=======================================================
```

Listing 37: Zahlenwert in einen Textwert umwandeln

```
Sub ZahlInTextWandeln()
  Dim lng As Long

  lng = 70469
  Debug.Print Str(lng)
End Sub
```

Listing 37: Zahlenwert in einen Textwert umwandeln (Forts.)

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

Tabelle 26: Die Vergleichstypen der Funktion StrComp

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
' 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

Beim 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 im 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
' 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 als 10, dann werden die restlichen Zeichen durch einen Stern vervollständigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 einen 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
' 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
  End If

Fehler:
  MsgBox "Das Land ist nicht eingepflegt!"
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
' 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
' Dateiname  Funktionen.mdb
' Modul      MdlTXT
'=====================================================

Function LeereRaus(str As String) As String
  Dim intZ As Integer

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

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

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 der aufrufenden Prozedur als Rückgabewert übergeben.

Die Dateifunktionen

Die bekanntesten Funktionen in dieser Kategorie sind die Funktionen zum Erstellen, Löschen oder Wechseln von Verzeichnissen. Weitere Funktionen können eingesetzt werden, um Dateien einzufügen bzw. zu entfernen.

35 Aktuelles Verzeichnis wechseln

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

```
ChDir Pfad
```

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

Im folgenden Beispiel aus *Listing 44* wird das aktuelle Verzeichnis neu gesetzt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 aktuelle Verzeichnis über ChDir wechseln

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

36 Aktuelles Laufwerk wechseln

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
' 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 Doppelpunktes.

Hinweis

Im *Kapitel Weitere Funktionen* dieses Buches können Sie nachschlagen, wie Sie ein Laufwerk bzw. ein Verzeichnis durch den Einsatz einer API-Funktion über einen Auswahldialog einstellen können.

37 Aktuelles Verzeichnis ermitteln

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 aktuelle Verzeichnis des Laufwerks C am Bildschirm ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 des Laufwerks, dessen aktuelles Verzeichnis ermittelt werden soll.

38 Dateien und Verzeichnisse auflisten

Um die Dateien aus einem Verzeichnis aufzulisten, können Sie die Ihnen vielleicht 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.
`vbReadOnly`	1	Schreibgeschützte Dateien, zusätzlich zu Dateien ohne Attribute.

Tabelle 31: Die möglichen Einstellungen beim Argument Attribute

Konstante	Wert	Beschreibung
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 protokolliert.

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

Sub DateienAuflisten()
  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 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.

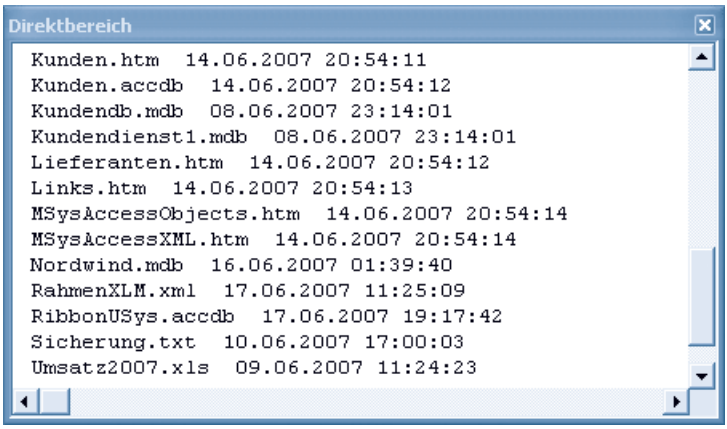


Abbildung 22: Dateidokumentation im Direktfenster der Entwicklungsumgebung

39 Verzeichnis anlegen

Mithilfe der Anwesung `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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=======================================================

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

Listing 48: Neues Verzeichnis über MkDir anlegen

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

40 Verzeichnis löschen

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.

Achtung

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

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

Listing 49: Ein Verzeichnis über die Anweisung RmDir löschen

Übergeben Sie der Anweisung `RmDir` den kompletten Pfad zu dem Verzeichnis, das Sie löschen möchten.

41 Datei löschen

Die Anweisung `Kill` löscht eine Datei ohne Rückfrage. 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

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

Listing 50: Datei über die Anweisung Kill löschen

Übergeben Sie der Anweisung `Kill` den Namen der Datei inklusive des Pfades, 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

Beim folgenden Beispiel aus *Listing 51* wird eine Textdatei kopiert.

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

Sub DateiKopieren()
  Dim strDat As String
```

Listing 51: Dateien kopieren über die Anweisung FileCopy

```
  Dim strDatKopie As String
  strDat = "C:\Eigene Dateien\Artikel.txt"
  strDatKopie = "C:\Eigene Dateien\ArtikelKopie.txt"

  On Error GoTo Fehler
  FileCopy strDat, strDatKopie
  Exit Sub

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

Listing 51: Dateien kopieren über die Anweisung FileCopy (Forts.)

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.

43 Bilder umbenennen und nummerieren

Auf vielen PCs gibt es Verzeichnisse, die eine große Anzahl von Bildern beziehungsweise Fotos enthalten. Oft werden diese Bilder aus dem Internet herauskopiert oder selbst mit der digitalen Kamera fotografiert. Die Namen der einzelnen Bilder liegen oft 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 beziehungsweise 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 die Prozedur aus *Listing 52*.

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

Sub BilderBenennenUndNummerieren()
  Dim strDateiname As String
  Dim intZ As Integer

  intZ = 1
  ChDir "C:\Eigene Dateien\Kopie\"
  strDateiname = Dir$("*.jpg")
```

Listing 52: Dateien umbenennen und durchnummerieren

```
  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
End Sub
```

Listing 52: Dateien umbenennen und durchnummerieren (Forts.)

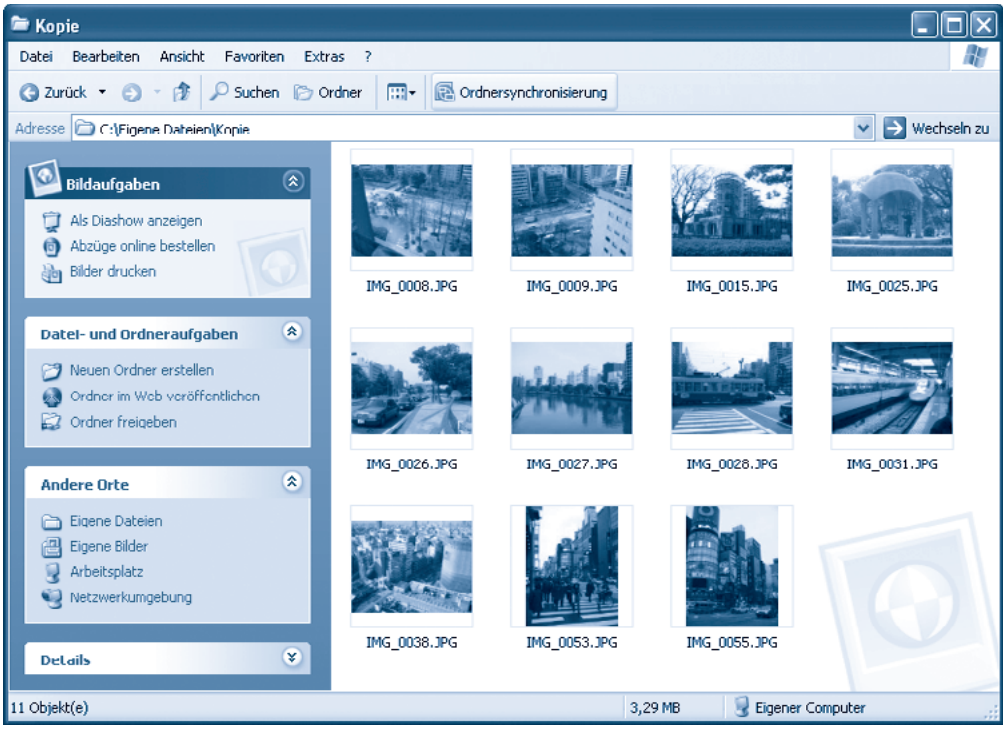


Abbildung 23: Diese Dateien sollen umbenannt und durchnummeriert werden.

Zu Beginn der Prozedur werden zwei Variable deklariert. In der Variablen `strDateiname` werden im weiteren Verlauf der Prozedur 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 geben Sie im ersten Argument der Anweisung `FileCopy` den Namen des Bildes an, das Sie kopieren möchten. Diese Information steht direkt in der Variablen `Dateiname` zur Verfügung. Im zweiten Argument wird der 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 Bildes 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, müssen Sie über die Funktion `Instr` prüfen, wann die Funktion `Dir` auf die erste neue Kopie eines Bildes stößt. Dabei können Sie 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` die Prozedur 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.

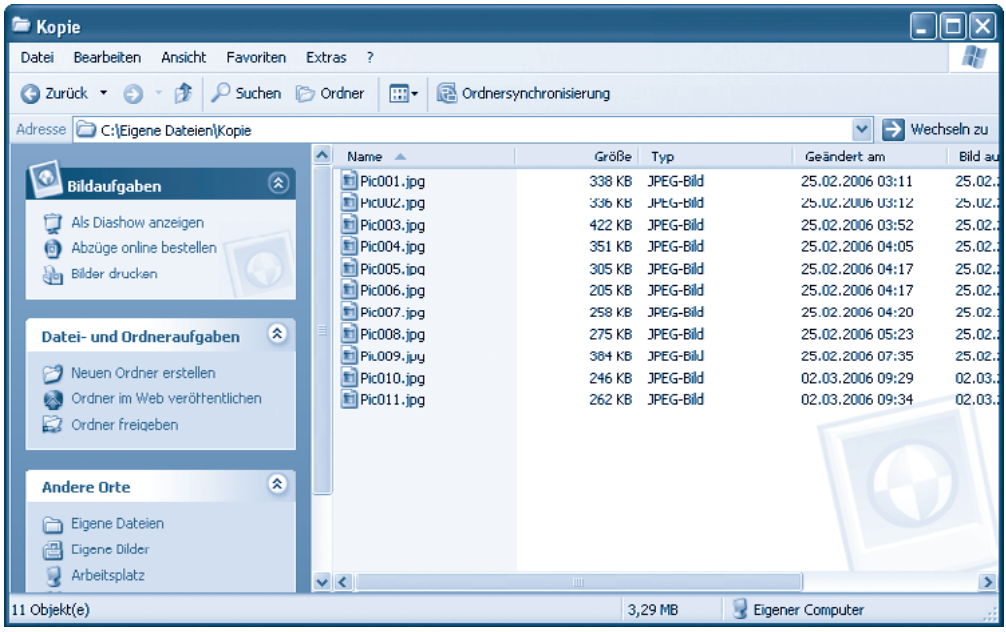


Abbildung 24: Die Bilder wurden umbenannt.

44 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

Sub DateiUmbenennen()
  Dim strAlterName As String
  Dim strNeuername As String

  strAlterName = "C:\Eigene Dateien\Artikel1.txt"
  strNeuername = "C:\Eigene Dateien\ArtikelX.txt"

  Name strAlterName As strNeuername
End Sub
```

Listing 53: Datei neu benennen über die Anweisung Name

In der Prozedur aus *Listing 53* speichern Sie den alten sowie den neuen Namen der Datei in String-Variablen. Danach wenden Sie die Anweisung `Name` an, der Sie die beiden Strings als Argumente übergeben.

45 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 Bytes 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.

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

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

Sub DateiExistenzPrüfen()
  Dim lng As Long

  On Error GoTo Fehler
  lng = FileLen("C:\Eigene Dateien\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, können Sie über die Eigenschaft `Number` den Fehler näher auswerten. Die Fehlernummer `53` steht dafür, dass eine Datei nicht gefunden werden kann.

46 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=====================================================

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

  intZ = 1
  Do Until Environ(intZ) = ""
    strText = strText & vbLf & Environ(intZ)
    intZ = intZ + 1
  Loop
  MsgBox strText
End Sub
```

Listing 55: Die Betriebssystemumgebung auslesen

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.

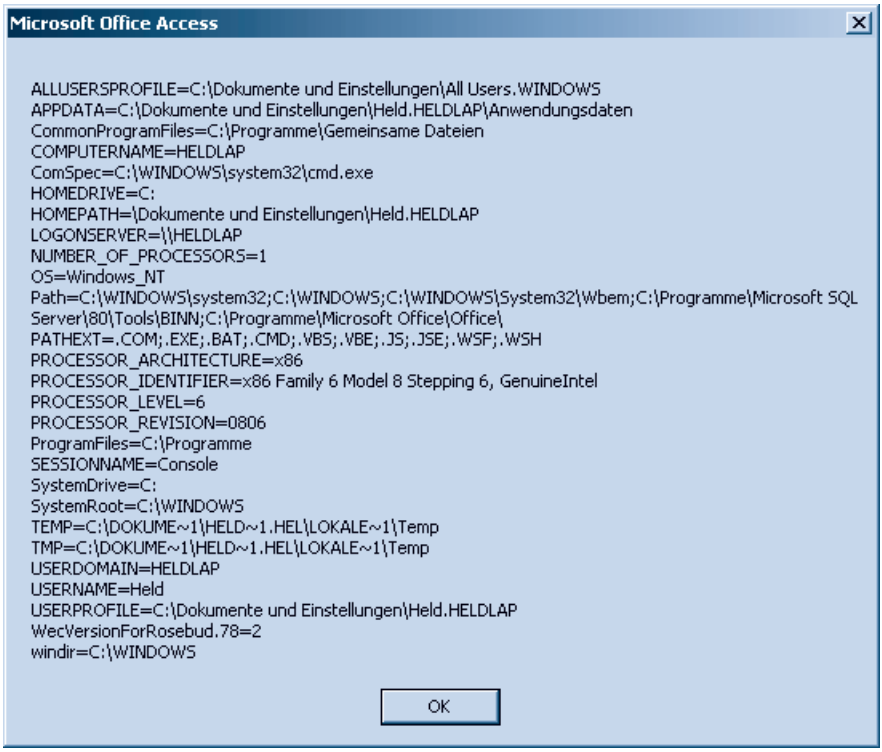


Abbildung 25: Die Betriebssystemumgebung checken

Sie können die einzelnen Betriebssystem-Umgebungsvariablen auch direkt ansprechen. In der Prozedur aus *Listing 56* wird der Benutzername sowie der Computername am Bildschirm ausgegeben.

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

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

Listing 56: Anwendername sowie Computername auslesen

Hinweis

Im *Kapitel Weitere Funktionen* dieses Buches können Sie nachschlagen, wie Sie den Anwender- sowie den Computernamen über eine API-Funktion ermitteln können.

47 Textdatei lesen

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

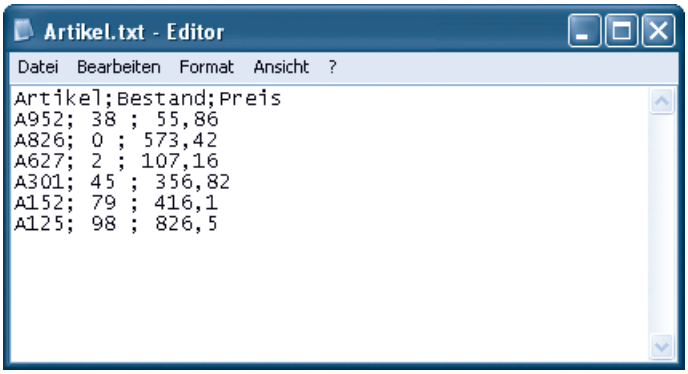


Abbildung 26: Die Ausgangsdatei

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- oder Ordner- sowie Laufwerksangaben enthalten kann.
`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 32.767 (Bytes). Bei Dateien mit wahlfreiem Zugriff ist dies die Datensatzlänge, bei sequenziellen Dateien die Anzahl der gepufferten Zeichen.

Tabelle 35: Die Argumente der Anweisung Open

Mithilfe der Anweisung `Line Input` können Sie eine einzelne Zeile aus einer geöffneten sequenziellen 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=======================================================

Sub TextdateiLesenErsterSatz()
  Dim strTextzeile As String

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

Listing 57: Erste Zeile einer Textdatei lesen

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.

48 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

49 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 ausgelesen.

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

Sub EinzelneZeichenLesen()
  Dim strZeichen As String

  Open "C:\Eigene Dateien\Artikel.txt" For Input As #1
  strZeichen = Input(7, #1)
  Debug.Print strZeichen
  Close #1
End Sub
```

Listing 58: Einzelne Zeichen über die Funktion Input lesen

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

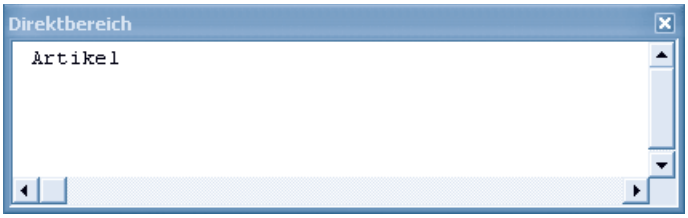


Abbildung 27: Nur wenige Zeichen einer Textdatei werden gelesen.

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

50 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=======================================================

Sub TextdateiEinlesenKomplett()
  Dim strZeile As String

  Open "C:\Eigene Dateien\Artikel.txt" For Input As #1
  Do While Not EOF(1)
    Line Input #1, strZeile
    Debug.Print strZeile
  Loop
  Close #1
End Sub
```

Listing 59: Textdatei komplett einlesen

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, um die Zeile der Textdatei in der String-Variablen `strZeile` zwischenzuspeichern. Die jeweiligen 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)` und `Chr(10)`) gelesen wird. Die Folge aus Wagenrücklaufzeichen und Zeilenvorschubzeichen wird übersprungen und nicht an die gelesene Zeichenfolge angehängt.

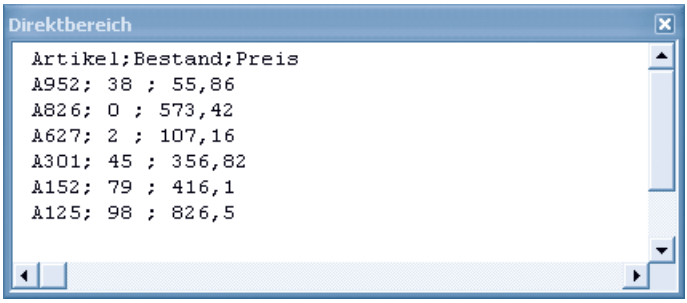


Abbildung 28: Die komplette Textdatei wurde eingelesen.

51 Textdatei schreiben

Die Anweisung `Print` schreibt Daten, die für die Ausgabe formatiert sind, in eine sequenzielle 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 und das aktuelle Datum sowie der Anwendername erfasst. Anschließend wird die Textdatei wieder geschlossen.

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

Sub TextdateiSchreiben()
  Open "C:\Eigene Dateien\user.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. Mit der Funktion `Tab` können Sie die einzelnen Informationen durch eine Tabulatorschrittweite voneinander trennen.

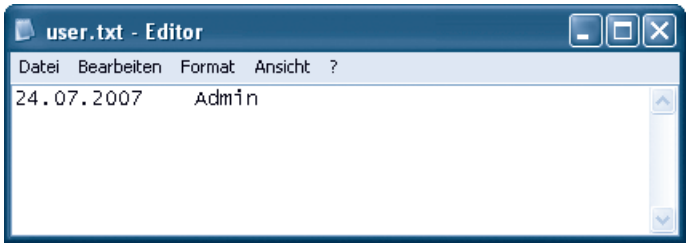


Abbildung 29: Die Textdatei wurde geschrieben.

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], Variablenname
```

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

Tabelle 39: Die Argumente der Anweisung Put

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

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

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

Listing 61: Daten aus einer Variablen in eine Textdatei schreiben

```
Sub TextdateiFüllen()
  Dim typDatensatz As MeinDatensatz
  Dim intNr As Integer
  Dim strDateiName As String

  strDateiName = "C:\Eigene Dateien\ArtikelNeu.txt"

  If Dir(strDateiName) <> "" Then
    Kill strDateiName
  End If
  intNr = FreeFile
  Open strDateiName For _
  Random As #intNr Len = Len(typDatensatz)
  With typDatensatz
    .NName = "Vorname"
    .VName = "Nachname"
  End With
  Put #intNr, , typDatensatz
  Close #intNr
End Sub
```

Listing 61: Daten aus einer Variablen in eine Textdatei schreiben (Forts.)

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.

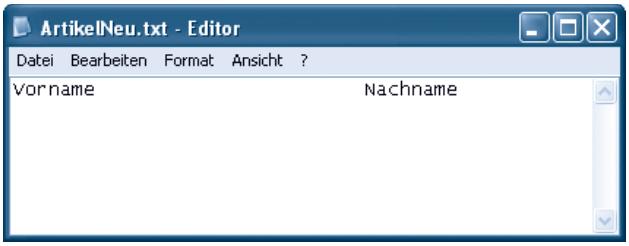


Abbildung 30: Variableninhalte in Textdatei schreiben

52 Textdateien im Batch erstellen

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

Mit der Anweisung `Write` können Sie Daten in eine sequenzielle 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. Eine 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 können.

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

Sub TextdateienImBatch()
  Dim intDateiNr As Integer

  ChDir "C:\Eigene Dateien\"
  For intDateiNr = 1 To 3
    Open "Datei" & intDateiNr & ".txt" _
      For Output As intDateiNr
    Write #intDateiNr, Now
  Next intDateiNr
  Reset
End Sub
```

Listing 62: Mehrere Dateien erstellen und schreiben

Wechseln Sie zunächst über die Anweisung `ChDir` in das gewünschte Verzeichnis, in das Sie die Textdateien schreiben möchten. Setzen Sie danach eine Schleife auf, die genau drei Mal 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.

53 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`, der die Art der zurückzugebenden Informationen festlegt: Geben Sie `1` an, um einen Wert zur Kennzeichnung des Zugriffsmodus zurückzugeben. Ausschließlich auf 16-Bit-Systemen können Sie `2` angeben, um die Dateizugriffsnummer des Betriebssystems zurückzugeben. Auf 32-Bit-Systemen wird `returntype = 2` nicht unterstützt und verursacht daher einen Fehler. Bei Betriebssystemen ab Windows 2000 geben Sie also hier immer `1` an.

Tabelle 41: Die Argumente der Funktion FileAttr

Übergeben Sie dem Parameter `returntype` den Wert `1`, sind bei der Funktion `FileAttr` die folgenden Rückgabewerte möglich, welche die in der Tabelle dargestellten Zugriffsmodi repräsentieren.

Wert	Zugriffsmodus
0	`Input`
1	`Output`
2	`Random`
8	`Append`
32	`Binary`

Tabelle 42: Die Argumente der Anweisung GetAttr

Die Anweisung `GetAttr` gibt einen Wert vom Typ `Integer` zurück, der die Attribute einer Datei oder 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 43: Die Argumente der Anweisung GetAttr

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

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

Sub DateiAttributAuslesen()
  Dim attr As VbFileAttribute

  attr = GetAttr("C:\Windows\WindowsShell.Manifest")
  If vbHidden And attr 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 44: 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 45: 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
' Dateiname  Funktionen.mdb
' Modul      MdlFile
'=======================================================

Sub DateiAttributSetzen()
  SetAttr "C:\Eigene Dateien\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.

54 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 Befehlszeilen-Optionen. 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 46: Die Argumente der Funktion Shell

Die Werte des benannten Arguments `windowstyle` lauten:

Konstante	Wert	Beschreibung
`vbHide`	0	Das Fenster ist ausgeblendet und erhält den Fokus
`vbNormalFocus`	1	Das Fenster hat den Fokus, und die ursprüngliche Größe und Position wird 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 47: Die Einstellungen für das Argument windowstyle

Microsoft Editor aufrufen

Im folgenden Beispiel aus *Listing 65* wird der in Windows enthaltene Microsoft Editor gestartet, wobei gleichzeitig eine Datei geöffnet wird.

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

Sub ExternesProgrammAufrufen()
  Dim var As Variant

  var = Shell("C:\Windows\Notepad.exe C:\Eigene Dateien\Artikel.txt", _
    vbMaximizedFocus)
End Sub
```

Listing 65: Ein externes Programm aufrufen

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 Pfades an. Über das letzte Argument legen Sie fest, in welchem Fensterstil die Anwendung angezeigt werden soll. Dabei steht die Konstante `vbMaximizedFocus` 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 gleich ein bestimmtes Verzeichnis ausgewählt.

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

Sub WindowsExlorerAufrufen()
  On Error GoTo Fehler
  Shell "C:\windows\explorer.exe C:\Eigene Dateien", vbNormalFocus
  Exit Sub

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

Listing 66: Den Windows-Explorer aufrufen und ein Verzeichnis voreinstellen – Variante 1

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

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 ein Verzeichnis einstellen – Variante 2

In beiden Prozeduren geben Sie das gewünschte Verzeichnis, das im Windows-Explorer angezeigt werden soll, gleich im Anschluss an die ausführbare Datei (.EXE) an.

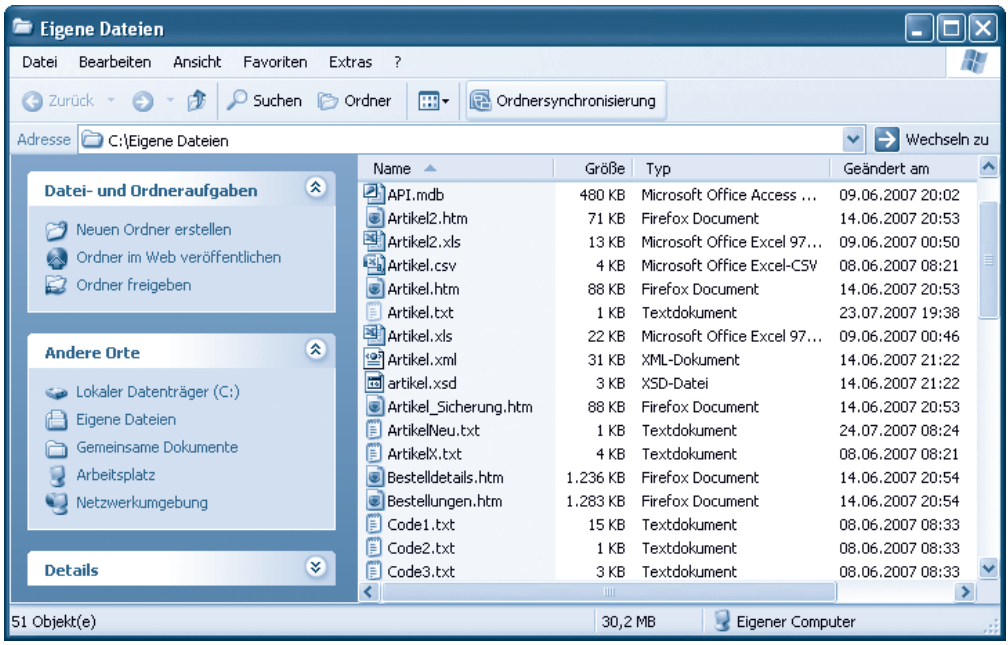


Abbildung 31: Der Explorer wurde im angegebenen Verzeichnis geöffnet.

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

Sub InternetExplorerAufrufen()
  On Error GoTo Fehler
  Shell "c:\Programme\Internet Explorer\iexplore.exe " & _
    "C:\Eigene Dateien\IMG_0006.JPG", vbMinimizedFocus
  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.

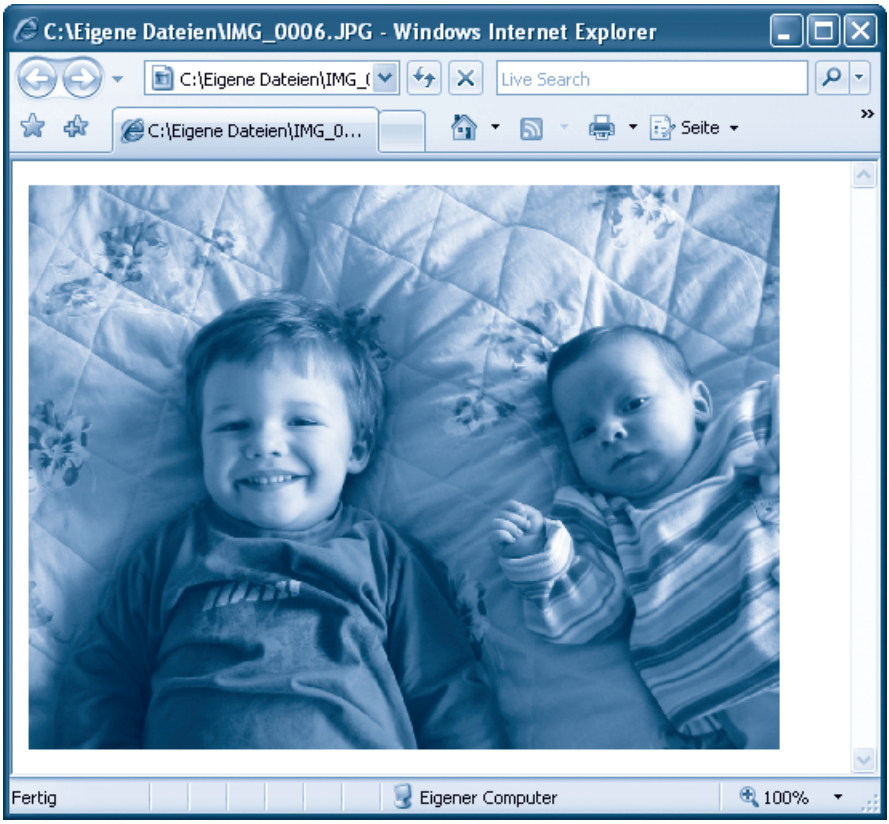


Abbildung 32: Das Bild wurde im Internet Explorer geöffnet.

Mathematische Funktionen

Zur letzten Kategorie an Funktionen in diesem Kapitel gehören die mathematischen Funktionen.

55 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.

56 Vorzeichen auslesen

Die Funktion `Sgn` gibt das Vorzeichen eines Wertes als Faktor (–1, 0 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 48: 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 49* entnehmen können.

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

57 Abschreibung errechnen (degressiv)

Soll die Abschreibung eines Gutes 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 Vermögenswertes angibt.
`salvage`	Erforderlich. Ein Wert vom Typ `Double`, der den Wert des Vermögenswertes am Ende seiner Nutzungsdauer angibt.
`life`	Erforderlich. Ein Wert vom Typ `Double`, der die Länge der Nutzungsdauer des Vermögenswertes angibt.
`period`	Erforderlich. Ein Wert vom Typ `Double`, der den Zeitraum angibt, für den die Abschreibung des Vermögenswertes 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 49: Die Argumente der Funktion DDB

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

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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
  curEndWert = 1
  For intPeriode = 1 To intDauer
    Debug.Print "Jahr " & intPeriode & ": " & _
```

Listing 70: Die degressive Abschreibung berechnen

```
      DDB(curWert, curEndWert, intDauer, intPeriode)
  Next intPeriode
End Sub
```

Listing 70: Die degressive Abschreibung berechnen (Forts.)

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

Abbildung 33: Die degressive Abschreibung wurde errechnet.

58 Abschreibung errechnen (linear)

Soll die Abschreibung eines Gutes 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 Vermögenswertes 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 Vermögenswertes angibt.

Tabelle 50: Die Argumente der Funktion SLN

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

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 diese mit den Parametern, die für die Abschreibung gefordert werden. In einer `For Next`-Schleife ermitteln Sie mithilfe der Funktion `SLN` die jeweiligen Abschreibungsbeträge und geben sie im Direktfenster der Entwicklungsumgebung aus.

Abbildung 34: Die lineare Abschreibung wurde errechnet.

59 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 Funktionen `Fix` und `Int` geben den Vorkommawert einer Zahl zurück.

```
Fix(Zahl)
```

```
Int(Zahl)
```

Der Unterschied zwischen `Int` und `Fix` besteht darin, dass bei einem negativen Wert von `Zahl` `Int` die erste negative ganze Zahl zurückgibt, die kleiner oder gleich `Zahl` ist, während `Fix` die erste negative ganze Zahl zurückgibt, die größer oder gleich `Zahl` ist.

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
' 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

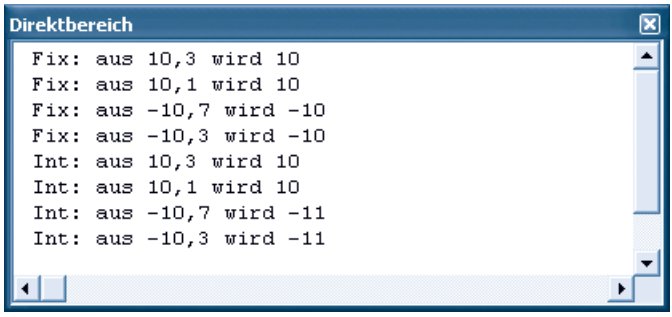


Abbildung 35: Die beiden Funktionen liefern zum Teil unterschiedliche Ergebnisse.

60 Zahlen runden

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

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

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 51: Die Argumente der Funktion Round

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

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

61 Zinssatz errechnen

Wird ein Kredit angenommen, bei dem 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 fester Anzahl von 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 52: 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 zurückgezahlt sein. Monatlich werden jeweils 650 Euro abbezahlt. Wie hoch ist nun der Zinssatz?

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

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

Listing 74: Zinssatz errechnen über die Funktion Rate

```
  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 diese mit den Parametern, die für die Zinssatzberechnung gefordert werden. Übergeben Sie diese der Funktion `Rate`. Geben Sie danach das Ergebnis am Bildschirm aus.

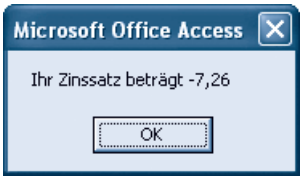


Abbildung 37: Das Ergebnis der Zinsrechnung

62 Ansparungsbetrag errechnen

Wenn Sie einen monatlichen Betrag 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.

Tabelle 53: Die Argumente der Funktion FV

Teil	Beschreibung
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 53: Die Argumente der Funktion FV (Forts.)

Beim folgenden Beispiel aus *Listing 75* wird errechnet, wie sich ein angelegtes Kapital von 10.000 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
' 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

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

Listing 75: Das Endkapital wurde berechnet.

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.

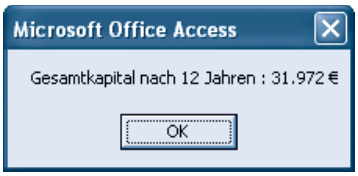


Abbildung 38: Das formatierte Endergebnis

63 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 zufällige Kommazahl zwischen 0 und 1 vom Datentyp `Single`.

  ```
  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

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

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

Sub ZufallszahlenErzeugen()
  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

Über die Anweisung `Randomize` initialisieren Sie den Zufallsgenerator. Danach setzen Sie eine Schleife auf, die genau zehn 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, Ganzzahlwert gemacht.

Abbildung 39: Zufallszahlen wurden erzeugt.

Zufallsbuchstaben zwischen A und Z erzeugen

Wenn Sie die Prozedur 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
' 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 zehn Mal durchlaufen wird. Innerhalb der Schleife erstellen Sie Zufallsbuchstaben. Dabei wenden Sie die Funktion `Chr` an, um den Zahlenwert in einen Buchstaben umzuwandeln.

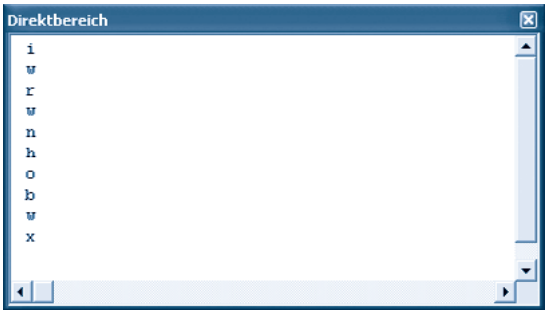


Abbildung 40: Die Zufallsbuchstaben wurden erzeugt.

Prüffunktionen

Abschließend gibt es noch die Prüffunktionen, über die Sie beispielsweise prüfen können, welcher Datentyp vorliegt.

64 Eingabe prüfen

Die Prüffunktionen kommen immer dann zum Einsatz, wenn Sie kontrollieren möchten, welche Eingabe eine 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
' 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 o.k.!"
    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, wenden Sie die Funktion `IsNumeric` an, um zu testen, ob eine korrekte Zahl eingegeben wurde.

65 Datenfeld-Check

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

Um diese Funktion zu üben, schreiben Sie eine Prozedur, in der 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
' Dateiname  Funktionen.mdb
' Modul      MdlPrüf
'=======================================================

Function ArrayBilden(intA As Integer) As Variant
  ReDim avarArray(intA)
  Dim intZ As Integer

  For intZ = LBound(avarArray, 1) To UBound(avarArray, 1)
    avarArray(intZ) = intZ
  Next intZ
  ArrayBilden = avarArray
End Function


Sub DynamischesArrayBilden()
  Dim avar As Variant
  Dim intZ As Integer

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

Listing 79: Datenfelder bilden und prüfen

In der Zeile `avar=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, und zwar mit den Indizes 0 bis 9. Nachdem die Funktion das angelegte Datenfeld an die aufrufende Prozedur zurückgemeldet hat, 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 Datenfeldes und arbeiten Sie sich dann bis zum letzten Feld des Datenfeldes vor, das Sie über die Funktion `UBound` ermitteln. Geben Sie nach jedem Schleifendurchlauf den Inhalt des jeweiligen Datenfeldes im Direktbereich über die Anweisung `Debug.Print` aus.

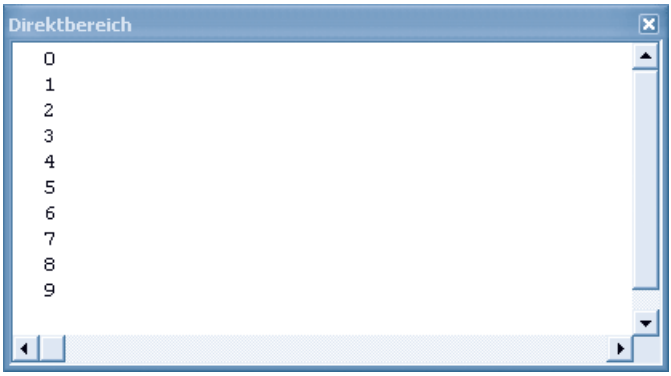


Abbildung 41: Das Datenfeld wurde angelegt.

66 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. Anschließend soll diese Tabelle auch geöffnet werden. Diese Prüfroutine sieht wie folgt aus:

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

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

Listing 80: Tabellenexistenz prüfen

```
Sub PrüfenAufTabelle()
  If TabelleExistiert("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 eine Tabelle, mit dem in der Variablen `strTab` übergebenen Namen, 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.

Umwandlungsfunktionen

Hin und wieder müssen Sie einzelne Datentypen in andere Datentypen umwandeln. Für diesen Zweck gibt es eine ganze Reihe von Umwandlungsfunktionen.

67 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
' Dateiname  Funktionen.mdb
' Modul      MdlKonv
'=======================================================

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

Listing 81: Punkte aus Zeichenfolge eliminieren

```
  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 (Forts.)

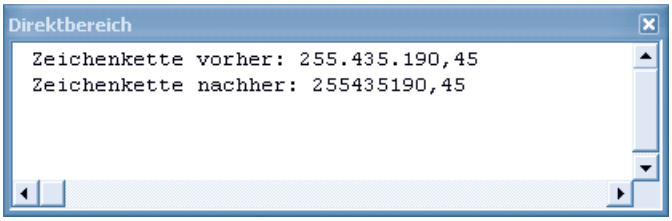


Abbildung 42: Die Punkte wurden entfernt.

68 Datumswerte umwandeln

Mithilfe der Typumwandlungsfunktion `CDate` können Sie Datumsangaben in Zeichenfolgen in echte Datumsangaben umwandeln. So wird beispielsweise aus der Zeichenfolge 21. November 2008 das Datum 21.11.2008.

Sehen Sie weitere Beispiele im *Listing 82*.

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

Sub DatumswerteWandeln()
  Const Datum1 = "11/25/2008 8:30 AM"
  Const Datum2 = "12. Februar 2008  "
  Const Datum3 = "Januar, 2008      "

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

Listing 82: Zeichenfolgen in Datumswerte umwandeln

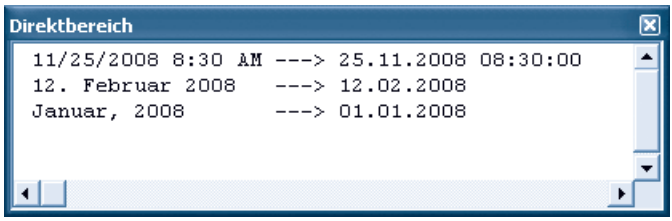


Abbildung 43: Die Textfolgen wurden erfolgreich umgewandelt.

69 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 10.000 und 99.999 gebildet. Diese werden mithilfe der Funktion `CLng` in den richtigen Datentyp umgewandelt.

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

Sub NachkommastellenWeg()
  Dim intZ As Integer
  Dim lng As Long
  Dim dbl As Double

  For intZ = 1 To 10
    dbl = 90000 * Rnd + 10000
    lng = CLng(dbl)

    Debug.Print "Zufallszahl " & Format(intZ, "00") & _
      "   Vorher: " & dbl & " Nachher: " & _
      Format(lng, "0,###")
  Next intZ
End Sub
```

Listing 83: Zufallszahlen ohne Nachkommastellen bilden

Über die Anweisung `dbl(90000 * Rnd + 10000)` bilden Sie eine Zufallszahl zwischen 10.000 und 99.999 mit Nachkommastellen und weisen diese Zahl zunächst einer Variable vom Typ `Double` zu. Anschließend machen Sie eine Typumwandlung in der Zeile `lng = CLng(dbl)` wobei die Zahl gleichzeitig korrekt gerundet wird. Über die Funktion `Format` bringen Sie die Werte in die gewünschte Form und geben zur Demonstration im Direktfenster die Zahlen jeweils im Format vor und nach der Umwandlung aus.

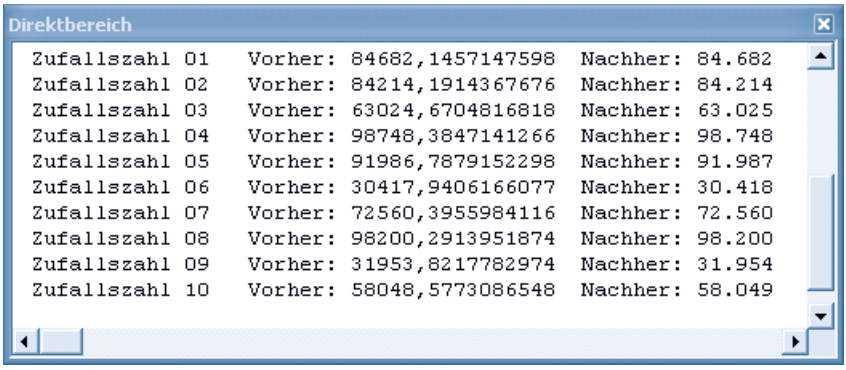


Abbildung 44: Zufallszahlen mit Nachkommastellen wurden umgewandelt und formatiert.

70 Zahl aus Zeichenfolge extrahieren

Mithilfe der Funktion `Val` können Sie aus einer Zeichenfolge darin enthaltene numerische Werte herausholen.

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

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

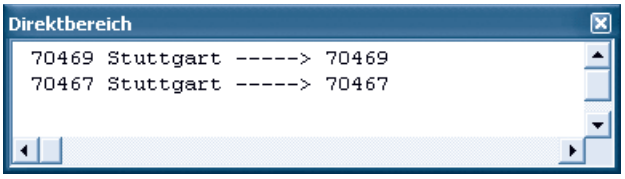


Abbildung 45: Die PLZ wird extrahiert.

71 Zahlen in Texte umwandeln

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

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

Sub ZahlInText()
  Dim strText As String

  strText = str(679)     ' Liefert " 679".
  strText = str(-679.34) ' Liefert "-679.34".
  strText = str(123.986) ' Liefert " 123.986".
EndSub
```

Listing 85: Zahlen in Textwerte umwandeln

Weitere Funktionen

In diesem Kapitel können Sie ausgesuchte API-Funktionen sowie Beispiele für eigene Funktionen nachschlagen.

API-Funktionen

Viele Aufgaben in Access müssen nicht unbedingt durch den Einsatz von VBA gelöst werden. Viele 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. Ein 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 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 in unserem Buch vorrangig um die VBA-Programmierung handeln soll, 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.

72 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
' 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 lngLWTyp As Long
  Dim strLW As String
  Dim lng As Long
  Dim lng1 As Long
  Dim strBuffer As String

  strBuffer = Space(200)
  lng = GetLogicalDriveStrings(200, strBuffer)
  If lng = 0 Then
    CdRomLWBuchstabe = vbNullString
    Exit Function
  End If
  lng1 = 1
  strLW = Mid(strBuffer, lng1, 3)
  Do While (Mid(strBuffer, lng1, 1) <> vbNullChar)
    lngLWTyp = GetDriveType(strLW)
    If lngLWTyp = 5 Then
      CdRomLWBuchstabe = strLW
      Exit Function
    End If
    lng1 = lng1 + 4
    strLW = Mid(strBuffer, lng1, 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 meisten 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 den Sie alle logischen Laufwerksbezeichnungen Ihres Systems einlesen. Danach rufen Sie die API-Funktion `GetLogicalDriveStrings` auf, die Ihnen 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`, die 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 `lng`. 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 `strLW`. Setzen Sie nun eine Schleife auf, die den Puffer Stück für Stück ausliest. Innerhalb der Schleife wenden Sie die API-Funktion `GetDriveType` an. Diese API-Funktion liefert den Index für das Laufwerk in der Variablen `strLW` zurück und weist diesen der Variable `lngLWTyp` zu. Jeder Laufwerkstyp weist einen eindeutigen Laufwerksindex auf. Diese Indizes können Sie der *Tabelle 54* entnehmen.

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

Tabelle 54: Die Laufwerksindizes für die API-Funktion GetDriveType

Rufen Sie nun die API-Funktion über die Prozedur aus *Listing 87* auf.

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

Sub CDRomLW()
  Dim str As String

  str = CdRomLWBuchstabe()
  MsgBox "Das CD-ROM-Laufwerk hat den Buchstaben " & str
End Sub
```

Listing 87: CD-ROM-Laufwerk ermitteln

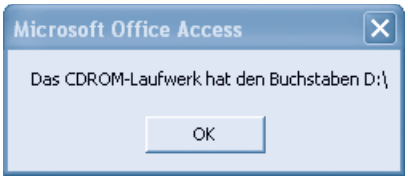


Abbildung 46: Das CD-ROM-Laufwerk wurde ermittelt.

73 Den 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 nutzen die API-Funktion `GetUserName`.

Erfassen Sie jetzt die API-Funktion aus *Listing 88*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 lng As Long
  Dim str As String

  Const D_Puffer = 255
```

Listing 88: Anwendername ermitteln

```
  str = Space(D_Puffer)
  lng = D_Puffer
  If CBool(GetUserName(str, lng)) Then
    AnwenderName = Left$(str, lng - 1)
  Else
    AnwenderName = ""
  End If
End Function

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

Listing 88: Anwendername 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 angehängte 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: Anwendername anzeigen

74 Den Computernamen ermitteln

Den Computernamen können Sie in der Systemsteuerung von Windows unter NETZWERK herausfinden. Automatisiert über VBA geht dies mit der API-Funktion aus *Listing 89*.

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

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

Listing 89: Computername ausgeben

```
Sub ComputerNameAnzeigen()
 Dim strString As String

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

Listing 89: Computername ausgeben (Forts.)

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

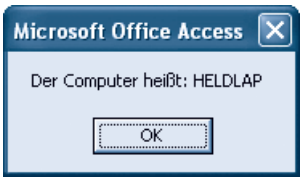


Abbildung 48: Den Computernamen auslesen

75 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
' 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
```

Listing 90: CD-ROM-Laufwerk öffnen und schließen

```
End Sub

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

Listing 90: CD-ROM-Laufwerk öffnen und schließen (Forts.)

76 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
' 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) & BildschirmAuflösung()
End Sub

Function BildschirmAuflösung()
  Dim lngRval As Long
  Dim lngDc As Long
  Dim lngHSize As Long
  Dim lngVSize As Long

  lngDc = GetDC(0&)
  lngHSize = GetDeviceCaps(lngDc, HORZRES)
  lngVSize = GetDeviceCaps(lngDc, VERTRES)
  lngRval = ReleaseDC(0, lngDc)
  BildschirmAuflösung = lngHSize & "x" & lngVSize
End Function
```

Listing 91: Bildschirmauflösung ermitteln

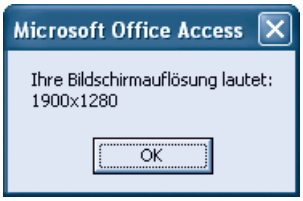


Abbildung 49: Bildschirmauflösung ermitteln

77 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlFindW
'=======================================================

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

Sub MailProgrammAktiv()
  Dim lngFenster As Long

  lngFenster = FindWindow(vbNullString, "Microsoft Outlook")
  If lngFenster = 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.

78 Externes Programm aufrufen

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

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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()
  Dim lngWnd As Long
  Const SW_SHOWNORMAL = 1

  ShellExecute lngWnd, "open", "calc.exe", 0, 0, SW_SHOWNORMAL
End Sub
```

Listing 93: Ein externes Programm starten

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

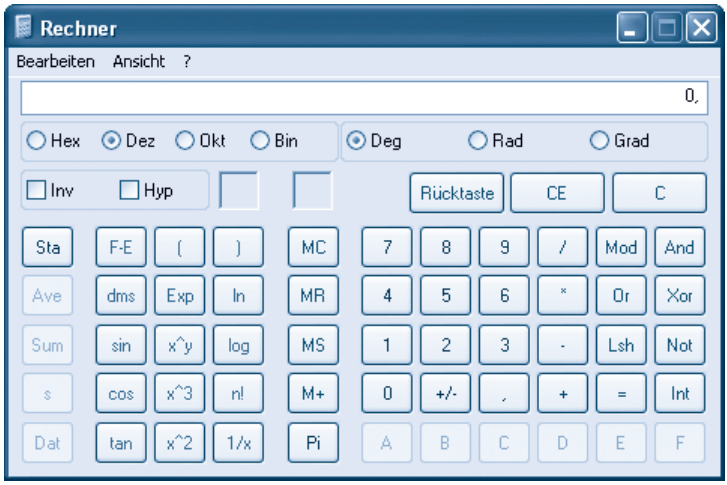


Abbildung 50: Der Microsoft Rechner wurde gestartet.

79 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlOpenProc
'=====================================================

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 dblBeginn As Double
  Dim lngTaskId As Long
  Dim lngProzess As Long
  Dim lngEnde As Long

  dblBeginn = Now
  Debug.Print "Start des Programms um: " & Now
  lngTaskId = Shell("sol.exe", 1)
  lngProzess = _
    OpenProcess(PROCESS_QUERY_INFORMATION, False, lngTaskId)
  MsgBox "Zeitnahme läuft..."
  Do
    GetExitCodeProcess lngProzess, lngEnde
    DoEvents
  Loop While lngEnde = STILL_ACTIVE
  lngEnde = Now
  Debug.Print "Die Anwendung wurde um " & Now & " beendet!"
  Debug.Print "Das Programm lief genau: " & _
  Format(lngEnde - dblBeginn, "s") & " Sekunden!"
End Sub
```

Listing 94: Wie lange läuft ein gestartetes Programm?

Speichern Sie gleich zu Beginn der Prozedur die aktuelle Uhrzeit mithilfe der Funktion `Now` in der Variablen `dblBeginn`. 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 55*. Der Rückgabewert der Funktion `Shell` stellt die sogenannte Task-ID des gestarteten Prozesses dar. Speichern Sie diesen Wert in der Variablen `lngTaskId`.

Index	Beschreibung
0	Das Fenster der aufgerufenen Anwendung ist ausgeblendet und 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 55: 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. Dieser Funktion müssen Sie den zuvor in der Variablen `lngTaskId` gespeicherten Wert als Parameter übergeben. Den Rückgabewert dieser Funktion speichern Sie in einer Variablen `lngProzess` 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. Diese Funktion benötigt als Parameter den in der Variablen `lngProzess` zuvor gespeicherten Prozess `Handle`. Mithilfe der Funktion `DoEvents` übergeben Sie innerhalb der Schleife immer wieder die Steuerung an das Betriebssystem, damit andere Ereignisse verarbeitet werden können. Somit läuft diese Schleife im Hintergrund 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.

80 Eine Wartezeit erzeugen

Möchten Sie innerhalb eines Programms eine Wartezeit erzeugen, dann können Sie die API-Funktion aus *Listing 95* einsetzen.

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

Declare Sub Sleep Lib "kernel32.dll" _
  (ByVal SleepTime As Long)

Sub WartezeitErzeugen()
  Sleep (10000)
  MsgBox "Ruhepause beendet"
End Sub
```

Listing 95: API-Funktion zum Erzeugen einer Wartezeit

Die Zeitangabe wird in Millisekunden angegeben. Demnach wird in diesem Beispiel 10 Sekunden gewartet.

81 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlCreateD
'=====================================================

Private Declare Function CreateDirectory Lib _
 "kernel32" Alias "CreateDirectoryA" _
 (ByVal lpPathName As String, _
 lpSecurityAttributes As SECURITY_ATTRIBUTES) As Long

Private Type SECURITY_ATTRIBUTES
  nLength As Long
  lpSecurityDescriptor As Long
```

Listing 96: Verzeichnis erstellen mit einer API-Funktion

```
  bInheritHandle As Long
End Type

Sub VerzeichnisErstellen()
  Dim Security As SECURITY_ATTRIBUTES
  Dim lng As Long
  Dim str As String

  str = InputBox("Geben Sie das Verzeichnis ein", _
    "Verzeichnis erstellen", "C:\Test")
  If str = "" Then Exit Sub

  lng = CreateDirectory(str, Security)
  If lng = 0 Then _
    MsgBox "Das Verzeichnis konnte nicht erstellt werden!", _
      vbCritical + vbOKOnly
End Sub
```

Listing 96: Verzeichnis erstellen mit einer API-Funktion (Forts.)

Mit der API-Funktion `CreateDirectory` können Sie eigene Verzeichnisse auf Ihrer Festplatte anlegen.

82 Verzeichnis löschen

Um ein Verzeichnis zu entfernen können Sie die API-Funktion `RemoveDirectory` aus *Listing 97* einsetzen.

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

Private Declare Function RemoveDirectory Lib _
 "kernel32" Alias "RemoveDirectoryA" _
 (ByVal lpPathName As String) As Long

Sub VerzeichnisLöschen()
  Dim lng As Long
  lng = RemoveDirectory("C:\Test")
End Sub
```

Listing 97: Verzeichnis löschen mit einer API-Funktion

83 Verzeichnisbaum anzeigen und auswerten

Im folgenden Beispiel aus *Listing 98* wird Ihre Verzeichnisstruktur in einem Dialogfeld als Verzeichnisbaum angezeigt und ausgewertet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 briInfo As BROWSEINFO
  Dim strPath As String
  Dim lng As Long

  briInfo.pidlRoot = 0&
  lng = SHBrowseForFolder(briInfo)
  strPath = Space$(512)
  If SHGetPathFromIDList(ByVal lng, ByVal strPath) Then
    VerzeichnisErmitteln = Left(strPath, InStr(strPath, _
    Chr$(0)) - 1)
  Else
```

Listing 98: Verzeichnis auswählen und abfragen

```
    VerzeichnisErmitteln = ""
  End If
End Function

Sub VerzeichnisBaum()
  Dim strVerz As String

  strVerz = VerzeichnisErmitteln(strVerz)
  If strVerz = "" Then Exit Sub
    MsgBox "Das ausgewählte Verzeichnis lautet: " & strVerz
End Sub
```

Listing 98: Verzeichnis auswählen und abfragen (Forts.)

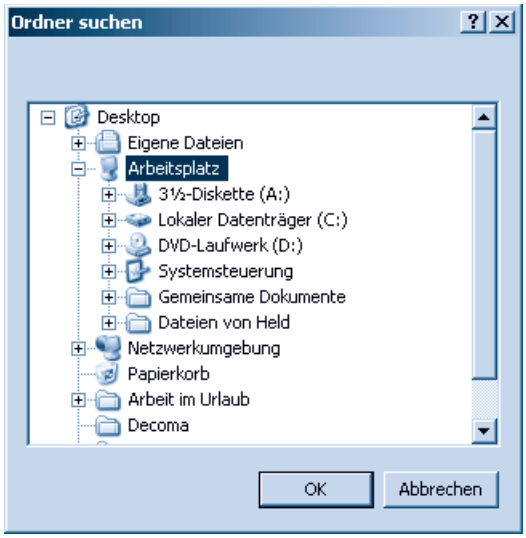


Abbildung 51: Den Verzeichnisbaum anzeigen

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

84 Windows-Version ermitteln

Um die installierte Windows-Version zu ermitteln, können Sie mit der API-Funktion `GetVersionEx` aus *Listing 99* arbeiten.

```
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

Private Sub WindowsVersionBestimmen()
  Dim osVerInfo As OSVERSIONINFO
  Dim lngSucc As Long

  ' dwOSVersionInfoSize mit Größe der OSVERSIONINFO-Struktur initialisieren
  osVerInfo.dwOSVersionInfoSize = Len(osVerInfo)
  ' Windows-Versionsinfo abrufen
  lngSucc = GetVersionEx(osVerInfo)

  If lngSucc = 0 Then Exit Sub

  Select Case osVerInfo.dwPlatformId ' Plattform-Kennzeichner
    Case VER_PLATFORM_WIN32s ' Win32s-Erweiterung
      Debug.Print "Win32s für Windows 3.x"
    Case VER_PLATFORM_WIN32_WINDOWS ' Windows 9x/Me
      Select Case osVerInfo.dwMinorVersion
        Case 0
          Debug.Print "Windows 95"
        Case 10
          Debug.Print "Windows 98"
        Case 90
          Debug.Print "Windows Me"
```

Listing 99: Die installierte Windows-Version erkennen

```
      End Select
    Case VER_PLATFORM_WIN32_NT ' Windows NT/2000/XP
      Select Case osVerInfo.dwMajorVersion
        Case 3
          Debug.Print "Windows NT 3." & CStr(osVerInfo.dwMinorVersion)
        Case 4
          Debug.Print "Windows NT 4.0"
        Case 5
          Select Case osVerInfo.dwMinorVersion
            Case 0
              Debug.Print "Windows 2000"
            Case 1
              Debug.Print "Windows XP"
            Case 2
              Debug.Print "Windows 2000 Server"
          End Select
        Case 6
          Select Case osVerInfo.dwMinorVersion
            Case 0
              Debug.Print "Windows Vista"
            Case Else
              Debug.Print "Windows Vista Folgeversion"
          End Select
        Case Else 'Zukünftige Windows Version
          Debug.Print "Windows Vista Folgeversion"
      End Select
  End Select

  Debug.Print "Version: " & osVerInfo.dwMajorVersion & "." & _
    osVerInfo.dwMinorVersion
  Debug.Print "Build: " & osVerInfo.dwBuildNumber
  Debug.Print "Info: " & osVerInfo.szCSDVersion
End Sub
```

Listing 99: Die installierte Windows-Version erkennen (Forts.)

Die neueren auf NT-Technologie basierenden Windows-Versionen liefern über die Eigenschaft `dwplatformId` den Wert `2` zurück. Darunter werten Sie die Argumente `dwMinorVersion` und `dwMajorVersion` aus, um das Betriebssystem zu bestimmen. Die nach heutigem Stand höchste Versionsnummer besitzt Windows Vista mit der Versionsnummer 6.0.

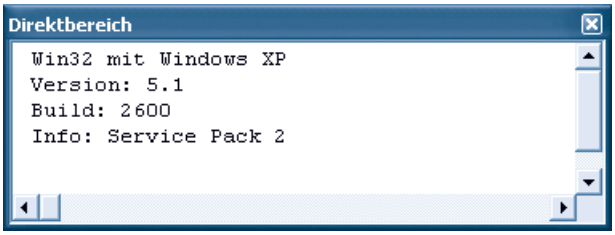


Abbildung 52: Die Windows-Version ermitteln

85 Windows-Verzeichnis ermitteln

Der Name des Verzeichnisses in dem Windows auf dem PC installiert wird, kann im Prinzip bei der Installation des Betriebssystems beliebig gewählt werden. Selbst wenn jeweils das Standardverzeichnis bei der Installation gewählt wird, gibt es Unterschiede bei den einzelnen Windows-Versionen. Möchten Sie in Ihrer Anwendung bestimmen in welchem Verzeichnis Windows auf einem PC installiert ist, können Sie folgende API-Funktion aus *Listing 100* einsetzen.

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

Declare Function GetWindowsDirectory Lib _
  "kernel32" Alias "GetWindowsDirectoryA" _
  (ByVal lpBuffer As String, ByVal nSize As Long) As Long

Function GetWinDir() As String
  Dim strBuffer As String * 255
  Dim lngLength As Long

  lngLength = _
    GetWindowsDirectory(strBuffer, Len(strBuffer))
  GetWinDir = Left(strBuffer, lngLength)
End Function

Sub WindowsVerzeichnis()
  Dim str As String

  str = GetWinDir
  MsgBox "Ihr Windows-Verzeichnis heißt: " & str
End Sub
```

Listing 100: Windows-Verzeichnis ermitteln

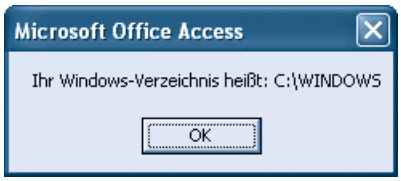


Abbildung 53: Windows-Installationsverzeichnis auslesen

86 Windows-Systemverzeichnis ermitteln

Soll das Windows-Systemverzeichnis ermittelt werden, dann setzen Sie die API-Funktion `GetSystemDirectory` aus *Listing 101* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 strBuffer As String * 255
  Dim lngLength As Long

  lngLength = _
    GetSystemDirectory(strBuffer, Len(strBuffer))
    GetSystemDir = Left(strBuffer, lngLength)
End Function

Sub WindowsSystemVerzeichnis()
  Dim str As String

  str = GetSystemDir
  MsgBox "Ihr Windows-Systemverzeichnis heißt: " & str
End Sub
```

Listing 101: Windows-Systemverzeichnis ermitteln

Abbildung 54: Das Windows-Systemverzeichnis anzeigen

87 Das temporäre Verzeichnis ermitteln

Windows legt ein temporäres Verzeichnis an, in dem Anwendungen temporäre Dateien ablegen können, die bei Bedarf geladen bzw. wieder geschlossen werden. Genau genommen wird für jeden User, der auf dem PC eingerichtet ist, ein eigenes temporäres Verzeichnis angelegt. Um das temporäre Verzeichnis des momentan angemeldeten Users zu bestimmen, setzen Sie die API-Funktion `GetTempPath` aus *Listing 102* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 strBuffer As String * 255
  Dim lng As Long

  lng = GetTempPath(Len(strBuffer), strBuffer)
  GetTempDir = Left(strBuffer, lng)
End Function

Sub TemporäresVerzeichnis()
  Dim str As String

  str = GetTempDir
  MsgBox "Ihr temporäres Verzeichnis heißt: " & str
End Sub
```

Listing 102: Das temporäre Verzeichnis ermitteln

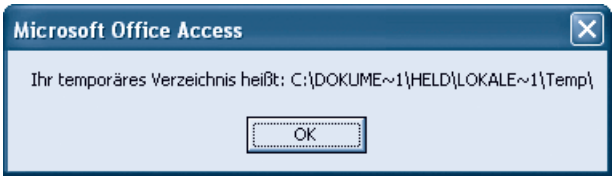


Abbildung 55: Das temporäre Verzeichnis anzeigen

88 Das aktuelle Verzeichnis ermitteln

Neben der API-Funktion `GetWindowsDirectory` gibt es weitere API-Funktionen, um bestimmte Verzeichnisse zu ermitteln. Mithilfe der Funktion `GetCurrentDirectory` aus *Listing 103* können Sie das aktuelle Verzeichnis ermitteln.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 strBuffer As String * 255
  Dim lng As Long

  lng = _
    GetCurrentDirectory(Len(strBuffer), strBuffer)
  GetAktDir = Left(strBuffer, lng)
End Function

Sub AktuellesVerzeichnis()
  Dim str As String

  str = GetAktDir
  MsgBox "Ihr aktuelles Verzeichnis heißt: " & str
End Sub
```

Listing 103: Das aktuelle Verzeichnis ermitteln

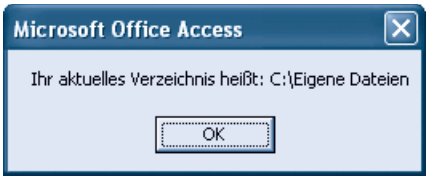


Abbildung 56: Das aktuelle Verzeichnis anzeigen

89 Windows-Info-Dialog 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 Dialogs. Die API-Funktion für diesen Zweck heißt `ShellAbout`.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 Info-Dialogs

Abbildung 57: Windows-Info-Dialoge manipulieren

90 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.

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

Declare Function GetCommandLine Lib "kernel32" _
  Alias "GetCommandLineA" () As String

Sub AccessVerzeichnisErmitteln()
  MsgBox "Access befindet sich unter: " & GetCommandLine()
End Sub
```

Listing 105: Ermitteln des Access-Startverzeichnisses

Die API-Funktion besteht lediglich aus einem Kommando, das Sie durch eine Prozedur aufrufen und über die Funktion `MsgBox` auf dem Bildschirm ausgeben.

91 Standardverzeichnis festlegen

Standardmäßig ist das Verzeichnis EIGENE DATEIEN Ihr Standardverzeichnis in Access. Hier werden Access-Datenbanken standardmäßig gespeichert bzw. wird beim Dialog ÖFFNEN immer dieses Verzeichnis eingestellt. Diese Einstellung können Sie ändern, indem Sie die API-Funktion aus *Listing 106* einsetzen.

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

92 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
' 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 gefunden.

93 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
' 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 lng As Long

  strLangerName = Application.CurrentDb.Name
  strKurzerName = Space$(250)

  lng = GetShortPathName(strLangerName, strKurzerName, Len(strKurzerName))

  strKurzerName = Left$(strKurzerName, lng)
  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 Pfades der aktuell geöffneten Datenbank. Die Variable `strKurzerName` wird zunächst mit 250 Leerzeichen initialisiert. Danach wird die API-Funktion `GetShortPathName` aufgerufen. Dieser Funktion wird sowohl die Variable `strLangerName` als auch die noch leere Variable `strKurzerName` sowie deren Länge übergeben. Die API-Funktion kürzt den Pfadnamen. Diesen geben Sie über die Funktion `MsgBox` auf dem Bildschirm aus.

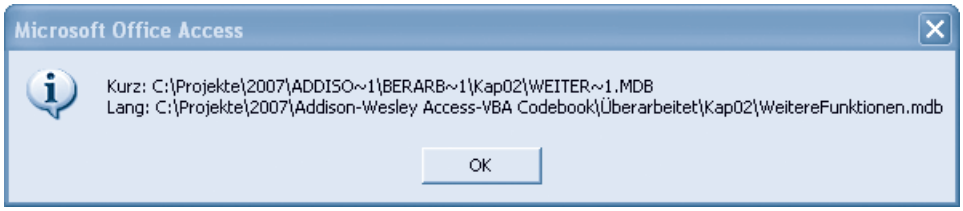


Abbildung 59: Kurze und lange Dateinamen auslesen

94 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
' 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 ok!"
  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.

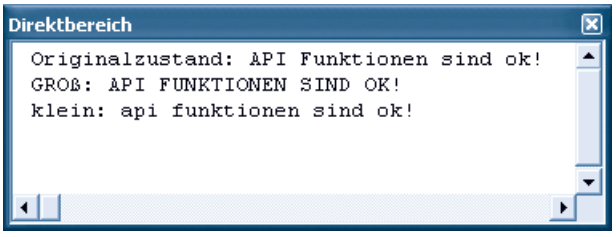


Abbildung 60: Texte über eine API-Funktion konvertieren

95 Zwischenablage löschen

Möchten Sie die Zwischenablage durch eine API-Funktion löschen, verwenden Sie die API-Funktion aus *Listing 110*.

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

96 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
' 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 bool 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

97 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
' 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:\windows\media\ringin.wav", 0)
End Sub
```

Listing 112: Sounddatei abspielen

98 PC piepsen lassen

Soll Ihr PC nur einen Piepton abgeben, dann setzen Sie die API-Funktion `Beep` aus *Listing 113* ein.

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

Declare Function Beep Lib "kernel32.dll" _
  (ByVal dwFreq As Long, ByVal dwDuration As Long) As Long

Sub ComputerPiepst()
  Dim lng As Long

  lng = Beep(250, 1000)
  If lng = 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 Hertz der Piepton erzeugt werden soll. Mögliche noch hörbare Werte liegen im Bereich zwischen 37 und 5500. Die Beschränkung liegt hierbei in den Möglichkeiten eines normalen internen PC-Lautsprechers. Im zweiten Argument legen Sie die Dauer des Pieptons in Millisekunden fest.

99 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlKeyboard
'=======================================================

Type KeyboardBytes
  kbb(0 To 255) As Byte
End Type
```

Listing 114: API-Funktion zum Abfangen von Tastenklicks

```
Declare Function GetKeyboardState Lib "User32.dll" _
 (kbArray As KeyboardBytes) As Long

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 die Prozedur `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 diese Prozedur wieder zu beenden und die Überwachung der Taste Esc abzuschließen.

100 Dateien suchen

Mithilfe der API-Funktion `SearchTreeForFile` aus *Listing 115* können Sie auch Dateien suchen und deren Speicherpfad ermitteln.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 str As String
```

Listing 115: Datei suchen über eine API-Funktion

```
  Dim lng As Long

  str = String(Verzeichnisse, 0)
  lng = SearchTreeForFile("C:\Eigene Dateien", "Artikel.xls", str)
  If lng <> 0 Then
    MsgBox "Datei wurde gefunden: " & vbLf & _
    str, vbInformation
  Else
    MsgBox "Datei konnte nicht gefunden werden!", vbCritical
  End If
End Sub
```

Listing 115: Datei suchen über eine API-Funktion (Forts.)

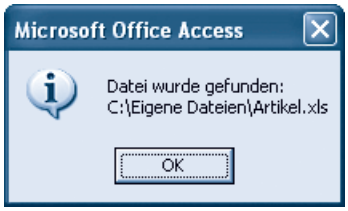


Abbildung 61: Dateien suchen

101 Dateiinformationen auslesen

Mithilfe von API-Funktionen lassen sich Datumsinformationen zu Dateien ermitteln. So wird im Beispiel aus *Listing 116* das Erstellungsdatum, das letzte Zugriffsdatum sowie das letzte Speicherdatum einer Datenbank ermittelt.

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

Listing 116: Dateiinformationen auslesen

```
Public Type SYSTEMTIME
  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 lng As Long
  Dim lngDatum As Double
  Dim lngZeit As Double

  lng = FileTimeToLocalFileTime(FT, LT)
  lng = FileTimeToSystemTime(LT, ST)
  If lng Then
    lngDatum = DateSerial(ST.wYear, ST.wMonth, ST.wDay)
    lngZeit = TimeSerial(ST.wHour, ST.wMinute, ST.wSecond)
    lngDatum = lngDatum + lngZeit

    If lngDatum > 0 Then
      DatPrüf = Format$(lngDatum, "dd.mm.yy hh:mm:ss")
    Else
      DatPrüf = "kein gültiges Datum"
    End If
  End If
End Function

Sub DateiInfoDialog()
  Dim lngFile As Long
  Dim WFD As WIN32_FIND_DATA
  Dim strDatei As String
  Dim strErstellt As String
  Dim strGespeichert As String
```

Listing 116: Dateiinformationen auslesen (Forts.)

```
  Dim strZugriff As String

  strDatei = Application.CurrentDb.Name
  lngFile = FindFirstFile(strDatei, WFD)

  If lngFile > 0 Then
    strErstellt = DatPrüf(WFD.ftCreationTime)
    strGespeichert = DatPrüf(WFD.ftLastWriteTime)
    strZugriff = DatPrüf(WFD.ftLastAccessTime)
    MsgBox "Datei angelegt: " & strErstellt & vbLf _
      & "Letzte Änderung: " & strGespeichert & vbLf _
      & "Letzter strZugriff: " & strZugriff, vbInformation, strDatei
  Else
    MsgBox "Datei wurde nicht gefunden!", vbCritical, strDatei
  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 die aufrufende Prozedur zurückgegeben. Dort werden alle Informationen zusammengefasst und in einer Bildschirmmeldung ausgegeben.

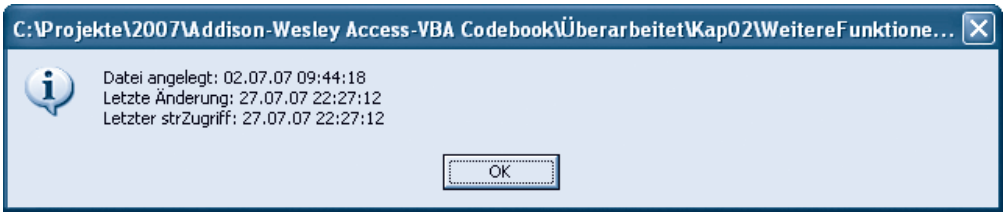


Abbildung 62: Auf Dateieigenschaften zugreifen

102 Internetverbindung aktiv?

Wenn Sie mit einer Anwendung 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlInternet
'=====================================================
```

Listing 117: Internetverbindung überprüfen

```
Public Declare Function InternetGetConnectedState Lib "wininet.dll" _
  (ByRef lpdwFlags As Long, ByVal dwReserved As Long) As Long

Sub InternetAktiv()
  Dim bool As Boolean

  bool = InternetGetConnectedState(0&, 0&)
  If bool = 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.

103 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 gerade 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
' 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 pnt As POINTAPI
  Dim i As Integer
```

Listing 118: Punktgenaue Koordinaten ermitteln

```
  i = GetCursorPos(pnt)
  If i <> 0 Then
    MsgBox "X-Position: " & pnt.x & vbLf & _
      "Y-Position: " & pnt.y, vbInformation
  Else
    MsgBox "Es konnte keine Position ermittelt werden"
  End If
End Sub
```

Listing 118: Punktgenaue Koordinaten ermitteln (Forts.)

Definieren Sie zuerst die Struktur `POINTAPI`, in der Sie die Position des Mauszeigers in Form der Pixel-Koordinaten speichern. Wenden Sie im Anschluss daran die API-Funktion `GetCursorPos` an, die die aktuelle Position des Mauszeigers wiedergibt. Geben Sie die Koordinaten danach in einer Meldung auf dem Bildschirm aus.

Abbildung 63: Die genaue Cursorposition ermitteln

Eigene Funktionen schreiben

Alles, was VBA und Access noch nicht zur Verfügung steht, können Sie selbst programmieren. Auf den nächsten Seiten finden Sie einige Funktionen, die Access noch erweitern.

104 Stückzahl pro Stunde errechnen

Stellen Sie sich vor, Sie haben eine Gesamtzeit, in der produziert wurde 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 Stück/Stunde.

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

Listing 119: Stückzahl pro Stunde errechnen

```
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 (Forts.)

Abbildung 64: Die einzelnen Stundenleistungen

105 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
' 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 1.440 multiplizieren.

106 Minuten in Stunden umrechnen

Auch der umgekehrte Fall 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Sub Bearbeitungszeit()
  MsgBox ZeitInMinuten("1:25")
End Sub

Function MinutenInStd(intMin As Integer) As Date
  MinutenInStd = intMin / 24 / 60
End Function
```

Listing 121: Minuten in Stunden umrechnen

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 1.440.

107 Normalzeit in Industriezeit umrechnen

Bei einer Industrieminute wird beispielsweise die Uhrzeit 3:30 als 3,5 dargestellt, also auf 100er-Basis umgerechnet.

Dazu kann eine Funktion erstellt werden, die Sie in *Listing 122* sehen.

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

Function IndZeit(Zeit As Date) As Single
  IndZeit = Zeit * 24
End Function

Sub NormalZeitInIndustriezeitWandeln()
  MsgBox IndZeit("3:30")
End Sub
```

Listing 122: Normalzeit in Industriezeit umrechnen

108 Industriezeit in Normalzeit umwandeln

Der umgekehrte Vorgang ist ebenso von Interesse und wird über die Funktion aus *Listing 123* realisiert.

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

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

109 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
' 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(2008) 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.

110 Stundengeschwindigkeit errechnen

Im folgenden Beispiel wird ein Marathon von einem Läufer in 2 Stunden und 13 Minuten zurückgelegt. Um jetzt eine Geschwindigkeitsangabe in km/Stunde zu errechnen, setzen Sie die Funktion aus *Listing 125* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 dtmZeitGesamt As Date

  sngGesamtKm = 42
  dtmZeitGesamt = "2:13"

  MsgBox "Die Geschwindigkeitsangabe km/h beträgt: " & _
  Round(StdKm(sngGesamtKm, dtmZeitGesamt), 2)
End Sub
```

Listing 125: Durchschnittsgeschwindigkeit km/h errechnen

Abbildung 65: Eine relativ flotte Geschwindigkeit für einen Marathon

111 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================
```

Listing 126: Den Benzinverbrauch errechnen

```
Function Benzin(ltr As Single, km As Single) As Single
  Benzin = Round((ltr / km) * 100, 2)
End Function

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

112 Das Idealgewicht bestimmen

Wenn Sie wissen möchten, wie hoch Ihr Ideal- bzw. Normalgewicht ist, dann können Sie beide Informationen in einer Funktion errechnen.

Das Idealgewicht errechnet sich, indem Sie von Ihrer Körpergröße in cm den Wert 100 subtrahieren und das Ergebnis daraus mit dem Multiplikator 0,9 multiplizieren.

Das Normalgewicht entspricht der Körpergröße in cm, von der 100 Zentimeter subtrahiert werden.

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

Listing 127: Normal- und Idealgewicht errechnen

```
  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 (Forts.)

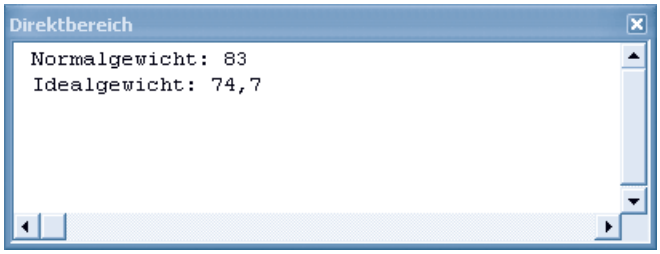


Abbildung 67: Normal- und Idealgewicht bei einer Größe von 1,83

113 Fahrleistung pro Tag errechnen

Möchten Sie Ihre durchschnittlich gefahrenen Kilometer/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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function KmProTag(sngKMGesamt As Single, dtmAkt As Date, dtmKauf As Date) _
  As Single
  KmProTag = Round(sngKMGesamt / (dtmAkt - dtmKauf), 2)
End Function

Sub FahrleistungProTag()
  MsgBox "Durchschnittlich wurden " & _
    KmProTag(34567, "20.07.2008", "25.10.2004") & " km pro Tag gefahren!"
End Sub
```

Listing 128: Die durchschnittliche Kilometerleistung pro Tag

Abbildung 68: Durchschnittsleistung/Tag

114 Berechnung des Bremswegs

Die Berechnung des Bremswegs bei einem Auto berechnet sich aus drei Einzelformeln:

- Reaktionsweg = (Geschwindigkeit (in km/h) / 10) *3
- Bremsweg = Geschwindigkeit (in km/h) / 10) ^2
- Anhalteweg = Reaktionsweg + Bremsweg

Beim Reaktionsweg wird die 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 vom 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function AnhalteWeg(intGeschw As Integer) As Single
  AnhalteWeg = (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 " & AnhalteWeg(intkm) & " Metern"
  Next intkm
End Sub
```

Listing 129: Anhalteweg errechnen

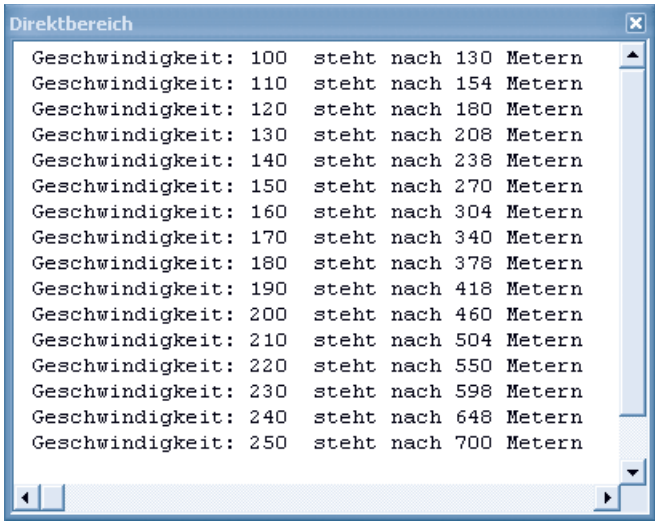


Abbildung 69: Die Dokumentation des Anhaltewegs

115 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 Prozeduren aufgerufen werden, und als Rückgabewert den Wert `True`, sofern ein Objekt verfügbar ist, bzw. den Wert `False`, wenn ein Objekt nicht zur Verfügung steht, zurückgeben. Im zweiten Fall können Sie dann entscheiden, wie Sie im Einzelnen weiter vorgehen möchten.

Im Beispiel aus *Listing 130* möchten Sie in ein bestimmtes Verzeichnis wechseln. Zu diesem Zweck müssen Sie sicherstellen, dass das Zielverzeichnis auch existiert. Wenn nicht, dann soll die Prozedur das nicht vorhandene Verzeichnis selbstständig anlegen. Danach soll der Verzeichniswechsel wie gewohnt stattfinden.

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

Function VerzeichnisExistiert(strName As String) As Boolean
  On Error GoTo Fehler
  ChDir (strName)
  VerzeichnisExistiert = True
  Exit Function
```

Listing 130: Prüfung, ob ein Verzeichnis existiert

```
Fehler:
  VerzeichnisExistiert = False
End Function


Sub VerzeichnisWechsel()
  Dim bool As Boolean
  Const Ordner = "C:\Eigene Dateien\"

  bool = VerzeichnisExistiert(Ordner)
  If bool = False Then
    'Neuanlage des Verzeichnisses
    MkDir Ordner
  End If
    ChDir Ordner
End Sub
```

Listing 130: Prüfung, ob ein Verzeichnis existiert (Forts.)

Die Funktion `VerzeichnisExistiert` 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 in dieses Verzeichnis zu wechseln. Schlägt dieser Vorgang fehl, dann wird ein Fehler erzeugt, den Sie über die `On Error`-Klausel abfangen. 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 bei der aufrufenden Prozedur im ersten Schritt eine Variable vom Typ `Boolean`. Diese Variable meldet Ihnen das Ergebnis der Funktion `VerzeichnisExistiert` in Form eines Wahrheitswertes 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 der Prozedur. Rufen Sie nun die Funktion `VerzeichnisExistiert` auf und warten auf den Rückgabewert der Funktion, den Sie über eine `If`-Abfrage auswerten. In dem 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.

116 Datenbank überprüfen

Wenn Sie versuchen eine Datenbank zu öffnen bzw. eine Datenbank zu löschen, sollte deren Existenz auf jeden Fall gewährleistet sein. 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 löschen, wird der Laufzeit-

fehler 53 (Datei nicht gefunden) angezeigt. Beide Fehlermeldungen können Sie unterdrücken, indem Sie die Lösung aus *Listing 131* anwenden:

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

Function DBExistiert(strName As String) As Boolean
  DBExistiert = False
  If Len(strName) > 0 Then DBExistiert = (Dir(strName) <> "")
End Function

Sub DBLöschen()
  Dim bool As Boolean
  Const Datei = "c:\Eigene Dateien\DB1.mdb"

  bool = DBExistiert(Datei)
  If bool = 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

Die Funktion DBExistiert 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 String an die Funktion übergeben wurde. Wenn ja, dann liefert diese Funktion einen Wert größer als null. In diesem Fall wenden Sie die Funktion Dir an, um zu testen, ob die gewünschte Datei vorhanden ist. Wird die spezifizierte Datei gefunden, dann meldet die Funktion Dir einen Wert ungleich leer zurück. In diesem Fall gibt die Funktion DBExistiert 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.

117 Datenbankstatus überprüfen

Für manche Aktionen ist es nötig, eine Datenbank im Exklusivzugriff zu öffnen. Dies können Sie in einem Netzwerk nur, wenn die Datei nicht bereits von einem anderen Anwender geöffnet wurde. Sie können feststellen, ob eine Datenbank bereits von einem Anwender geöffnet ist. Die Funktion für diesen Zweck lautet:

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

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

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 ist die Datenbank momentan von mindestens einem Anwender geöffnet. Ein Öffnen der Datenbank im Exklusivzugriff wird dann nicht möglich sein.

118 Objektstatus erkennen

Sehr nützlich ist auch die Funktion aus *Listing 133*. Diese Funktion prüft den Status eines Objektes. Dabei können Sie bei dieser Lösung erkennen, ob eine Tabelle, eine Abfrage, ein Formular, ein Bereicht, ein Makro oder ein Modul geöffnet ist oder nicht.

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

Listing 133: Prüfung, ob ein Objekt geöffnet ist

```
Function ObjektStatus(ObjTyp As Integer, strName As String) As Boolean
  ObjektStatus = SysCmd(acSysCmdGetObjectState, ObjTyp, strName)
End Function


Sub ObjektPrüfen()
  Dim bool As Boolean

  bool = ObjektStatus(0, "Kategorien")
  If bool = True Then
    MsgBox "Objekt geöffnet!"
  Else
    MsgBox "Objekt geschlossen!"
  End If
End Sub
```

Listing 133: Prüfung, ob ein Objekt geöffnet ist (Forts.)

Übergeben Sie der Methode über einen numerischen Wert, der 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 Objektes. In Abhängigkeit vom Status wird dementsprechend eine Meldung am Bildschirm angezeigt.

119 Dateien zählen

Stellen Sie sich vor, Sie müssten in einer Prozedur feststellen, wie viele Dateien sich in einem Verzeichnis befinden. Dazu erfassen Sie zunächst folgende Funktion aus *Listing 134*:

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

Listing 134: Dateien aus einem Verzeichnis zählen

```
Function DZ(strText As String) As Long
  Dim strDatei As String
  Dim lng As Integer

  strDatei = Dir$(strText & "\*.*")
  Do While Len(strDatei) > 0
    lng = lng + 1
    strDatei = Dir$()
  Loop
  DZ = lng
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 (Forts.)

Die Funktion `DZ` erwartet als Eingabe den Namen des Verzeichnisses, auf das Sie zugreifen möchten. Als Ergebnis liefert Ihnen die Funktion 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$(strText & "\*.*")` beispielsweise in `StrDatei = Dir$(strText & "\*.mdb")` ändern. Diese kleine Änderung bewirkt, dass nur Access-Datenbanken, und zwar im »alten« Format, gezählt werden.

In der aufrufenden Prozedur legen Sie gleich zu Beginn fest, welches Verzeichnis Sie durchsuchen möchten. Übergeben Sie anschließend der Funktion `DZ` genau dieses Verzeichnis.

Abbildung 70: Dateien zählen

120 Initialen erstellen

Sollen aus Vor- und Nachnamen Initialen gebildet werden, sodass aus dem Namen Bernd Michael Held die Initialen BMH werden, dann können Sie die Funktion aus *Listing 135* einsetzen.

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

Function Initit(str As String) As String
  Dim intz As Integer

  str = " " & RTrim(str)

  For intz = 2 To Len(str)
    If Mid(str, intz - 1, 1) = " " Then _
      Initit = Initit & Mid(str, intz, 1)
  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` eventuell vorkommende nachgestellte Leerzeichen. Danach beginnen Sie die Verarbeitung beim zweiten Zeichen des ersten Namens. Dabei setzen Sie eine Schleife auf, die Buchstabe für Buchstabe abarbeitet. Innerhalb der Schleife prüfen Sie, wann der jeweilige Name zu Ende ist, da Sie von jedem Einzelnamen nur das erste Zeichen übertragen müssen. Diese jeweils ersten Zeichen speichern Sie und geben sie am Ende der Funktion an die aufrufende Prozedur zurück.

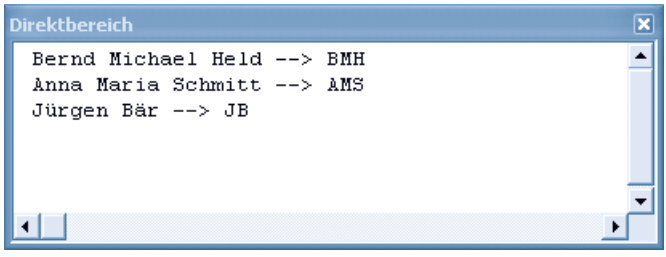


Abbildung 71: Auf schnelle Art und Weise Initialen bilden

121 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 sehen Sie in *Listing 136*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 "Jahr2008 --> " & PosErsteZahl("Jahr2008") & ". 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 und setzen 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 48–57 den Zahlen 0 bis 9. Diesen Wertbereich grenzen Sie innerhalb der `Select Case`-Anweisung aus. Wird das erste Zeichen im Text gefunden, das numerisch ist, dann 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.

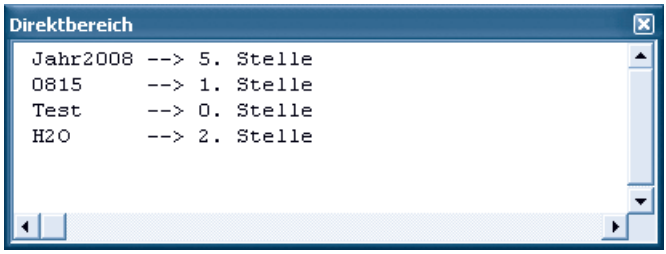


Abbildung 72: Das jeweils erste numerische Zeichen wurde gemeldet.

122 Position des ersten Buchstabens erkennen

Analog dazu können Sie die Position des ersten Buchstabens in den Bereichen A–Z und a–z ermitteln, indem Sie die Funktion aus *Listing 137* einsetzen.

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


Sub WoStecktDerBuchstabe()
 MsgBox "Der erste Buchstabe im Text 00045T steckt an Position " & _
 PosErsterBuch("00045T")
End Sub
```

Listing 137: Position erster Buchstabe ermitteln

Ermitteln Sie im ersten Schritt die Länge des Textes und setzen 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 65–90 Kleinbuchstaben und die Werte 97–122 den Großbuchstaben. Diese Wertbereiche grenzen Sie innerhalb der `Select Case`-Anweisung aus. Wird der erste Buchstabe in der Zelle gefunden, dann springen Sie mit der Anweisung `Exit Function` aus der Funktion. In der Variablen steht dann automatisch die richtige Position des Zeichens. Wurde kein Buchstabe gefunden, dann meldet die Funktion den Wert `0` zurück.

Abbildung 73: Das große O sieht der 0 ziemlich ähnlich.

123 Sonderzeichen entfernen

Müssen Sie Daten weiterverarbeiten, in denen Sonderzeichen wie Punkte und Kommas vorkommen, die Sie nicht weiter verarbeiten 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=====================================================

Function SonderzRaus(strText As String) As Double
  Dim strPuffer As String
  Dim intz As Integer

  strPuffer = ""
  intz = 1
  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
```

Listing 138: Punkte und Kommas aus Zeichenfolge entfernen

```
  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 mit übertragen, da Sie es jeweils wieder über die Variable `intZ` subtrahieren. Ermitteln Sie danach die Position des nächsten Kommas über die Funktion `InStr`.

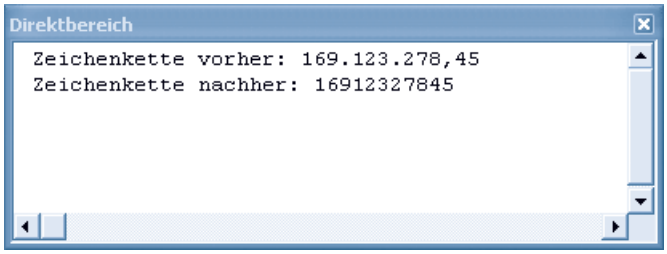


Abbildung 74: Bereinigung von Zeichenketten

124 Numerische Zeichen in Zeichenfolge zählen

Im nächsten Beispiel aus *Listing 139* wird die Anzahl der Zahlen, die in einem Text enthalten sind, gezählt.

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

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 "WG0897    --> " & AnzahlZahlen("WG0897")
```

Listing 139: Die Anzahl der Zahlen zählen

```
  Debug.Print "XY O90   --> " & AnzahlZahlen("XY O90")
  Debug.Print "101010TZ --> " & AnzahlZahlen("101010TZ")
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` im Zähler in Ihrer Funktion.

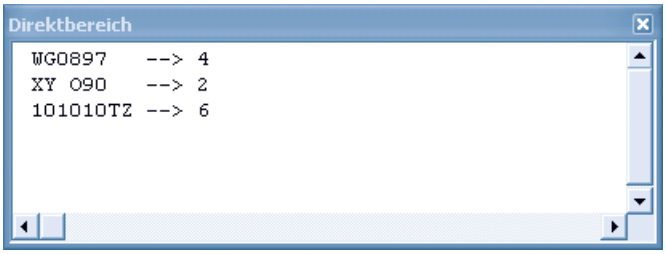


Abbildung 75: Auch hier ist wieder auf den Unterschied zwischen 0 und O zu achten!

125 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=======================================================

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
```

Listing 140: Alle Buchstaben in einem Text zählen

```
Sub BuchstabenZählen()
  Debug.Print "WG0897   --> " & AnzahlBuch("WG0897")
  Debug.Print "XY 090   --> " & AnzahlBuch("XY 090")
  Debug.Print "101010TZ --> " & AnzahlBuch("101010TZ")
End Sub
```

Listing 140: Alle Buchstaben in einem Text zählen (Forts.)

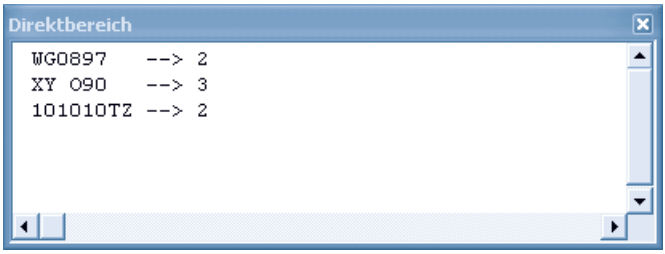


Abbildung 76: Die Anzahl der Buchstaben wurde ermittelt.

126 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 verwendet Buchstaben als Zahlen. So ist der Buchstabe M gleichbedeutend mit dem Wert 1.000. Von daher müssen Sie Zeichen für Zeichen im Text abarbeiten und die entsprechenden Werte sammeln und am Ende als Rückgabewert liefern.

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

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)
```

Listing 141: Umwandeln von römischen Zahlen

```
      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

127 Arabische Zahlen umwandeln

Beim gerade umgekehrten Vorgang werden Sie an dieser Stelle eine etwas andere Vorgehensweise kennenlernen. 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 den Quellcode aus *Listing 142*:

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

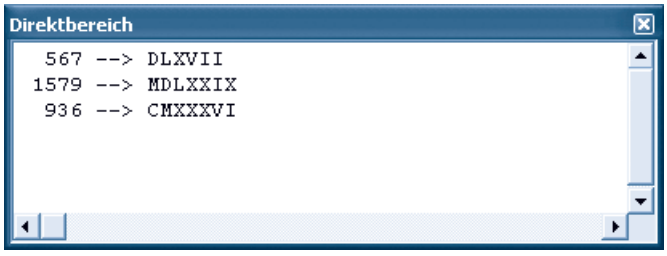


Abbildung 78: Die Zahlen wurden in die römische Syntax umgesetzt.

128 Auf Dokumenteigenschaften zugreifen

Im folgenden Beispiel aus *Listing 143* wird auf die Dokumenteigenschaften einer Datenbank zugegriffen. Dabei orientiert sich die Funktion an *Tabelle 56*.

Eigenschafts-Nummer	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 56: Die verfügbaren Dokumenteigenschaften

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

Function ZeigeDateiEigenschaften(Dateiname, EigenschaftsNr As Byte)
  Dim fso As Object
  Dim str As String

  On Error Resume Next
  Set fso = CreateObject("Scripting.FileSystemObject")
  With fso.GetFile(Dateiname)
    Select Case EigenschaftsNr
      Case Is = 0: str = .path
      Case Is = 1: str = Mid(.path, 1, Len(.path) - Len(.Name))
      Case Is = 2: str = .Name
      Case Is = 3: str = .Type
      Case Is = 4: str = .Size
      Case Is = 5: str = CDate(.DateCreated)
      Case Is = 6: str = CDate(.DateLastModified)
      Case Is = 7: str = CDate(.DateLastAccessed)
      Case Else
        str = "Ungültige EigenschaftsNr!"
    End Select
  End With
  ZeigeDateiEigenschaften = str
End Function
```

Listing 143: Dokumenteigenschaften über eine Funktion abfragen

```
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.

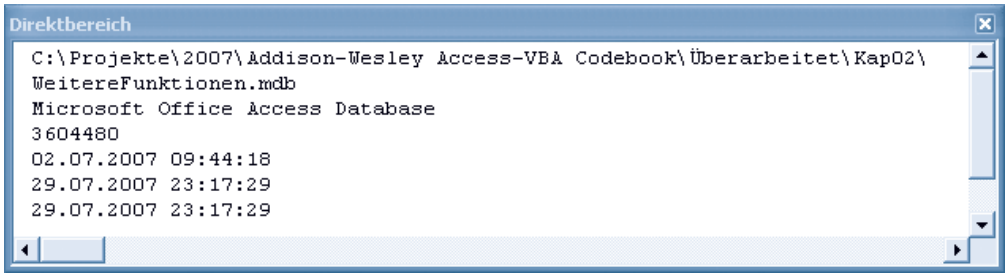


Abbildung 79: Die Dokumenteigenschaften werden über eine Nummer ermittelt.

129 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 gibt es doch eine 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
' Dateiname  WeitereFunktionen.mdb
' Modul      MdlWeitereFunktionen
'=======================================================
```

Listing 144: Den letzten Tag im Monat ermitteln

```
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 im Monat ermitteln (Forts.)

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.

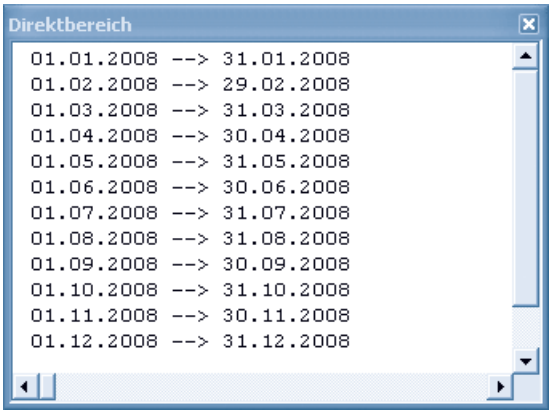


Abbildung 80: Eine komplette Liste mit den »letzten Tagen« liegt vor.

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.

Das AccessObject-Objekt

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 Datenzugriffsseiten (Bem.: ab Access 2007 nicht mehr unterstützt)
Datenbankdiagramm	`AllDatabaseDiagrams`	gespeicherte Datenbankdiagramme
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 57: 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
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'======================================================

Sub ModuleAuflisten()
  Dim obj As AccessObject
  Dim dbs As CurrentProject
```

Listing 145: Alle Module werden im Direktfenster dokumentiert.

```
  Set dbs = Application.CurrentProject
  For Each obj In dbs.AllModules
    Debug.Print "Name        : " & obj.Name
    Debug.Print "Erstellt am : " & obj.DateCreated & vbLf
  Next obj
End Sub
```

Listing 145: Alle Module werden im Direktfenster dokumentiert. (Forts.)

Über die Eigenschaft `CurrentProject` haben Sie Zugriff auf die aktuelle Datenbank. Über die Eigenschaft `Name` können Sie 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.

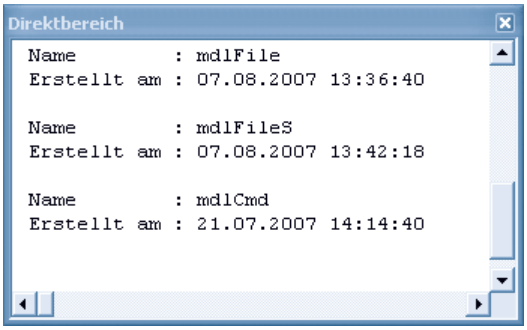


Abbildung 81: Die verfügbaren Module der Datenbank

Hinweis

Im *Kapitel VBE & Security* 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. In der Prozedur 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
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'=====================================================
```

Listing 146: Tabellennamen dokumentieren

```
Sub TabellenAuflisten()
  Dim obj As AccessObject
  Dim dat As CurrentData

  Set dat = Application.CurrentData
  For Each obj In dat.AllTables
    Debug.Print "Tabellenname: " & obj.Name
    Debug.Print "Geöffnet    : " & obj.IsLoaded & vbLf
  Next obj
End Sub
```

Listing 146: Tabellennamen dokumentieren (Forts.)

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` können Sie den Namen der jeweiligen Tabelle ermitteln. Die Funktion `IsLoaded` gibt Auskunft darüber, ob eine Tabelle momentan geöffnet ist. Wenn ja, dann meldet diese Funktion den Rückgabewert `True`.

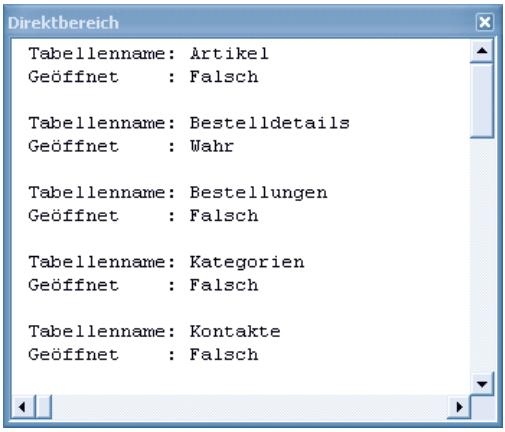


Abbildung 82: Tabellen auflisten und Status prüfen

132 Alle Abfragen auflisten

Über den Einsatz des `AccessObject` können Sie alle Abfragen der Datenbank anzeigen lassen. In der Prozedur aus *Listing 147* werden die Namen aller in der Datenbank befindlichen Abfragen dokumentiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlAccObj
'=====================================================
```

Listing 147: Alle Abfragen dokumentieren

```
Sub AbfragenAuflisten()
  Dim obj As AccessObject
  Dim dat As CurrentData

  Set dat = Application.CurrentData

  For Each obj In dat.AllQueries
    Debug.Print obj.Name
  Next obj
End Sub
```

Listing 147: Alle Abfragen dokumentieren (Forts.)

In einer `For Each Next`-Schleife durchlaufen Sie Abfragen und ermitteln die Namen der Abfragen über die Eigenschaft `Name`.

Die Application-Objekte

Das `Application`-Objekt steht an oberster Stelle. Es bezieht sich auf die aktive Microsoft Access-Anwendung und beinhaltet alle darunter liegenden Objekte wie Formulare, Reports, Drucker und Bildschirm.

133 Name der aktuellen Datenbank abfragen

Soll der Name der aktuell geöffneten Datenbank abgefragt werden, dann können Sie diese Aufgabe über die Prozedur aus *Listing 148* durchführen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DatenbankName()
  MsgBox "Die aktuelle Datenbank heißt: " & _
    Application.CurrentProject.Name
End Sub
```

Listing 148: Dateiname der Datenbank ermitteln – Variante 1

Über die Eigenschaft `Name`, die 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 in der Prozedur aus *Listing 149*. Gegenüber der vorigen Variante, die Ihnen lediglich den Dateinamen anzeigt, zeigt Ihnen diese Variante den kompletten Dateipfad der Datenbank an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DatenbankName2()
  MsgBox "Die aktuelle Datenbank heißt: " & _
    Application.CurrentDb.Name
End Sub
```

Listing 149: Pfad der Datenbank ermitteln – Variante 2

Über die Eigenschaft `Name`, die Sie auf das Objekt `CurrentDb` anwenden, können Sie den Namen der aktuellen Datenbank ermitteln.

134 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
' 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
' 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

135 Pfad der Anwendung abfragen

Über den Einsatz der Methode `SysCmd` können Sie den Pfadnamen der Anwendung ermitteln. Im Beispiel aus *Listing 152* wird der Installationspfad von Microsoft Access ermittelt.

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

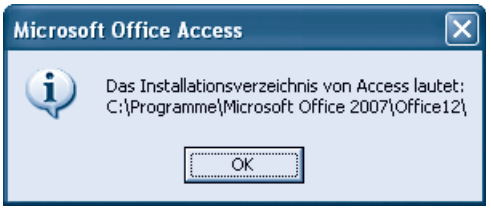


Abbildung 84: Den Installationspfad von Access ausgeben

136 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
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=======================================================

Sub AccessVersionErmitteln()
  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.

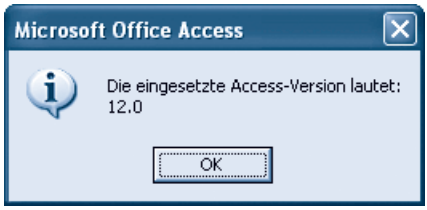


Abbildung 85: Die Versionsabfrage wurde durchgeführt.

Hinweis

Die Versionsnummer 12.0 steht für Microsoft Access 2007.

Es gibt eine zweite Möglichkeit, wie Sie die Version von Access ermitteln können. Erledigen können Sie dies über die Eigenschaft `Version` des Objekts `Application`:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=======================================================

Sub AccessVersionErmitteln2()
  MsgBox "Sie arbeiten mit der Access-Version: " _
    & Application.Version
End Sub
```

Listing 154: Die eingesetzte Access-Version ermitteln

137 Aktuellen Anwendernamen abfragen

Mithilfe der Methode `CurrentUser` können Sie den Namen des aktuellen Benutzers der Datenbank zurückgeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub AnwenderNameAusgeben()
  MsgBox "Der aktuelle Benutzer ist: " & _
    Application.CurrentUser, vbInformation
End Sub
```

Listing 155: Den aktuellen Anwendernamen abfragen

138 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 Automatiken geöffnet haben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DatenbankBeenden()
  Application.CloseCurrentDatabase
End Sub
```

Listing 156: Aktuelle Datenbank schließen

139 Applikation beenden

Möchten Sie Microsoft Access beenden, dann setzen Sie die Methode `Quit` ein, die Sie auf das Objekt `Application` anwenden. In der Prozedur aus *Listing 157* wird die Anwendung beendet und alle Änderungen an der aktuellen Datenbank vorher gesichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub ApplikationBeenden()
  Application.Quit acQuitSaveAll
End Sub
```

Listing 157: 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.

140 Drucker auflisten

Um zu ermitteln, welche Drucker Sie im Einsatz haben und an welchem Anschluss diese hängen, können Sie das Auflistungsobjekt `Printers` nutzen, das Sie im Zusammenspiel mit dem Objekt `Application` einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlApp
'=====================================================

Sub DruckerListen()
  Dim prt As Printer

  For Each prt In Application.Printers
    With prt
      Debug.Print "Druckername: " & .DeviceName & vbCr _
        & "Anschluss: " & .Port
    End With
  Next prt
End Sub
```

Listing 158: Drucker und Ports listen

Die Eigenschaft DeviceName zeigt den Druckernamen an. Mithilfe der Eigenschaft Port können Sie den Anschluss anzeigen, dem der Drucker zugeordnet ist.

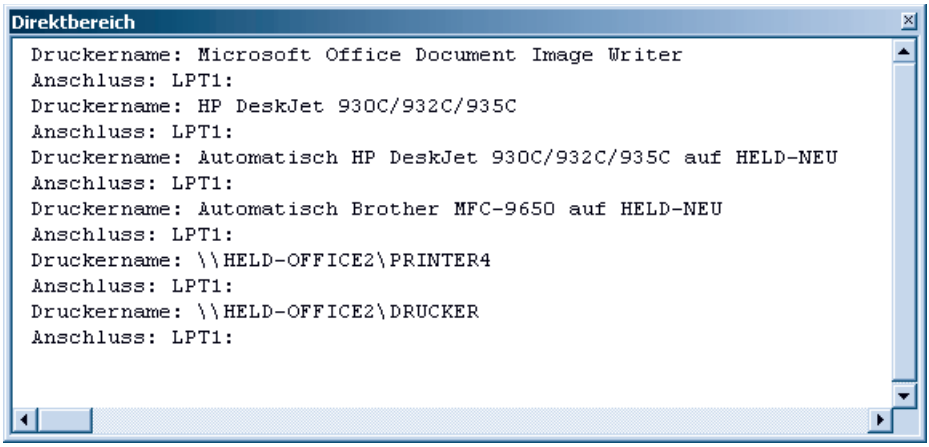


Abbildung 86: Die verfügbaren Drucker mit Anschlussbelegung

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.

Bei dem folgenden Beispiel aus *Listing 159* werden alle Textfelder des Formulars KUNDEN im Direktfenster ausgegeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlCtl
'=======================================================

Sub SteuerelementeDokumentieren()
  Dim ctl As Control
  Dim frm As Form

  Set frm = Form_Kunden

  For Each ctl In frm.Controls
    If TypeName(ctl) = "Textbox" Then
    Debug.Print ctl.Name
  End If
  Next ctl
End Sub
```

Listing 159: Alle Textfelder in einem Formular auslesen

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 Steuerelementes über die Eigenschaft `Name` aus.

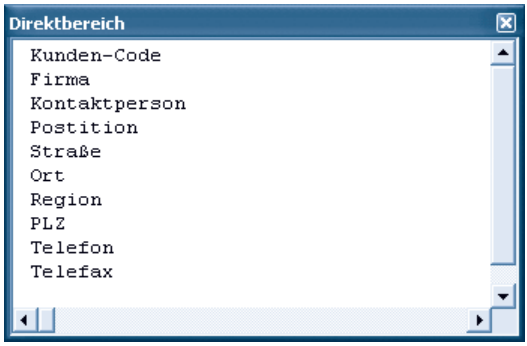


Abbildung 87: Die Textfelder eines Formulars wurden dokumentiert.

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 nun eine Auswahl von typischen Aufgaben, die in Verbindung mit diesem Objekt durchgeführt werden können.

141 Filter in Tabellen setzen

Mithilfe der Methode `ApplyFilter` können Sie einen Filter hinzufügen. Diesen Filter können Sie gewöhnlich in Tabellen oder Formularen einsetzen. 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 58: 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 als 10 aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 unter 10 Stück werden angezeigt.

Sollen beispielsweise gleich mehrere Artikel ausgefiltert werden, dann können Sie diese Aufgabe über die Prozedur aus *Listing 161* lösen. Dabei werden alle Artikel mit dem Namen TOFU oder KONBU in der Tabelle ARTIKEL herausgefiltert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 herausgefiltert.

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.

142 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 59: Die Argumente der Methode Rename

Im folgenden Beispiel aus *Listing 162* wird eine Tabelle umbenannt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=======================================================

Sub TabelleUmbenennen()
  On Error GoTo Fehler
  DoCmd.Rename "ArtikelNeu", acTable, "ArtikelAktuell"
  Exit Sub
```

Listing 162: Eine Tabelle umbenennen

```
Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 162: Eine Tabelle umbenennen (Forts.)

Sollte die Umbenennung nicht erfolgreich sein, dann können Sie statt eines Prozedurabsturzes 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.

143 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 60: Die Argumente der Methode CopyObject

Beim folgenden Beispiel aus *Listing 163* wird die Tabelle ARTIKEL kopiert und der Name ARTIKELAKTUELL vergeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 die Prozedur aus *Listing 164* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleKopierenInExterneDB()
  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 statt eines Prozedurabsturzes 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 über das Objekt Err abfangen

144 Objekte löschen

Über die Methode `DeleteObject` können Sie ein Objekt wie beispielsweise eine Tabelle oder ein Formular aus einer Datenbank löschen. 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 61: Die Argumente der Methode DeleteObject

Im folgenden Beispiel aus *Listing 165* wird die Tabelle ARTIKELAKTUELL aus der aktuellen Datenbank gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleLöschen()
  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 löschen

145 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 inkl. aller Funktionen und Prozeduren ▶ `acOutputQuery`: Export der Ergebnisse eine 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.

Tabelle 62: Die Argumente der Methode OutputTo

Argument	Beschreibung
OutputFormat	Optional. 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.
OutputFile	Optional. Der Pfad sowie der Dateiname der Datei, in die 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 62: Die Argumente der Methode OutputTo (Forts.)

Im folgenden Beispiel aus *Listing 166* wird die Tabelle ARTIKEL in eine Textdatei geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

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

Geben Sie bei der Methode OutputTo den Objekttyp, den Namen des Objektes, 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.

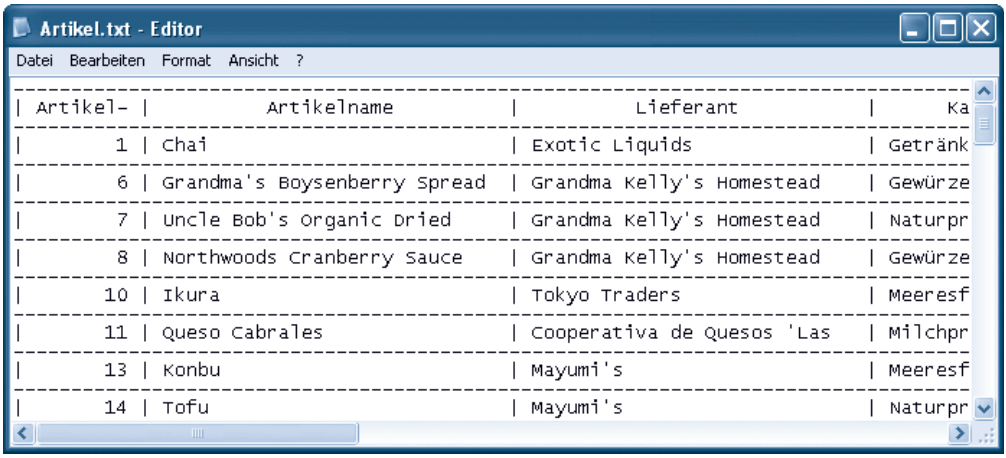


Abbildung 91: Die Tabelle liegt nun als Textdatei vor.

146 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`.
`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 während des Druckvorgangs nicht zu sortieren.

Tabelle 63: Die Argumente der Methode PrintOut

Im Beispiel aus *Listing 167* wird ein Bericht in der aktuell geöffneten Datenbank geöffnet und einige Seiten davon ausgedruckt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 Prozedurablaufs eingeschaltet bzw. ausgeschaltet wird. Wenn Sie diese Methode mit dem Argument `False` aufrufen, dann erfolgt der Druckvorgang völlig im Hintergrund. Nach dem Drucken der Seiten wenden Sie die Methode `Close` an, um den Bericht wieder zu schließen, und schalten die Bildschirmaktualisierung wieder ein.

147 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-Datenzugriffseite wird einem E-Mail-Empfänger zugestellt. ▶ `acSendForm`: Ein Formular soll über eine E-Mail versendet werden. ▶ `acSendModule`: Ein Modul wird per E-Mail versendet.

Tabelle 64: Die Argumente der Methode SendObject

Argument	Beschreibung
Objekttyp	▶ 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-Objektes 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 Text-Editor, beispielsweise im Zubehör-Programm von Windows Notepad, zu lesen. ▶ acFormatXLS: Dabei handelt es sich um das Excel-Tabellenformat. ▶ AcFormatDAP: Bei dieser Konstante handelt es sich um Datenzugriffsseiten.
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.
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-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.
Betreff	Optional. Repräsentiert die BETREFF-Zeile der E-Mail-Nachricht. Geben Sie dort einen Betreff in doppelten Anführungszeichen 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-Nachricht aufgenommen.

Tabelle 64: Die Argumente der Methode SendObject (Forts.)

Argument	Beschreibung
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. Es handelt 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 64: Die Argumente der Methode SendObject (Forts.)

Im folgenden Beispiel aus *Listing 168* wird eine E-Mail ohne Dateianhang versendet.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=======================================================

Sub TextMailVersenden()
  On Error GoTo Fehler
  DoCmd.SendObject , , , _
    "Fritz.Mueller@online.de", _
    "Machero@aol.com", , "Feedback erwünscht", _
    "Sehr geehrter Herr XY," & vbCrLf & vbCrLf & _
    "bitte senden Sie mir ein Feedback zu meiner Anfrage vom 20.02.2008" & _
    vbCrLf & vbCrLf & "Mit freundlichem Gruß," & vbCrLf & _
    "Bernd Held", True
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 168: Eine einfache Text-E-Mail versenden

Da bei dieser Art von E-Mail kein Dateianhang entsteht, lassen Sie die ersten drei Argumente der Methode SendObject einfach leer. Danach geben Sie den Adressaten sowie den Kopieempfänger und den Titel sowie den eigentlichen Text der E-Mail an. Über den Einsatz 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 vor dem Versand noch einmal zur Kontrolle angezeigt werden soll.

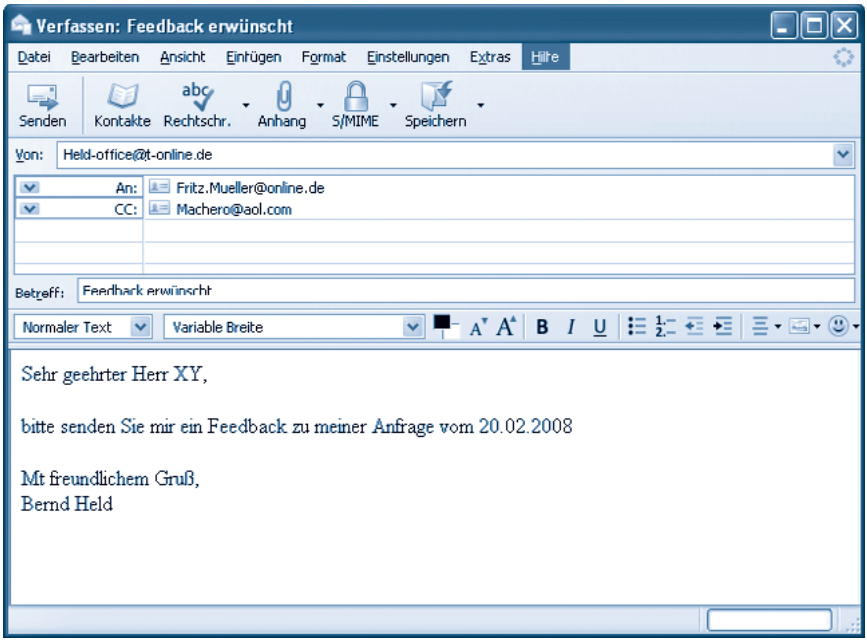


Abbildung 92: Eine Textmail ohne Anhang erstellen

Im nächsten Beispiel aus *Listing 169* wird die Tabelle ARTIKEL gleich an mehrere Anwender per E-Mail im Excel-Format verschickt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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", _
    "Hallo Kollegen, " & vbCrLf & vbCrLf & _
    "anbei die aktuell gut laufenden Artikel!" & _
    vbCrLf & vbCrLf & "Beste Grüße," & vbCrLf & _
    "Fritz Meier", True
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 169: E-Mail mit Dateianhang versenden

Legen Sie in den ersten drei Argumenten der Methode `SendObject` den Objekttyp, den Objektnamen und das gewünschte Ausgabeformat fest. Danach geben Sie den Adressaten sowie den Kopieempfänger und den Titel sowie den eigentlichen Text der E-Mail an.

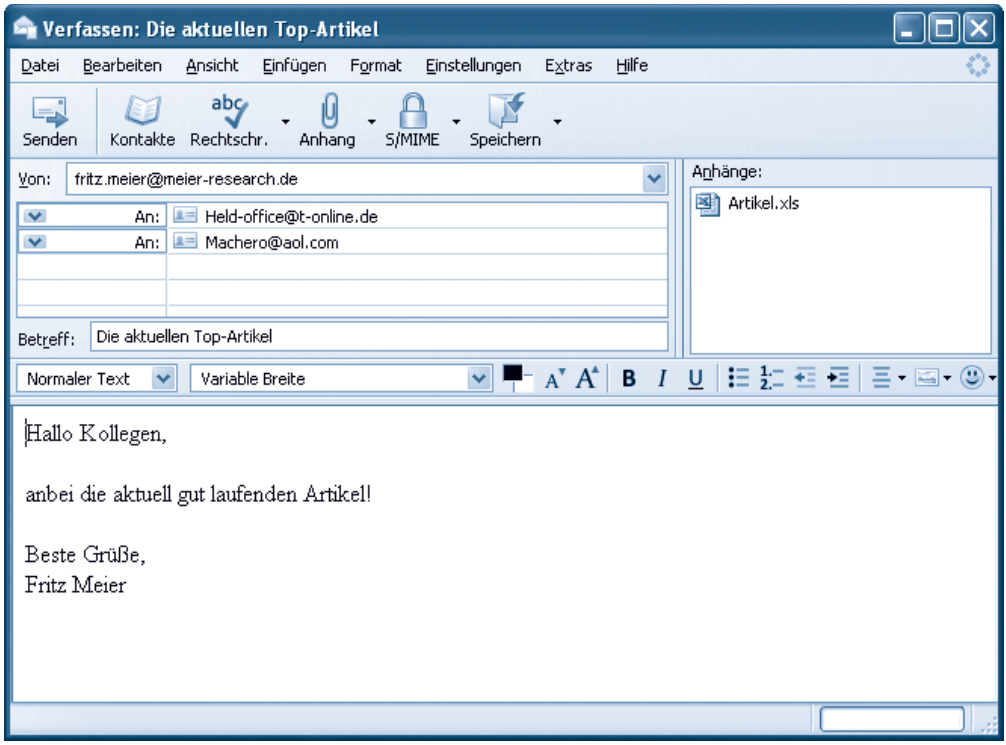


Abbildung 93: Die Tabelle Artikel wird im Excel-Format verschickt.

148 Feld im Formular aktivieren

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 65: Die Argumente der Methode GoToControl

Im Beispiel aus *Listing 170* wird das Formular KUNDEN geöffnet und der Fokus auf das Feld FIRMA gesetzt.

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

149 Datensatz in Tabelle aktivieren

Über die Methode `GoToRecord` können Sie einen bestimmten Satz in einem Formular oder einer Tabelle voreinstellen. 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 rückwärtsgeblättert werden soll, wenn Sie für das Argument Datensatz 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 66: Die Argumente der Methode GoToRecord

Im Beispiel aus *Listing 171* wird die Tabelle ARTIKEL geöffnet und der letzte Datensatz ausgewählt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=======================================================

Sub LetztenDatensatzAuswählen()
  DoCmd.OpenTable "Artikel"
  DoCmd.GoToRecord acDataTable, "Artikel", acLast
End Sub
```

Listing 171: Den letzten Datensatz einer Tabelle auswählen

Abbildung 95: Der letzte Datensatz der Tabelle wurde ausgewählt.

150 Menübefehle ausführen

Mithilfe der Methode `RunCommand` können Sie jeden möglichen Befehl in Access, den Sie über die Menüs ansteuern können, auch per Prozedur 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 68* können Sie den gewünschten Befehl ausführen.

Tabelle 67: 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 Suchen-Fenster 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, das die Beziehungen der Tabellen anzeigt
`acCmdSend`	Öffnet den E-Mail-Dialog

Tabelle 68: Die wichtigsten Command-Konstanten der Methode RunCommand

Im folgenden Beispiel aus *Listing 172* wird der Dialog DOKUMENTEIGENSCHAFTEN aufgerufen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=======================================================

Sub DokumenteigenschaftenAnzeigen()
  DoCmd.RunCommand acCmdDatabaseProperties
End Sub
```

Listing 172: Dokumenteigenschaften aufrufen

151 Symbolleisten ein- und ausblenden

Über die Methode `ShowToolbar` können Sie eine Symbolleiste ein- bzw. ausblenden. 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 69: Die Argumente der Methode ShowToolbar

Im folgenden Beispiel aus *Listing 173* wird die Symbolleiste WEB ein- und wieder ausgeblendet.

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

Ab Access-Version 2007 kann dieses Beispiel nicht mehr eingesetzt werden, da die bisherigen Symbolleisten durch die sog. Multifunktionsleiste ersetzt wurde.

152 SQL-Abfragen ausführen

Über die Methode RunSQL können Sie eine SQL-Abfrage ausführen. 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 70: 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 Texte von INHABER in CHEF umzuändern.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub SQLAbfrageAusführen()
  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-Abfrage ausführen

153 Fenstergröße festlegen

Mithilfe der Methode `MoveSize` können Sie Position und Größe des aktiven 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 71: Die Argumente der Methode MoveSize

Hinweis

Die Maßeinheit für die Argumente lautet Twips. (Twip: Eine Maßeinheit, die 1/20 Punkt und ca. 1/1440 Zoll entspricht. 567 Twips sind ca. ein Zentimeter.)

Im folgenden Beispiel aus *Listing 175* wird das Formular KUNDEN beim Aufruf in Größe und Position angepasst. Dazu stellen Sie das Ereignis `Form_Open` ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 soweit vergrößern, dass es den kompletten Dokumentenbereich des Access-Hauptfensters ausfüllt. Diese Methode hat keine weiteren Argumente.
- Über die Methode `Minimize` können Sie das aktive Fenster auf eine kleine Titelleiste unten im Microsoft Access-Hauptfenster reduzieren. Diese Methode hat keine weiteren Argumente.

Im Beispiel aus *Listing 176* wird ein Bericht aufgerufen und das Fenster maximiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub FensterMaximieren()
  On Error GoTo Fehler
  DoCmd.OpenReport "Rechnung", 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.

Bei dem 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
' Dateiname  Objekte.mdb
' Formular   Form_Kunden
'=====================================================

Private Sub Form_Close()
  DoCmd.Restore
End Sub
```

Listing 177: Formulargröße vor dem Schließen wieder zurücksetzen

154 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 des Datenbanktyps an, der zum Importieren, Exportieren oder Verknüpfen von Daten verwendet werden kann. 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 72: Die Argumente der Methode TransferDatabase

Im folgenden Beispiel aus *Listing 178* wird der Bericht RECHNUNG aus der externen Datenbank NORDWIND.MDB in die aktuell geöffnete Datenbank importiert und unter dem Namen RECHNUNGNEU gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub BerichtImportieren()
  On Error GoTo Fehler
  DoCmd.TransferDatabase acImport, _
    "Microsoft Access", _
    "C:\Eigene Dateien\Nordwind.mdb", acReport, _
```

Listing 178: Einen Bericht aus einer externen Datenbank importieren

```
    "Rechnung", "RechnungNeu"
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 178: Einen Bericht aus einer externen Datenbank importieren (Forts.)

Im nächsten Beispiel aus *Listing 179* wird die Tabelle PERSONAL in der externen Datenbank NORDWIND.MDB als gelinkte Tabelle importiert und in der aktuellen Datenbank unter dem Namen PERSONALGELINKT angezeigt.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleLinken()
  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 verlinken

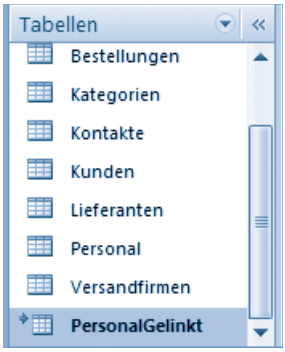


Abbildung 96: Die verlinkte Tabelle erkennen Sie am Pfeilsymbol.

155 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)
```

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.
`Tabellenname`	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 für Sie neu 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`	Dieses 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 73: Die Argumente der Methode TransferSpreadsheet

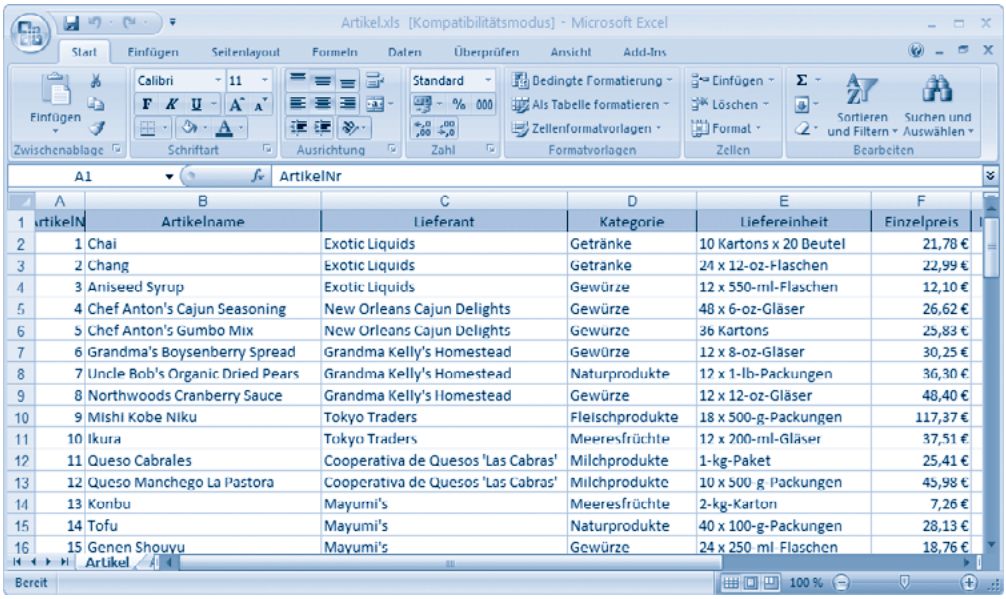
Im folgenden Beispiel aus *Listing 180* wird die Access-Tabelle ARTIKEL in eine Excel 2000-Tabelle überführt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleExportierenExcel()
  On Error GoTo Fehler
  DoCmd.TransferSpreadsheet acExport, _
    acSpreadsheetTypeExcel9, "Artikel", _
```

Listing 180: Eine Datentabelle als Excel-Tabelle exportieren

```
    "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 (Forts.)



	A	B	C	D	E	F
1	ArtikelN	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis
2	1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €
3	2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	22,99 €
4	3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	12,10 €
5	4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	26,62 €
6	5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €
7	6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	30,25 €
8	7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	36,30 €
9	8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	48,40 €
10	9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	18 x 500-g-Packungen	117,37 €
11	10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200-ml-Gläser	37,51 €
12	11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	1-kg-Paket	25,41 €
13	12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	10 x 500 g Packungen	45,98 €
14	13	Konbu	Mayumi's	Meeresfrüchte	2-kg-Karton	7,26 €
15	14	Tofu	Mayumi's	Naturprodukte	40 x 100-g-Packungen	28,13 €
16	15	Genen Shouyu	Mayumi's	Gewürze	24 x 250-ml-Flaschen	18,76 €

Abbildung 97: Das Ergebnis liegt in einer Excel-Tabelle vor.

156 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`.

Tabelle 74: Die Argumente der Methode TransferText

Argument	Beschreibung
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. Bei Textdateien mit Trennzeichen und Microsoft Word-Seriendruck-Datendateien können Sie dieses Argument auslassen, um so die Standardimport- und -exportspezifikation auszuwählen.
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 74: Die Argumente der Methode TransferText (Forts.)

Im folgenden Beispiel aus *Listing 181* wird eine Tabelle als Textdatei exportiert. Da der Parameter Spezifikationsname nicht spezifiziert wird, wird die Standardexportspezifikation angewendet, die in der Grundeinstellung das Semikolon als Trennzeichen verwendet. Geben Sie der Textdatei die Endung .CSV. CSV steht für COMMA SEPARATED VALUES und ist eine typische Dateinamen-Erweiterung für Textdateien, die durch Sonderzeichen getrennte Werte enthalten. Sie könnten der Datei jedoch genauso gut die Endung .TXT geben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TabelleExportierenText()
  On Error GoTo Fehler
  DoCmd.TransferText acExportDelim, , _
```

Listing 181: Tabelle in Textdatei exportieren

```
    "Artikel", "C:\Eigene Dateien\Artikel.csv"
  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 notwendig:

1. Wählen Sie bei geöffneter Datenbank auf der Registerkarte EXTERNE DATEN aus der Gruppe IMPORTIEREN das Element TEXTDATEI IMPORTIEREN, sofern Sie mit Access 2007 arbeiten. Arbeiten Sie mit früheren Versionen von Access, wählen Sie aus dem Menü DATEI den Befehl EXTERNE DATEN/IMPORTIEREN. Die folgenden Schritte beschreiben die Vorgehensweise bei Access 2007. Bei früheren Versionen von Access ist die Vorgehensweise jedoch recht ähnlich.

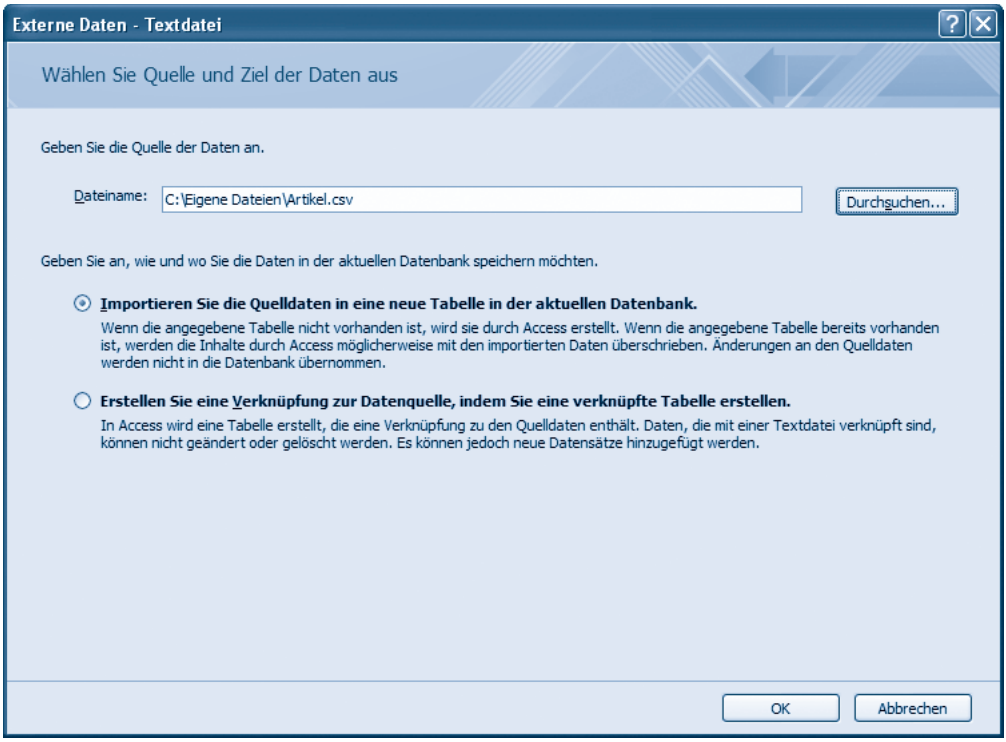


Abbildung 98: Textdatei auswählen

2. Im Dialogfeld EXTERNE DATEN – TEXTDATEI wählen Sie die Textdatei aus, die Sie in eine Access-Tabelle importieren möchten, und legen fest, ob Sie die ausgewählte Datei importieren oder eine Verknüpfung zu dieser erstellen möchten. Wählen Sie die erste Option, um die Daten zu importieren.
3. Klicken Sie auf die Schaltfläche OK.

Abbildung 99: Der Textimport-Assistent meldet sich.

4. Der TEXTIMPORT-ASSISTENT wird gestartet. Im angezeigten Dialog können Sie auswählen, ob die Daten der zu importierenden Datei durch Trennzeichen oder in Spalten mit fester Breite, also nur durch Leerzeichen, getrennt sind. Da bei der Textdatei ARTIKEL.CSV 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!).

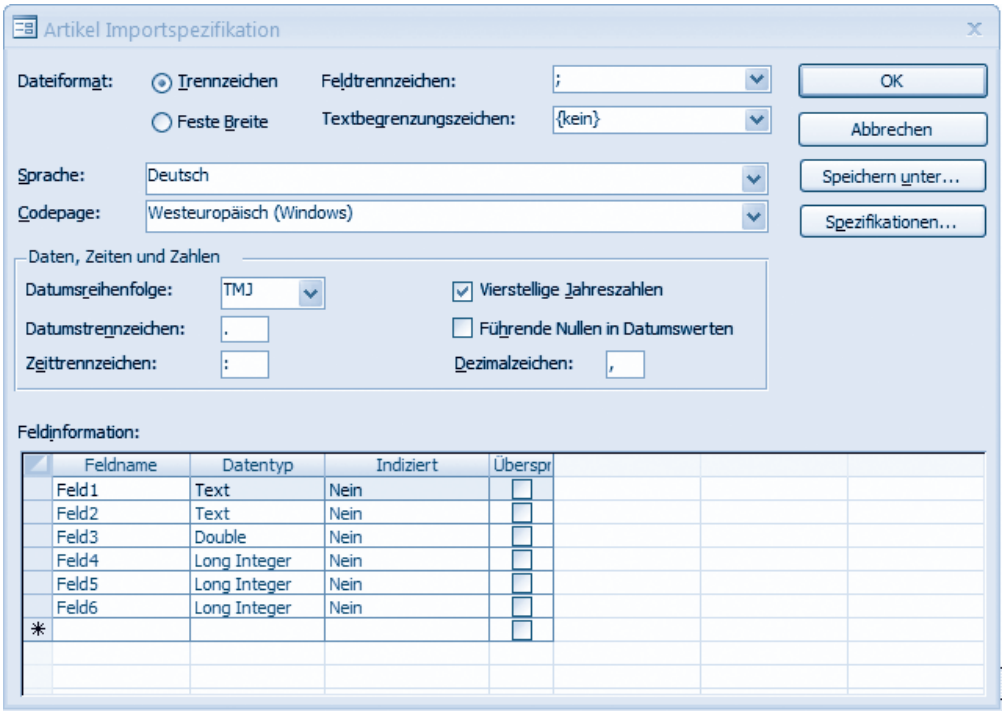


Abbildung 100: Die Importspezifikation definieren

6. Der Dialog ARTIKEL IMPORTSPEZIFIKATION wird angezeigt. Im Gruppenfeld FELDINFORMATION hat Access anhand der Textdatei ARTIKEL.CSV die Felder vom Datentyp her gesehen bereits automatisch für Sie 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 UNTER, um die soeben definierte Importspezifikation zu sichern.

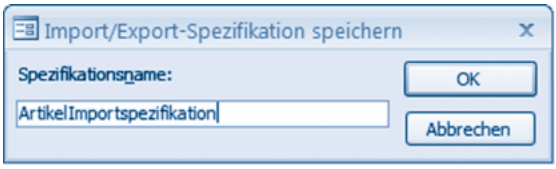


Abbildung 101: Die Importspezifikation speichern

8. Geben Sie der Importspezifikation einen Namen und bestätigen Sie mit OK. Merken Sie sich diesen Namen, den Sie später für Ihre Prozedur benötigen.
9. Klicken Sie danach auf die Schaltfläche OK, um das Dialogfeld ARTIKEL IMPORT-SPEZIFIKATION zu schließen.
10. Wieder im TEXTIMPORT-ASSISTENT angekommen, klicken Sie auf WEITER, um zum nächsten Importschritt zu gelangen.
11. Übergehen Sie auch den nächsten Schritt *(Abbildung 102)* mit einem Klick auf die Schaltfläche WEITER.

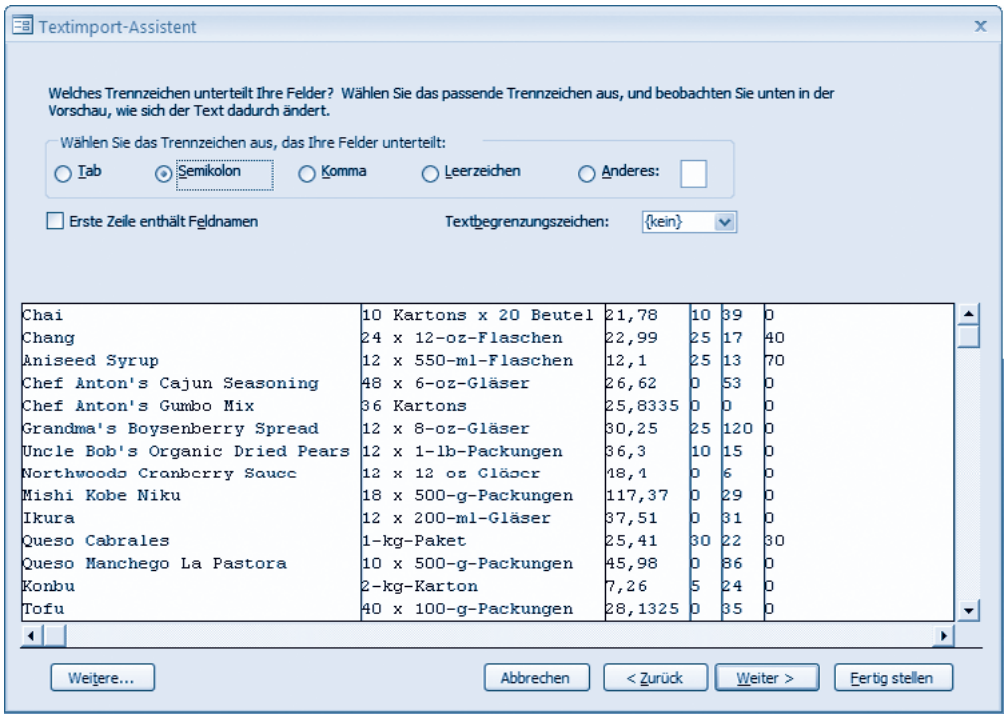


Abbildung 102: Trennzeichen festlegen

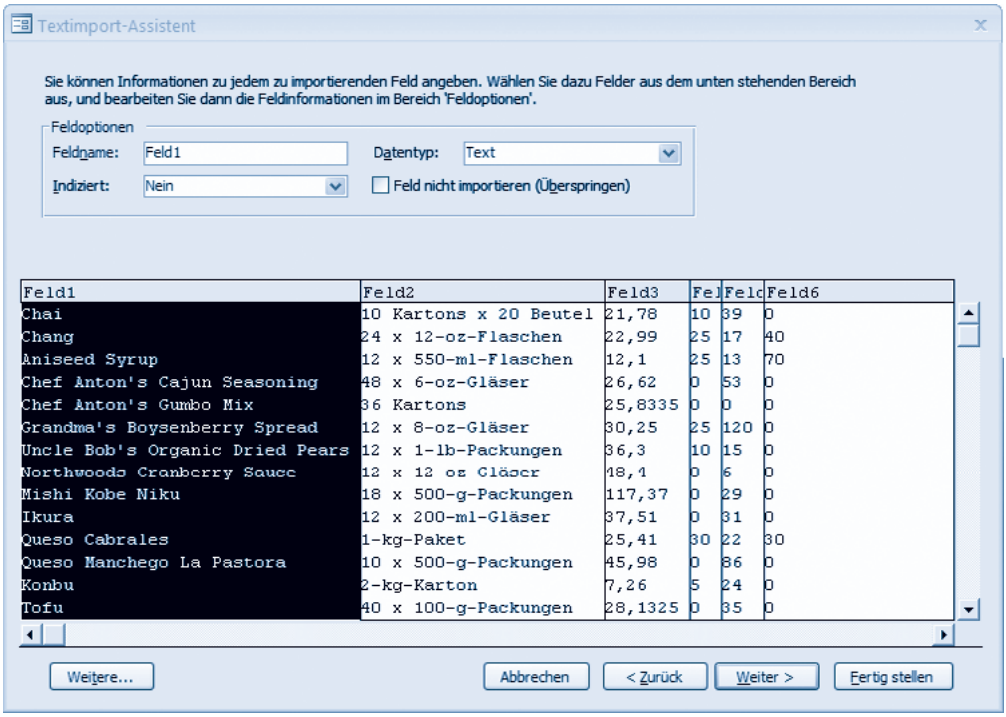


Abbildung 103: Felder festlegen

12. In diesem Schritt *(Abbildung 103)* können Sie noch einmal festlegen, wie die einzelnen Felder definiert werden sollen. Da Sie diese Aufgabe aber bereits vorher über die Importspezifikation durchgeführt haben, klicken Sie auf die Schaltfläche WEITER.

13. 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.

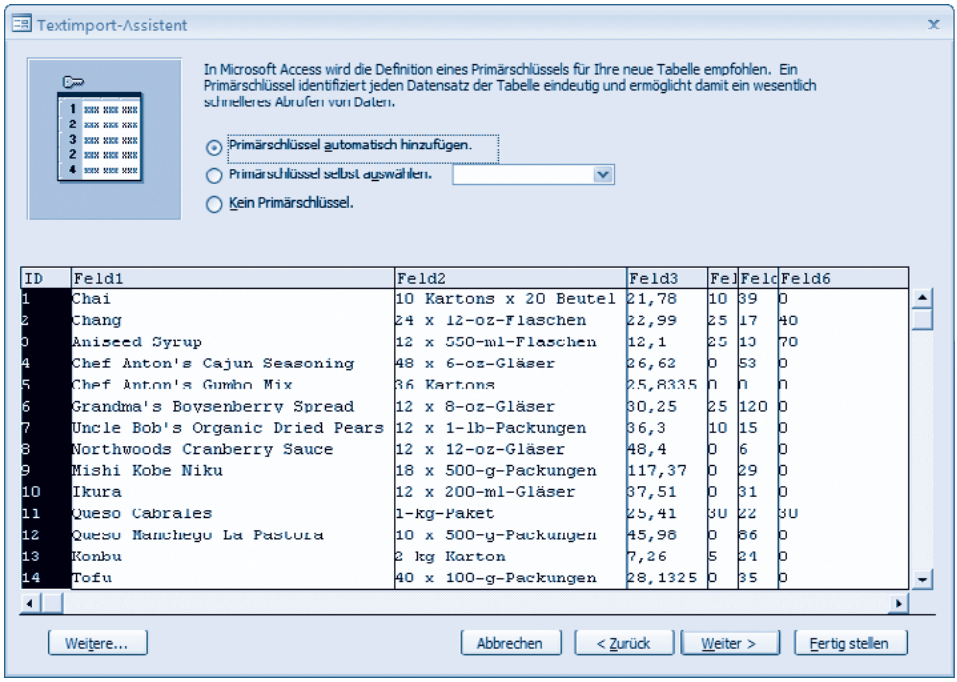


Abbildung 104: Ein Primärschlüssel kann angelegt werden.

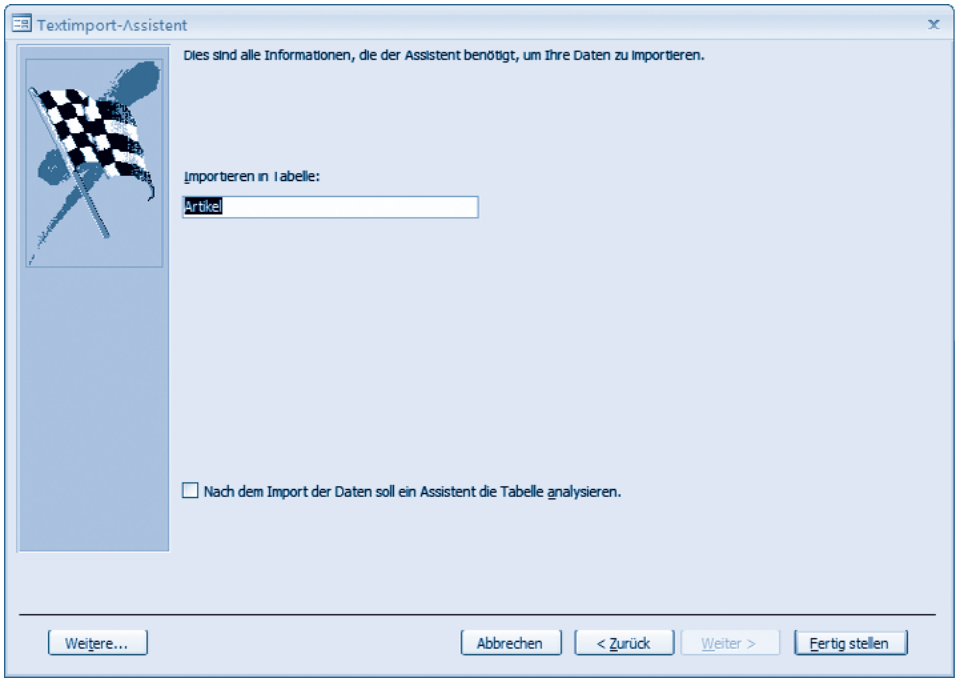


Abbildung 105: Zieltabelle angeben

14. Geben Sie an, in welche Tabelle Sie die Textdatei einfügen möchten. Sollte diese Tabelle noch nicht existieren, wird sie von Access automatisch angelegt.
15. Klicken Sie abschließend auf die Schaltfläche FERTIG STELLEN.

Jetzt können Sie die Prozedur für den Import einer Textdatei schreiben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlDoCmd
'=====================================================

Sub TextImportieren()
  On Error GoTo Fehler
  DoCmd.TransferText acImportDelim, _
    "ArtikelImportspezifikation", _
    "ArtikelX", _
    "C:\Eigene Dateien\Artikel.csv", False
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 182: Textdatei importieren

Das Reference-Objekt

Das Objekt `Reference` stellt einen Verweis auf die Typbibliothek einer anderen Anwendung oder eines anderen Projekts dar.

157 Bibliotheken auslesen

Über die `References`-Auflistung können Sie alle eingebundenen Bibliotheken in der Entwicklungsumgebung abarbeiten. Die `References`-Auflistung entspricht der Liste der Referenzen aus dem Dialogfeld VERWEISE.

Im Beispiel aus *Listing 183* werden alle eingebundenen Bibliotheken der Entwicklungsumgebung im Direktfenster aufgelistet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Objekte.mdb
' Modul      MdlRef
'=====================================================
```

Listing 183: Eingebundene Bibliotheken auslesen

```
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 (Forts.)

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.

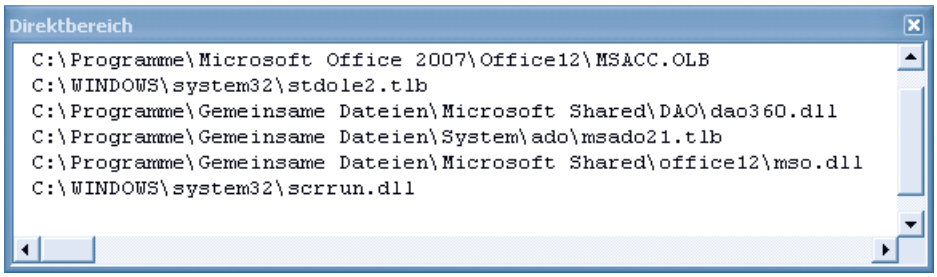


Abbildung 106: Alle geladenen Bibliotheken anzeigen

Hinweis

Im *Kapitel VBE & Security* können Sie nachschlagen, wie Sie mithilfe der VBE-Programmierung noch weiter gehende Aufgaben schnell erledigen können.

Das Balloon-Objekt

Über das Objekt `Balloon` können Sie Franz Klammer und Kollegen aus der Online-Hilfe aufrufen und mit eigenen Texten ausstatten. Um mit diesem Objekt arbeiten zu können, muss die Office-Bibliothek in der Entwicklungsumgebung aktiviert werden.

Hinweis

Mit Office 2007 hat sich Microsoft von den Office-Assistenten verabschiedet. Die folgenden Beispiele, die die Office-Assistenten betreffen, können Sie demnach nur bis einschließlich Office 2003 einsetzen. Für Neuentwicklungen sollten Sie, aus Gründen der Aufwärtskompatibilität Ihrer Programme, auf die Programmierung von Office-Assistenten verzichten.

158 Office-Assistenten einsetzen

Der Office-Assistent wird in der Prozedur aus *Listing 184* dafür eingesetzt, den Ablauf einer VBA-Schulung darzustellen.

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

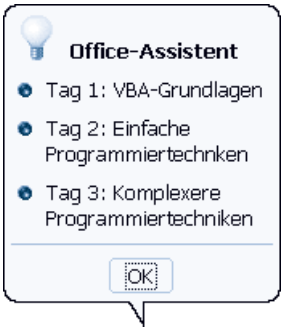


Abbildung 107: Der Office-Assistent wird aufgerufen.

Bei der Prozedur 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 75* zur Verfügung:

Konstante	Symbol
`msoIconAlert`	Gelbes Ausrufezeichen
`msoIconAlertCritical`	Rotes X-Symbol
`msoIconAlertInfo`	Info
`msoIconAlertQuery`	Fragezeichen
`msoIconAlertWarning`	Gelbes Ausrufezeichen
`msoIconNone`	Kein Symbol
`msoIconTip`	Glühbirne

Tabelle 75: Die Konstanten für das Symbol eines Balloons

Die Eigenschaft `Mode` gibt das modale Verhalten der Sprechblase des Office-Assistenten zurück oder legt es fest. Dabei stehen die folgenden Konstanten aus *Tabelle 76* 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 76: 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 77* einsetzen.

Konstante	Aufzählungszeichen
`msoBalloonTypeBullets`	Kleine Punkte
`msoBalloonTypeButtons`	Größere Punkte
`msoBalloonTypeNumbers`	Numerische

Tabelle 77: Die Konstanten der Eigenschaft BalloonType

Die Eigenschaft `Labels` legt die Beschriftungen für die Schaltflächen, Nummerierungen und Aufzählungen in einer Sprechblase des Office-Assistenten zurück.

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 ein 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

Tabelle 78: Die Konstanten für die Eigenschaft Animation

Konstante
msoAnimationThinking
msoAnimationWorkingAtSomething
msoAnimationWritingNotingSomething

Tabelle 78: 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 79* 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 79: Die Konstanten für die Schaltflächen eines Balloons

Die Methode `Show` zeigt das angegebene `Balloon`-Objekt an.

Die Frage, welche Schaltfläche im Office-Assistent angeklickt wurde, beantwortet die *Tabelle 80*.

Konstante	Schaltfläche
msoBalloonButtonAbort	Abbrechen
msoBalloonButtonBack	Zurück
msoBalloonButtonCancel	Abbrechen

Tabelle 80: Die Rückgabekonstanten der Methode Show

Konstante	Schaltfläche
msoBalloonButtonClose	Schließen
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 80: 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
' 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 108: Die Auswertung der Schaltfläche findet direkt nach dem Klick statt.

Das FileSystem-Objekt

Über das Objekt `FileSystem` können Sie Zugriff auf das Dateisystem Ihres Computers gewinnen. Unter anderem können Sie über dieses Objekt Dateien kopieren und umbenennen, Verzeichnisse erstellen und löschen und einige andere Dinge mehr.

159 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 81: Die Argumente der Methode CopyFile

Im folgenden Beispiel aus *Listing 186* wird die Excel-Arbeitsmappe KATALOG.XLS aus dem Verzeichnis C:\EIGENE DATEIEN in das Verzeichnis C:\TEMP kopiert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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:\Eigene Dateien\Katalog.xls", "C:\Temp\"
  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 die Prozedur aus *Listing 187*.

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

Sub DateienKopieren()
  Dim fs As Object

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")
  fs.CopyFile "C:\Eigene Dateien\*.xls", "C:\Temp\"
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 187: Mehrere Dateien kopieren

160 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 82: Die Argumente der Methode MoveFile

In der Prozedur aus *Listing 188* wird die Datei KATALOG.XLS von einem Verzeichnis in ein anderes verschoben.

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

Sub DateiVerschieben()
  Dim fs As Object

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")
  fs.MoveFile "C:\Eigene Dateien\Katalog.xls", "C:\Temp\"
  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 Sie in *Listing 189* sehen können.

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

Sub DateienVerschieben()
  Dim fs As Object

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")
```

Listing 189: Mehrere Dateien verschieben

```
  fs.MoveFile "C:\Eigene Dateien\*.xls", "C:\Temp\"
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 189: Mehrere Dateien verschieben (Forts.)

161 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. der `False` ist (Voreinstellung), wenn sie nicht gelöscht werden sollen.

Tabelle 83: Die Argumente der Methode DeleteFile

In der Prozedur aus *Listing 190* wird eine Datei ohne Rückfrage von der Festplatte gelöscht.

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

Sub DateiLöschen()
  Dim fs As Object

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")
  fs.DeleteFile "C:\Eigene Dateien\Katalog.xls"
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 190: Einzelne Datei löschen

Im Beispiel aus *Listing 191* werden alle Dateien aus dem Verzeichnis C:\TEMP gelöscht.

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

Sub DateienLöschen()
  Dim fs As Object

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")
  fs.DeleteFile "C:\Temp\*.*"
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 191: Alle Dateien in einem Verzeichnis löschen

162 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 vorhanden ist.

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

Sub DateiExistiert()
  Dim fs As Object

  Set fs = CreateObject("Scripting.FileSystemObject")
  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

Die Methode FileExists meldet den Wert True zurück, wenn die Datei im angegebenen Verzeichnis vorhanden ist.

163 Dateiinfo abfragen

Im folgenden Beispiel aus *Listing 193* werden Sie mithilfe des FileSystem-Objekts die Dateieigenschaften Erstellungsdatum, letzte Änderung und letzter Zugriff auf die Datei ermitteln. Dabei kommen die Eigenschaften DateCreated, DateLastModified und DateLastAccessed zum Einsatz.


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

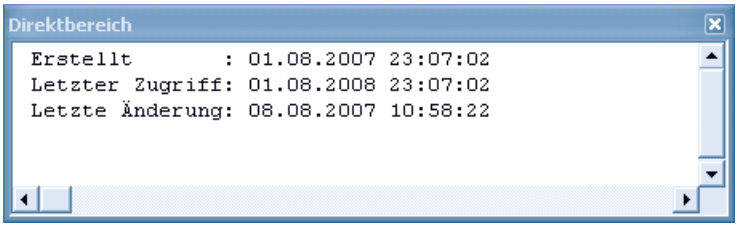


Abbildung 109: Die wichtigsten Informationen zur Datei

164 Verzeichnis anlegen

Über die Methode `CreateFolder` können Sie ein neues Verzeichnis anlegen. In der Prozedur aus *Listing 194* wird der neue Ordner KOPIE angelegt.

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

165 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. der `False` ist (Voreinstellung), wenn sie nicht gelöscht werden sollen.

Tabelle 84: Die Argumente der Methode DeleteFolder

Im folgenden Beispiel aus *Listing 195* wird der Ordner KOPIE ohne Rückfrage gelöscht.

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

Sub OrdnerLöschen()
  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

166 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 85: Die Argumente der Methode FolderExists

In der Prozedur aus *Listing 196* wird überprüft, ob der Ordner C:\EIGENE DATEIEN\KOPIE existiert.

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

Sub OrdnerExistiert()
  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

167 Ordnergröße ermitteln

Soll die Größe eines Ordners ermittelt werden, um beispielsweise festzustellen, ob der Inhalt des Ordners auf eine Diskette oder einen USB-Stick passt, dann setzen Sie die Eigenschaft `Size` ein.

Im Beispiel aus *Listing 197* wird die Größe des Ordners C:\EIGENE DATEIEN in MBytes ermittelt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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")
  MsgBox "Der Ordner hat eine Größe von " & _
    Round(fsOrdner.Size / 1024 / 1024, 2) & _
    " MByte", vbInformation
```

Listing 197: Die Ordnergröße ermitteln

```
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 197: Die Ordnergröße ermitteln (Forts.)

Die Größenangabe wird in Bytes ausgegeben. Um daraus Kilobytes zu machen, teilen Sie das Ergebnis durch 1.024. Sollen daraus MByte gemacht werden, dann teilen Sie wiederum durch 1.024. Runden Sie das Ergebnis auf zwei Stellen hinter dem Komma, indem Sie die Funktion `Round` einsetzen.

168 Ordner auflisten

Wenn Sie mit Verzeichnissen arbeiten, dann sollten Sie auch eine Prozedur-Lösung haben, die überprüft, ob in einem Verzeichnis auch 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
' Dateiname  Objekte.mdb
' Modul      MdlFile
'=====================================================

Function OrdnerDurchsuchen(strV As String) As Boolean
  Dim fs As Object
  Dim objOrdner As Object
  Dim objSub As Object
  Dim intZ As Integer

  intZ = 0
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set objOrdner = fs.GetFolder(strV)
  For Each objSub In objOrdner.subFolders
    Debug.Print "Ordner " & intZ & ": " & objSub.Name
    intZ = intZ + 1
  Next

  Set objOrdner = Nothing
  Set fs = Nothing
End Function
```

Listing 198: Unterordner aus einem Verzeichnis auflisten lassen

```
Sub VerzeichnisStrukturAuslesen()
  Dim bool As Boolean

  bool = OrdnerDurchsuchen("C:\Windows")
End Sub
```

Listing 198: Unterordner aus einem Verzeichnis auflisten lassen (Forts.)

Über die Prozedur `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.

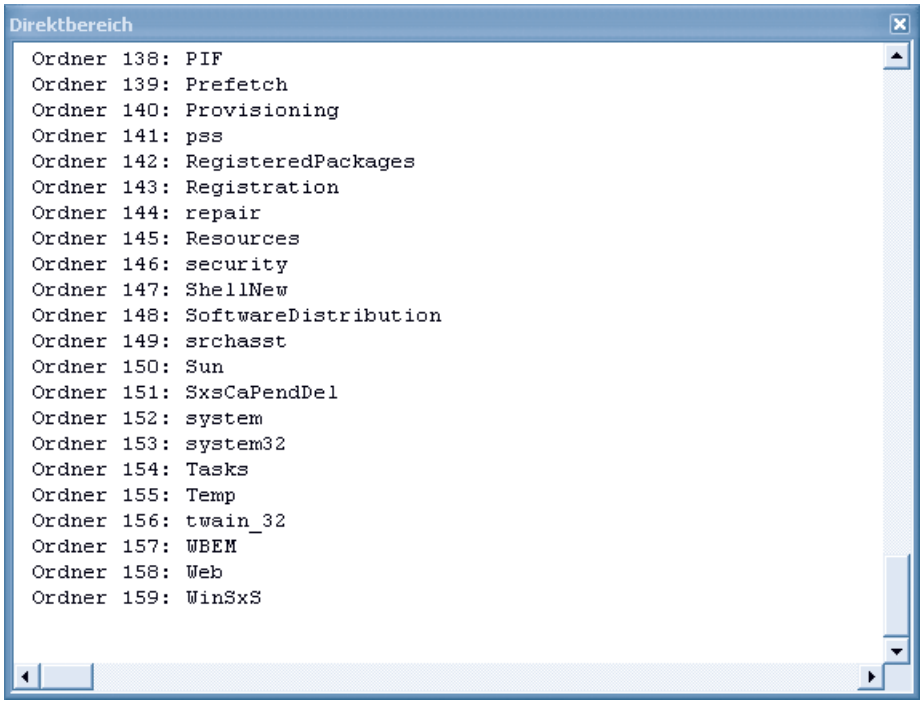


Abbildung 110: Alle Unterordner im Verzeichnis C:\Windows wurden aufgelistet.

169 Dateien auflisten

Möchten Sie alle Dateien eines Verzeichnisses ermitteln, dann erfassen Sie die Prozedur aus *Listing 199*.

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

Function OrdnerDurchsuchen2(strV As String) As Boolean
  Dim fs As Object
  Dim objOrdner As Object
  Dim objDatei As Object
  Dim intZ As Integer

  intZ = 1
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set objOrdner = fs.GetFolder(strV)
  For Each objDatei In objOrdner.Files
    Debug.Print "Datei " & intZ & ": " & objDatei.Name
    intZ = intZ + 1
  Next
  Set objOrdner = Nothing
  Set fs = Nothing
End Function

Sub DateienListen()
  Dim bool As Boolean

  bool = OrdnerDurchsuchen2("C:\Windows")
End Sub
```

Listing 199: Alle Dateien aus einem Ordner auflisten

Über die Prozedur `DateienListen` wird die Funktion `OrdnerDuchsuchen2` 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.

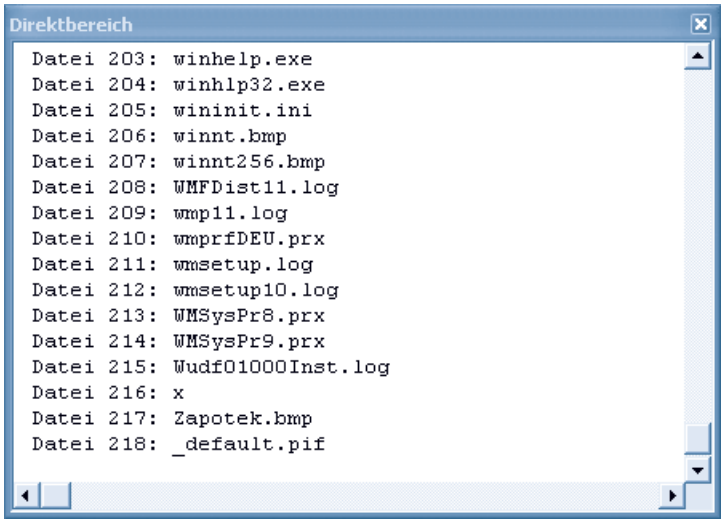


Abbildung 111: Alle Dateien wurden aufgelistet.

170 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.

In der Prozedur aus *Listing 200* werden alle Laufwerke Ihres Computers ausgelesen und deren Laufwerkstypen festgestellt.

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

Sub LaufwerkeAuslesen()
  Dim fs As Object
  Dim obj As Object
  Dim str As String

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")

  For Each obj In fs.Drives
    Select Case obj.DriveType
      Case 1
        str = "Disketten-Laufwerk/USB-Datenstick"
```

Listing 200: Laufwerkstypen ermitteln und ausgeben

```
      Case 2
        str = "Festplatte"
      Case 3
        str = "Netzlaufwerk"
      Case 4
        str = "CD-ROM-Laufwerk"
      Case 5
        str = "RAM-Disk"
      Case Else
        str = "nicht ermittelbar"
    End Select
    Debug.Print obj.DriveLetter & " ist ein(e) " & str
  Next obj
  Exit Sub

Fehler:
  MsgBox Err.Number & vbTab & Err.Description
End Sub
```

Listing 200: Laufwerkstypen ermitteln und ausgeben (Forts.)

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 ein anderes Wechsellaufwerk
2	Festplatte (fest eingebaut)
3	Netzlaufwerk
4	CD-ROM
5	RAM-Disk

Tabelle 86: Die DriveType-Eigenschaft 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.

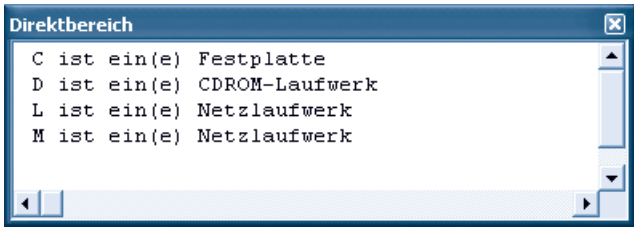


Abbildung 112: Die Laufwerkstypen wurden ermittelt.

171 Weitere Laufwerkseigenschaften

In der Prozedur aus *Listing 201* können Sie weitere Laufwerkseigenschaften ermitteln, wie beispielsweise den verfügbaren bzw. der Gesamtspeicherplatz von Ihrer Festplatte.


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

Sub LaufwerksEigenschaften()
  Dim fs As Object
  Dim obj As Object
  Dim str As String

  On Error GoTo Fehler
  Set fs = CreateObject("Scripting.FileSystemObject")

  For Each obj In fs.Drives
    With obj
      Select Case .DriveType
        Case 1
          str = "Disketten-Laufwerk/USB-Datenstick"
        Case 2
          str = "Festplatte"
        Case 3
          str = "Netzlaufwerk"
        Case 4
          str = "CD-ROM-Laufwerk"
        Case 5
          str = "RAM-Disk"
        Case Else
          str = "nicht ermittelbar"
      End Select
```

Listing 201: Weitere Laufwerkseigenschaften abfragen

```
      If .DriveType = 2 Then
        Debug.Print "Laufwerksbuchstabe:    " _
          & .DriveLetter & " ----> " & str
        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 obj
  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 können Sie mithilfe der Eigenschaft `TotalSize` ermitteln. Den momentan freien Speicher liefert die Eigenschaft `FreeSpace`. Beide Angaben werden in Bytes ausgegeben. Diese können Sie in MBytes umrechnen, indem Sie das Ergebnis durch 1.048.576 teilen. Da das Ergebnis noch diverse Nachkommastellen aufweist, runden Sie es über den Einsatz der Funktion `Round` auf zwei Stellen hinter dem Komma. Das Stammverzeichnis Ihrer Festplatte können Sie über die Eigenschaft `RootFolder` abfragen. Die Info über das Dateisystem liefert Ihnen die Eigenschaft `FileSystem`.

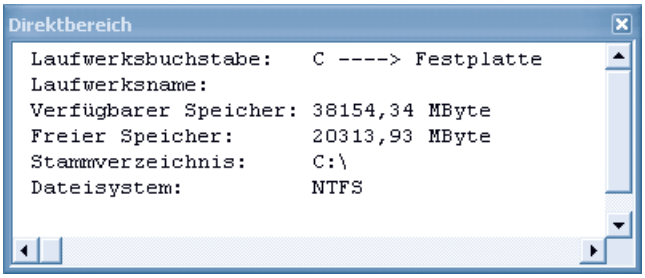


Abbildung 113: Die wichtigsten Eigenschaften der Festplatte

172 Textdatei einlesen

Über das `FileSystemObject` lassen sich auch Textdateien einlesen. Im folgenden Beispiel aus *Listing 202* wird die Textdatei ARTIKEL.TXT Zeile für Zeile eingelesen und im Direktfenster der Entwicklungsumgebung ausgegeben.

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

Sub TextdateiEinlesen()
  Dim fs As Object
  Dim file As Object
  Dim strZeile As String

  Set fs = CreateObject("Scripting.FileSystemObject")
  Set file = fs.OpenTextFile("C:\Eigene Dateien\Artikel.txt")

  Do Until file.AtEndOfStream
    strZeile = file.ReadLine
    Debug.Print strZeile
  Loop
  file.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 `file` zur Verfügung. In einer Schleife lesen Sie nun zeilenweise die `Textstream`-Variable aus. Geben Sie in diesem Beispiel den Inhalt jeder Zeile im Direktfenster der Entwicklungsumgebung aus. Über die Eigenschaft `AtEndOfStream` können Sie abfragen, wann das Ende der eingelesenen Textdatei erreicht ist. Über die Methode `Close` schließen Sie die noch geöffnete Textdatei.

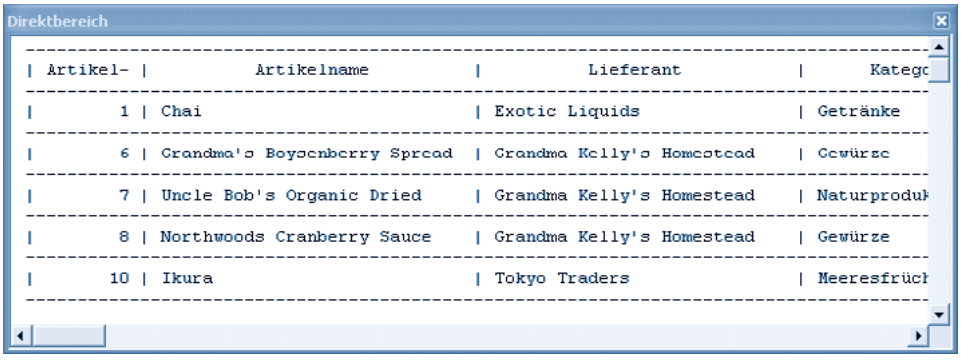


Abbildung 114: Die Textdatei wurde komplett eingelesen.

173 Textdatei schreiben

Mithilfe der Methode `CreateTextFile` können Sie eine Textdatei anlegen und über die Methode `WriteLine` einzelne Zeilen erfassen. Die Syntax der Methode `CreateTextFile` sieht wie folgt aus:

```
Objekt.CreateTextFile(Dateiname[, überschreiben[, Unicode]])
```

Teil	Beschreibung
`Objekt`	Erforderlich. Immer der Name eines `FileSystemObject`- oder `Folder`-Objekts.
`Dateiname`	Erforderlich. Zeichenfolgenausdruck, der die zu erstellende Datei identifiziert.
`überschreiben`	Optional. Boolescher Wert, der angibt, ob eine vorhandene Datei überschrieben werden soll. Wenn er `True` ist, wird eine vorhandene Datei überschrieben; ist er `False`, wird sie nicht überschrieben. Wenn der Parameter ausgelassen wird, wird der Wert auf `False` gesetzt, d. h., vorhandene Dateien werden 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 87: Die Argumente der Methode CreateTextFile

In der Prozedur aus *Listing 203* wird eine Textdatei angelegt und eine Zeile erfasst. Dabei wird das aktuelle Datum sowie die aktuelle Uhrzeit über die Funktion `Now` ermittelt.

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

Sub TextdateiSchreiben()
  Dim fs As Object
  Dim file As Object

  Set fs = CreateObject("Scripting.FileSystemObject")
  Set file = fs.CreateTextFile("C:\Eigene Dateien\held.ini", True)
  file.WriteLine (Now)
  file.Close
End Sub
```

Listing 203: Eine Textdatei anlegen und füllen

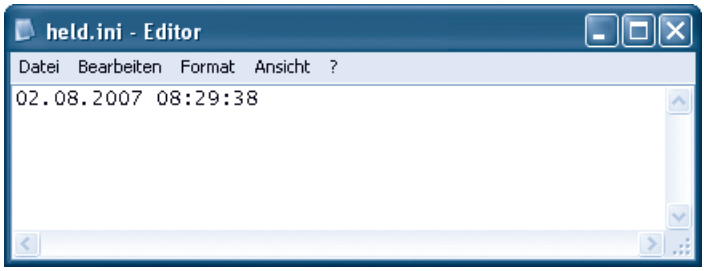


Abbildung 115: Eine »Ini-Datei« anlegen

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, die DAO *(Data Access Objects)* heißt.

In diesem Kapitel können Sie neben ausgewählten Aufgaben mit dem Objekt `DoCmd` auch Objekte, Methoden und Eigenschaften von ADO/DAO nachschlagen und deren Einsatz anhand praktischer Beispiele nachvollziehen.

Einfache Aufgaben mit DoCmd

Einige Aufgaben im Umfeld von Tabellen haben Sie bereits im *Kapitel Die Access-Objekte* dieses Buches durchgeführt. Die folgenden Seiten führen jetzt weitere typische Beispiele auf.

174 Tabellen öffnen

Als erste Aufgabe öffnen Sie in der Prozedur aus *Listing 204* über die Methode `OpenTable` die Tabelle KUNDEN.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 204: 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.

175 Datensatz finden

Über die Methode `FindRecord` haben Sie die Möglichkeit, schon beim Aufruf der Tabelle einen bestimmten Datensatz anzeigen zu lassen. In der Prozedur aus *Listing 205* wird die Tabelle KUNDEN geöffnet und der erste Datensatz angezeigt, in dem in einem beliebigen Feld der String »Ernst Handel« gefunden wird.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleÖffnenUndDatensatzFinden()
  On Error GoTo Fehler
  DoCmd.OpenTable "Kunden", acViewNormal
  DoCmd.FindRecord "Ernst Handel", acEntire, _
    True, acSearchAll, True, acAll
  Exit Sub
```

Listing 205: Einen Datensatz aktivieren

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

Listing 205: Einen Datensatz aktivieren (Forts.)

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 Feldes (Teil des Feldinhaltes = `acAnywhere`), nach Daten, die das gesamte Feld ausfüllen (gesamter Feldinhalt = `acEntire`), oder nach Daten, die sich am Anfang des Feldes befinden (Anfang des Feldinhaltes = `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 sowie nach Informationen zu suchen, 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 Feldern 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 in dem Datensatz zu beginnen, der auf den aktuellen Datensatz folgt. Wenn Sie dieses Argument nicht angeben, wird der Standardwert (`True`) verwendet.

Abbildung 116: Tabelle öffnen und Datensatz auswählen

176 Datensatz auswählen

Alternativ zur Methode `FindRecord` können Sie auch die Methode `GoToRecord` wie in *Listing 206* einsetzen, um gezielt einen bestimmten Datensatz, im Beispiel den letzten Datensatz der Tabelle, auszuwählen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleDatensatzAuswählen()
  On Error GoTo Fehler

  DoCmd.OpenTable "Kunden", acViewNormal
  DoCmd.GoToRecord acDataTable, "Kunden", acLast
  Exit Sub

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

Listing 206: Den letzten Datensatz der Tabelle auswählen

Die Methode `GoToRecord` hat dabei folgende Syntax:

```
GoToRecord(Objekttyp, Objektname, Datensatz, Offset)
```

Im Argument Objekttyp müssen Sie angeben, um welchen Objekttyp es sich dabei 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 KUNDEN.

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 gemacht.
- acGoTo: Mit dieser Konstante können Sie über einen numerischen Wert festlegen, auf welchem Datensatz positioniert werden soll.
- acLast: Der letzte Datensatz in der Tabelle wird zum aktuell ausgewählten gemacht.
- 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 gibt die Anzahl der Datensätze an, um die vorwärts- oder rückwärtsgeblättert 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.

Abbildung 117: Der letzte Datensatz in der Tabelle wurde ausgewählt.

177 Tabellen verknüpfen

Wenn Sie eine Tabelle aus einer externen Datenbank mit Ihrer aktiven Datenbank verknüpfen möchten, dann können Sie das über den Menübefehl DATEI/EXTERNE DATEN/TABELLEN VERKNÜPFEN erledigen. Innerhalb eines Programms schreiben Sie eine Prozedur, die diese Aufgabe automatisch durchführt.

In der folgenden Prozedur aus *Listing 207* wird die Tabelle PERSONAL aus der Datenbank NORDWIND.MDB im Verzeichnis C:\EIGENE DATEIEN in die aktuelle Datenbank verlinkt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 207: 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 darauffolgende Argument bestimmt, wie der Name der Tabelle lautet. Das letzte Argument beinhaltet den Namen der Tabelle für Ihre Datenbank.

178 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 konvertieren.

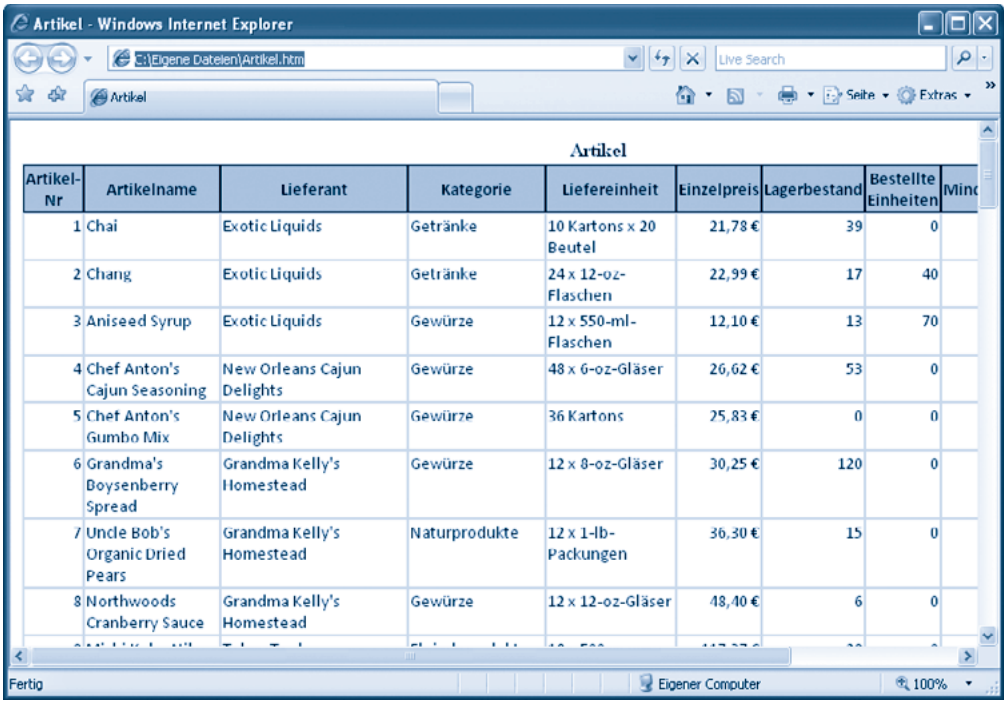
Tabelle ins HTML-Format übertragen

Im folgenden Beispiel aus *Listing 208* wird die Tabelle ARTIKEL ins HTML-Format gespeichert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=======================================================

Sub TabelleInHTMLexportieren()
  DoCmd.OpenTable "Artikel", acViewNormal
  DoCmd.OutputTo acOutputTable, "Artikel", acFormatHTML, "Artikel.htm"
  DoCmd.Close acTable, "Artikel"
End Sub
```

Listing 208: Tabelle ins HTML-Format exportieren

Mit der Methode `OutputTo` können Sie eine Tabelle im HTML-Format speichern, indem Sie die Konstante `acFormatHTML` verwenden.



Artikel-Nr	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbestand	Bestellte Einheiten	Mind
1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €	39	0	
2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	22,99 €	17	40	
3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	12,10 €	13	70	
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	26,62 €	53	0	
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €	0	0	
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	30,25 €	120	0	
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	36,30 €	15	0	
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	48,40 €	6	0	

Abbildung 118: Die Tabelle wurde ins HTML-Format exportiert.

Alle Tabellen der Datenbank ins HTML-Format übertragen

Wenn Sie die Prozedur aus *Listing 208* noch ein wenig erweitern, dann können Sie alle Ihre Tabellen ins HTML-Format abspeichern. Dazu starten Sie die Prozedur aus *Listing 209*:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=======================================================

Sub AlleTabellenInHTMLexportieren()
  Dim obj As AccessObject
  Dim db As Object

  Set db = Application.CurrentData
  DoCmd.Echo False
```

Listing 209: Alle Tabellen ins HTML-Format exportieren

```
  For Each obj In db.AllTables
    DoCmd.OpenTable obj.Name, acViewNormal
    DoCmd.OutputTo acOutputTable, obj.Name, acFormatHTML, obj.Name & ".htm"
    DoCmd.Close acTable, obj.Name
   Next obj

   DoCmd.Echo True
End Sub
```

Listing 209: Alle Tabellen ins HTML-Format exportieren (Forts.)

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 diese um und schließen 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

In Access existiert auch die Möglichkeit, Tabellen als XML-Datei zu exportieren. Im folgenden Beispiel aus *Listing 210* wird die Tabelle ARTIKEL als XML-Datei gespeichert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDoCmd
'=======================================================

Sub TabelleInXMLKonvertieren()
 Application.ExportXML acExportTable, "Artikel", "Artikel.xml", "artikel.xsd"
End Sub
```

Listing 210: Tabelle in eine XML-Datei exportieren

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.

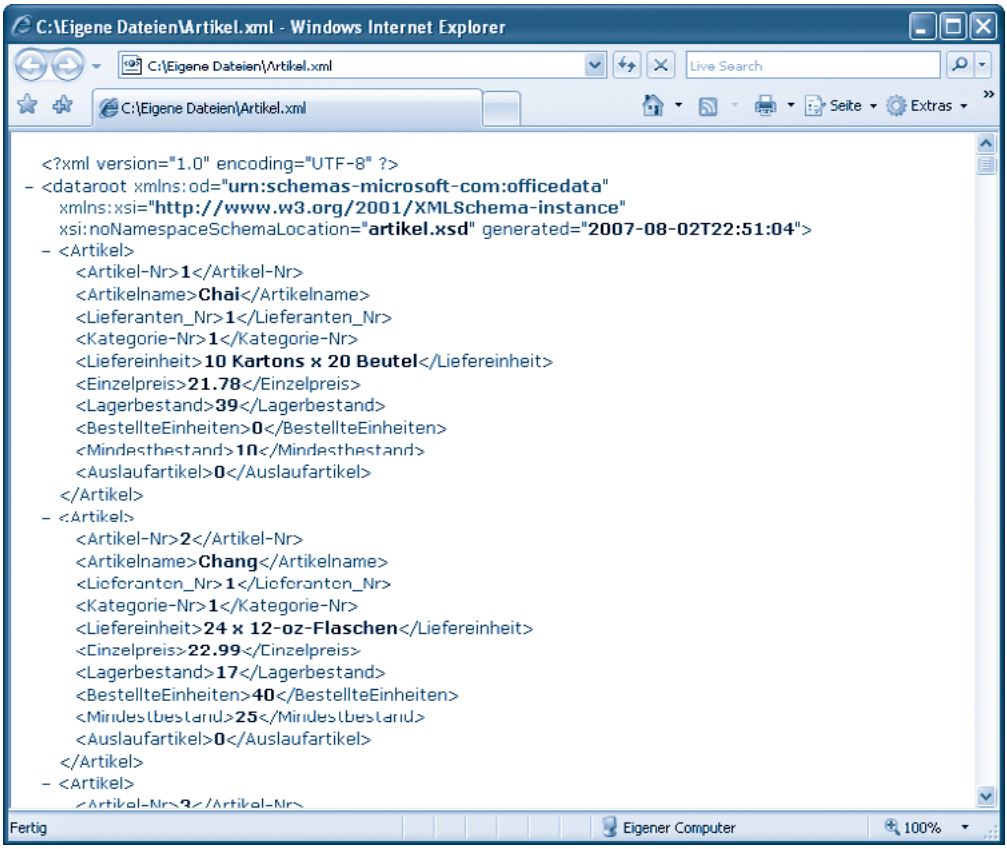


Abbildung 119: Die Tabelle wurde in eine XML-Datei exportiert.

Komplexere Aufgaben mit ADO/DAO & 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.

179 Datenbank öffnen und schließen (ADO)

Zu Beginn der »Lösungs-Staffel« sehen Sie in der Prozedur aus *Listing 211*, wie eine Datenbank über ADO-Methoden geöffnet und wieder geschlossen wird.

Sie können von einer Access-Datenbank aus eine weitere externe Datenbank im Hintergrund öffnen und auf deren Daten zugreifen. Abhängig davon, mit welcher Version von Access die Datenbank erstellt wurde, müssen Sie bei der Initialisierung der

Verbindung unterschiedliche Angaben machen. Wenn Sie mit einer Access-Version vor Access 2007 arbeiten bzw. eine entsprechende Datenbank im Format der genannten Version öffnen möchten, verwenden Sie die Microsoft Jet Engine. Als Provider geben Sie dann einen String in der Form "Microsoft.Jet.OLEDB.4.0" an.

Arbeiten Sie mit der zum Zeitpunkt des Drucks dieses Buches neuesten Version Access 2007, bzw. wollen Sie eine Datenbank im entsprechenden Format öffnen, geben Sie als Provider "Microsoft.ACE.OLEDB.12.0" an, um die Microsoft Office Access Database Engine 2007 zum Öffnen der Datenbank zu verwenden. Arbeiten Sie mit Access 2007, möchten aber eine Access-Datenbank im »alten« Format öffnen, bleibt es im Prinzip Ihnen überlassen, welche Schnittstelle Sie verwenden möchten.

Das folgende Listing zeigt, wie Sie eine externe Datenbank über die Microsoft Jet 4.0 Engine öffnen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub ExterneDatenbankÖffnen()
  Dim conn As ADODB.Connection

  Set conn = New ADODB.Connection

  With conn
    .Provider = "Microsoft.Jet.OLEDB.4.0"
    .Open "C:\Eigene Dateien\Nordwind.mdb"
  End With

  'Weitere Aktionen

  conn.Close
  Set conn = Nothing
End Sub
```

Listing 211: Externe Datenbank über die Microsoft Jet 4.0 Engine öffnen

Das Objekt Connection stellt über den Provider die Verbindung zur Datenbank her. Der Eigenschaft Provider weisen Sie den entsprechenden Namen des Providers zu. Mit der Methode Open öffnen Sie über den Provider die Datenbank. Mithilfe der Methode Close schließen Sie die Datenbank und beenden die Verbindung zum Provider.

Das folgende Listing zeigt, wie Sie eine externe Datenbank über die Microsoft Office Access Database Engine 2007 öffnen.

```
Sub ExterneDatenbankÖffnenAccess2007()
  Dim conn As ADODB.Connection

  Set conn = New ADODB.Connection

  With conn
    .Provider = "Microsoft.ACE.OLEDB.12.0"
    .Open "C:\Eigene Dateien\Nordwind.accdb"
  End With

  'Weitere Aktionen

  conn.Close
  Set conn = Nothing
End Sub
```

Listing 212: Externe Datenbank über die Microsoft Office Access Database Engine 2007 öffnen

Hinweis

Bei den weiteren Beispielen in diesem Buch beschränken wir uns der Einfachheit halber darauf, jeweils nur eine Verbindung über die Microsoft Jet 4.0 Engine darzustellen.

180 Tabelle öffnen und Dump ziehen (ADO)

Beim folgenden Beispiel aus *Listing 213* wird die Tabelle ARTIKEL geöffnet und der komplette Inhalt als »Dump« im Direktfenster der Entwicklungsumgebung ausgegeben. Sie nutzen dazu ein Objekt vom Typ `ADODB.Recordset`.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub TabelleÖffnenUndDumpErstellen()
  Dim rst As New ADODB.Recordset

  On Error GoTo Fehler
  rst.Open "Artikel", CurrentProject.Connection
  Debug.Print rst.GetString
```

Listing 213: Tabelle öffnen und auslesen

```
  rst.Close
  Exit Sub

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

Listing 213: Tabelle öffnen und auslesen (Forts.)

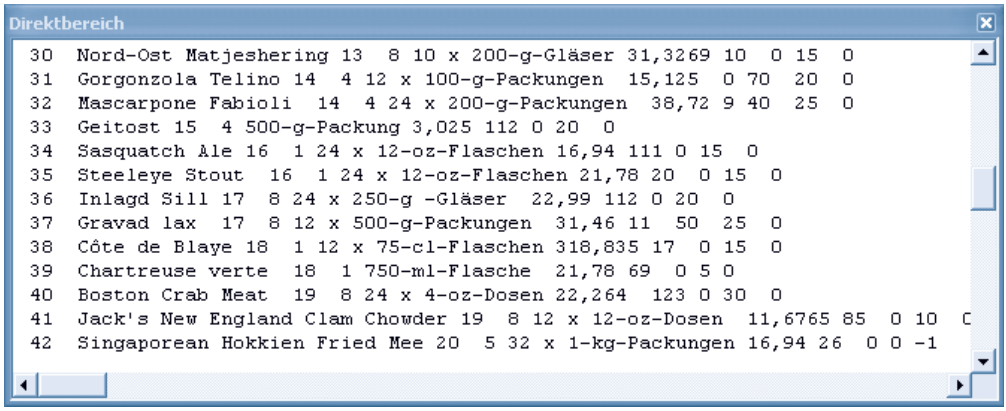


Abbildung 120: Inhalt 1:1 ausgeben

Über den Einsatz der Methode `Open` des Objekts `ADODB.Recordset` 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 »Schlüsselgruppe«. 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.

Tabelle 88: Die Argumente der Methode Open

Argument	Beschreibung
CursorType	▶ 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.
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 88: Die Argumente der Methode Open (Forts.)

Über die Methode GetString können Sie alle Daten aus der Tabelle auf einmal als String abrufen, sofern Sie keine weiteren Argumente angeben. Über die Methode Close schließen Sie die Tabelle.

181 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 die Prozedur aus *Listing 214*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub TabelleÖffnenUndDumpErstellen2()
  Dim rst As New ADODB.Recordset
```

Listing 214: Nur eine Auswahl von Sätzen schreiben

```
  On Error GoTo Fehler
  rst.Open "Artikel", CurrentProject.Connection
  Debug.Print rst.GetString(NumRows:=10, ColumnDelimeter:=";")
  rst.Close

  Set rst = Nothing
  Exit Sub

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

Listing 214: Nur eine Auswahl von Sätzen schreiben (Forts.)

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. In unserem Beispiel wünschen wir ein Semikolon als Trennzeichen.

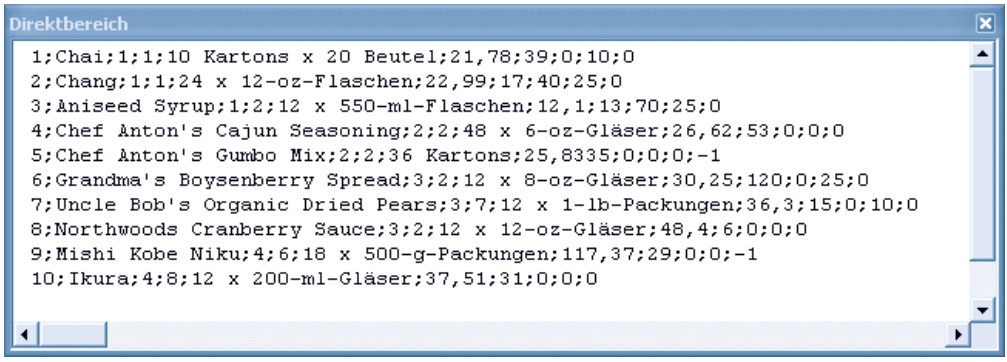


Abbildung 121: Ein anderes Trennzeichen angeben sowie eine Vorauswahl an Sätzen treffen

182 Nur bestimmte Felder einer Tabelle ausgeben (ADO)

Die Abfrage aus *Listing 213* kann noch weiter eingeschränkt werden, indem Sie mithilfe der SQL-Anweisung `SELECT` nur bestimmte Felder auswählen und ausgeben.

Im Beispiel aus *Listing 215* werden im Direktfenster der Entwicklungsumgebung nur die Datenfelder LAND, FIRMA und KONTAKTPERSON ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub NurBestimmteDatenAuswählen()
  Dim rst As New ADODB.Recordset

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  rst.Open "SELECT Land, Firma, Kontaktperson " & _
    " FROM Kunden ORDER BY Land", _
    CurrentProject.Connection
  Debug.Print rst.GetString
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 215: Nur bestimmte Felder ausgeben

Geben Sie bei der SQL-Anweisung die gewünschten Informationen, jeweils getrennt durch ein Komma, an. Über die SQL-Klausel `ORDER BY` können Sie eine Sortierreihenfolge festlegen. Durch diese Vorauswahl wählt die Methode `GetString` nur noch die so angegebenen Felder aus.

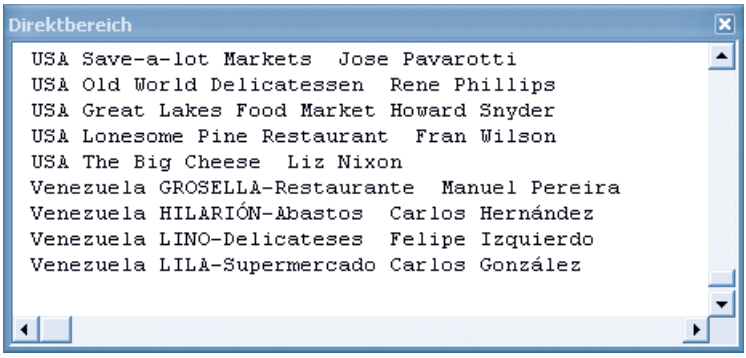


Abbildung 122: Es werden nur die Felder Land, Firma und Kontaktperson ausgegeben.

183 Tabelle öffnen und Daten ausgeben (DAO)

Sollen Daten aus einer Tabelle über DAO ausgelesen werden, dann greifen Sie auf die DAO-Objekte `DAO.Database` und `DAO.Recordset` zu.

Beim folgenden Beispiel aus *Listing 216* wird die Tabelle KUNDEN geöffnet und einige Felder in das Direktfenster der Entwicklungsumgebung eingelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub TabelleÖffnenUndAuslesenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Kunden", dbOpenTable, dbReadOnly)
  rst.MoveFirst

  Do Until rst.EOF
    Debug.Print rst.Fields("Firma").Value
    Debug.Print rst.Fields("Kontaktperson").Value
    Debug.Print rst.Fields("Straße").Value
    Debug.Print rst.Fields("Ort").Value
    Debug.Print rst.Fields("Land").Value & vbLf
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
  Set db = Nothing
  Exit Sub

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

Listing 216: Tabelle öffnen und auslesen (DAO)

Deklarieren Sie im ersten Schritt zwei Variable eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank 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 db = OpenDataBase("C:\Eigene Dateien\DB1.mdb")
```

Rufen Sie danach die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Wenden Sie die Methode `MoveFirst` an, um den ersten Datensatz in der Tabelle anzuspringen.

In einer `Do Until`-Schleife durchlaufen Sie so lange alle Datensätze der Tabelle, bis die `EOF`-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf einzelne Felder der Tabelle zu und geben diese über den Befehl `Debug.Print` im Direktfenster der Entwicklungsumgebung aus. Über die Methode `MoveNext` springen Sie am Ende jedes Schleifendurchlaufs zum nächsten Datensatz.

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, wodurch reservierte Ressourcen wieder freigegeben werden.

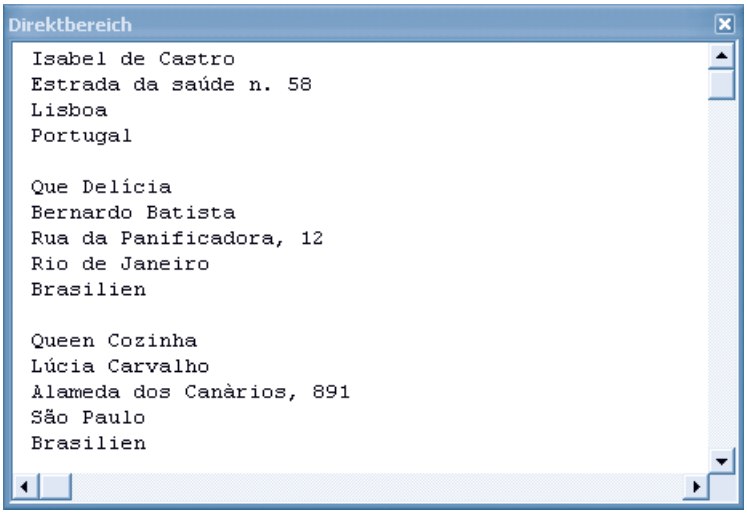


Abbildung 123: Alle Datensätze wurden ausgelesen.

184 Mehrere Datensätze in ein Array einlesen (ADO)

Sie können in nur einer Programmzeile mehrere Spalten und Zeilen einer Tabelle komplett als Array in eine Variable vom Typ `Variant` kopieren. Dazu wenden Sie die Methode `GetRows` an. Die Methode gibt ein zweidimensionales Array zurück. Der erste Index identifiziert das Feld und der zweite die Nummer der Zeile.

Die Syntax der Methode `GetRows` 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.
Start	Optional. Legt die Startposition fest, ab der Datensätze eingelesen werden sollen.
Fields	Legt die Spalten fest, die eingelesen werden sollen.

Tabelle 89: Die Argumente der Methode GetRows

Im folgenden Beispiel aus *Listing 217* werden aus der Tabelle ARTIKEL die ersten zehn Datensätze mit den ersten drei Spalteninhalten ausgelesen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MehrereSätzeAuslesen()
  Dim rst 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
  rst.Open "Artikel", CurrentProject.Connection
  varDat = rst.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
  rst.Close
  Exit Sub

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

Listing 217: Mehrere Datensätze in eine Variable kopieren (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 im Direktbereich ausgeben.

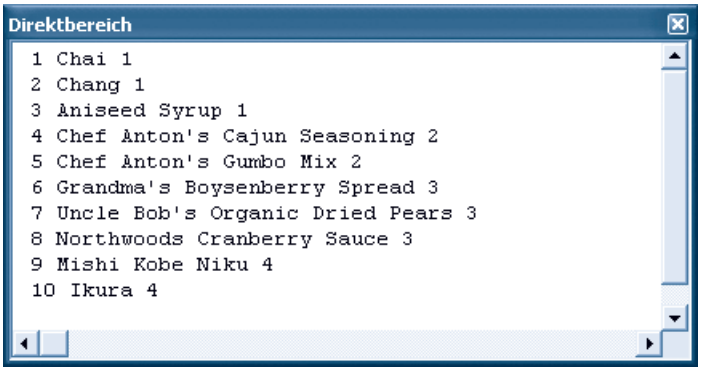


Abbildung 124: Die ersten zehn Datensätze

185 Datensätze in Datenfeld einlesen (DAO)

Die Methode GetRows ist auch in DAO einsetzbar. In der Prozedur aus *Listing 218* werden die ersten zehn Datensätze sowie die ersten drei Spalten der Tabelle ARTIKEL ausgelesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatenInFeldEinlesenDAO()
  Dim db As DAO.Database
  Dim rst 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 db = CurrentDb()
  Set rst = db.OpenRecordset("Artikel", dbOpenSnapshot)

  varDat = rst.GetRows(intGesamtZeilen)
```

Listing 218: Mehrere Datensätze in eine Variable kopieren (DAO)

```
  For intZeilen = 0 To intGesamtZeilen - 1
    For intSpalten = 0 To 2
      Debug.Print varDat(intSpalten, intZeilen) & " ";
    Next intSpalten
    Debug.Print
  Next intZeilen
  rst.Close
  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 218: Mehrere Datensätze in eine Variable kopieren (DAO) (Forts.)

Bei der Prozedur aus *Listing 218* wurde für die Bekanntgabe der aktuellen Datenbank die Kurzform

```
Set db = CurrentDb()
```

gewählt. Diese Kurzform ist gleichbedeutend mit der etwas längeren Form

```
Set DBS =  Application.CurrentDb.
```


186 SQL-Abfragen ausführen (ADO)

Über die Methode `Execute` können Sie eine SQL-Anweisung oder eine beliebige andere Abfrage ausführen. 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 90: Die Argumente der Methode Execute

Im folgenden Beispiel aus *Listing 219* werden alle Datensätze aus der Tabelle KUNDEN extrahiert, bei denen die Kunden aus DEUTSCHLAND kommen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeAuslesen()
  Dim rst As ADODB.Recordset

  Set rst = CurrentProject.Connection.Execute _
    ("SELECT * FROM Kunden WHERE Land ='Deutschland'" _
    , , adCmdText)

  Do Until rst.EOF
    Debug.Print rst!Firma & vbTab & _
      rst!Kontaktperson & _
      vbTab & rst!Land
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
End Sub
```

Listing 219: Datensätze auslesen

Bei der Prozedur aus *Listing 219* werden mithilfe der SQL-Anweisung `SELECT` die Datensätze ausgelesen, bei denen das Feld LAND den Wert »Deutschland« enthält. Die Auswahl wird hierbei durch das Schlüsselwort `WHERE` festgelegt. Die SQL-Anweisung wird als String an die Methode `Execute` übergeben. Das Ergebnis steht nach der Abfrage im `Recordset`-Objekt `rst` zur Verfügung und kann von dort ausgelesen werden. Das Auslesen wird im Beispiel über eine `Do Until`-Schleife realisiert, bei der über die Eigenschaft `EOF` der letzte gefundene Satz im `Recordset`-Objekt festgestellt wird. Vom jeweils aktuell ausgewählten Datensatz werden innerhalb der Schleife die Daten der Felder FIRMA, KONTAKTPERSON und LAND über den Befehl `Debug.Print` im Direktfenster der Entwicklungsumgebung ausgegeben. Über die Methode `MoveNext` wird jeweils der nächste Datensatz im `Recordset`-Objekt ausgewählt.

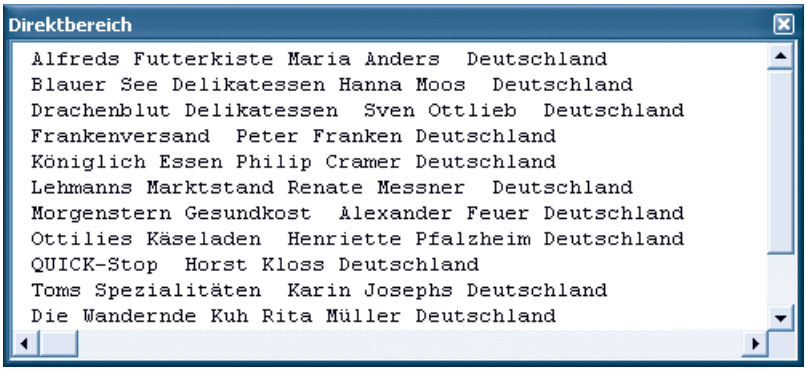


Abbildung 125: Einige Datensätze auslesen

187 Unikatliste aus Tabelle erstellen (ADO)

Um eine Unikatliste 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 220* wird die Tabelle KUNDEN ausgewertet. Ziel ist es, eine Liste zu erstellen, in der alle Länder der Tabelle nur einmal vorkommen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub KeineDoppelten()
  Dim rst As ADODB.Recordset

  Set rst = _
    CurrentProject.Connection.Execute _
    ("SELECT DISTINCT LAND FROM KUNDEN" _
     , , adCmdText)

  Do Until rst.EOF
    Debug.Print rst!Land
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
End Sub
```

Listing 220: Eine Unikatliste erstellen (ADO)

Über die Anweisung `DISTINCT`, die Sie vor dem Feld angeben, wird sichergestellt, dass im Ergebnis jeder Satz nur einmal auftaucht.

Abbildung 126: Die Unikat-Länderliste wurde erstellt.

In der Prozedur aus *Listing 220* wurde sichergestellt, dass jedes Land aus der Tabelle KUNDEN nur einmal im Direktfenster ausgegeben wird. Diese Prozedur kann insofern erweitert werden, dass die Einmaligkeit für das Land in Kombination mit der Stadt gelten soll. In der Prozedur aus *Listing 221* werden also nur Datensätze ausgegeben, die bei Land und Stadt zusammen keine Doppelung aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub KeineDoppelten2()
  Dim rst As ADODB.Recordset

  Set rst = _
    CurrentProject.Connection.Execute _
    ("SELECT DISTINCT LAND, ORT FROM KUNDEN" _
    , , adCmdText)

  Do Until rst.EOF
    Debug.Print rst!Land & vbTab & rst!Ort
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
End Sub
```

Listing 221: Keine doppelten Länder und Orte (ADO)

Die Anweisung DISTINCT kann erweitert werden, indem alle folgenden Felder, getrennt durch ein Komma, aufgeführt werden.

Abbildung 127: Jeder Ort in einem Land wird nur einmal genannt.

188 Unikatliste aus Tabelle erstellen (DAO)

Um eine Unikatliste über eine DAO-Prozedur zu erstellen, verwenden Sie ebenso das SQL-Schlüsselwort Distinct.

In der Prozedur aus *Listing 222* werden aus der Tabelle KUNDEN alle Ortsnamen ausgelesen. Dabei darf im Ergebnis kein Ortsname doppelt vorkommen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub KeineDoppeltenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("SELECT DISTINCT ORT FROM KUNDEN", _
    dbOpenSnapshot)

  With rst
  Do Until .EOF
    Debug.Print .Fields("Ort")
    .MoveNext
  Loop
```

Listing 222: Eine Unikatliste erstellen (DAO)

```
  .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 222: Eine Unikatliste erstellen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank 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` aus, 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 so lange alle Datensätze, die jetzt als Ergebnis im Objekt `rst` zur Verfügung stehen, bis die `EOF`-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` gehen Sie nach jedem Schleifendurchlauf jeweils zum nächsten Datensatz im Recordset-Objekt weiter.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

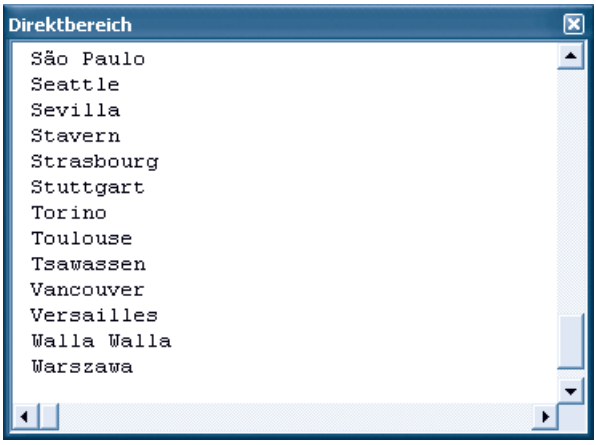


Abbildung 128: Jeder Ort taucht nur einmal auf.

189 Datensätze sortieren (ADO)

Um Datensätze sortiert auszugeben, setzen Sie die SQL-Anweisung 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 223* 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.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeSortiertAuslesen()
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set rst = _
    CurrentProject.Connection.Execute _
    ("SELECT * FROM Artikel WHERE Einzelpreis > 20 ORDER BY Einzelpreis ASC" _
    , , adCmdText)

  Do Until rst.EOF
    Debug.Print rst!Artikelname & vbLf & _
                rst!Einzelpreis & vbLf & _
                rst!Lagerbestand & vbLf
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 223: Datensätze sortiert ausgeben

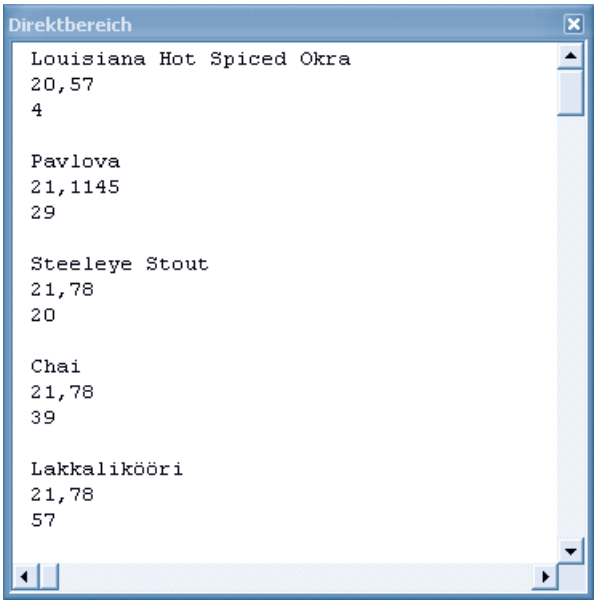


Abbildung 129: Der billigste Artikel steht ganz oben.

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 224* wird die Tabelle ARTIKEL geöffnet und die Datensätze werden nach dem Artikelnamen sortiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeSortieren()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  With rst
    .CursorLocation = adUseClient
    .Open "Artikel", conn, adOpenKeyset, adLockOptimistic
    .Sort = "Artikelname ASC"
```

Listing 224: Datensätze über die Eigenschaft Sort sortieren

```
    Do While Not .EOF
      Debug.Print .Fields("Artikel-Nr.").Value & _
        " " & .Fields("Artikelname").Value
      .MoveNext
    Loop

    .Close
  End With
  conn.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 224: Datensätze über die Eigenschaft Sort sortieren (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 eine aufsteigende Sortierung oder `DESC` für eine absteigende Sortierung angeben. Dabei lassen Sie nach dem Feldnamen ein Leerzeichen und hängen die gewünschte Sortierkonstante an. Wenn Sie diese Konstante nicht angeben, wird standardmäßig absteigend sortiert.

190 Datensätze sortieren (DAO)

Auch beim Anwenden von DAO müssen Sie nicht auf SQL-Anweisungen 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 225* wird die Tabelle ARTIKEL nach EINZELPREIS sortiert im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================
```

Listing 225: Datensätze sortieren und ausgeben (DAO)

```
Sub DatensätzeSortiertAuslesenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset _
    ("SELECT * FROM Artikel WHERE Einzelpreis > 20 ORDER BY Einzelpreis ASC" _
    , dbOpenSnapshot)

  With rst
  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 db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 225: Datensätze sortieren und ausgeben (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Danach rufen Sie die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Dabei übergeben Sie die SQL-Anweisung, die die Filterung und Sortierung angibt, direkt beim Öffnen der Tabelle. Als Ergebnis stehen nun alle Datensätze der Tabelle im Objekt `rst` zur Verfügung und können dort abgefragt werden.

In einer `Do Until`-Schleife durchlaufen Sie so lange alle Datensätze des Recordsets, bis die `EOF`-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` gehen Sie nach jedem Schleifendurchlauf zum nächsten Datensatz des Recordsets.

Schließen Sie am Ende der Prozedur die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

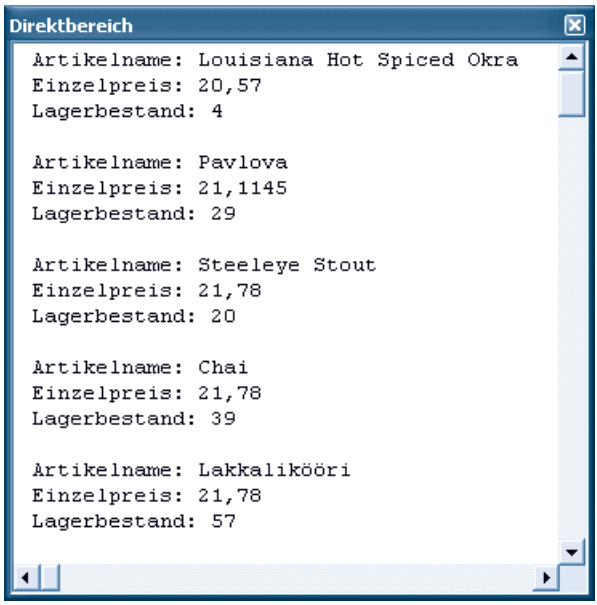


Abbildung 130: Der billigste Artikel steht ganz oben.

191 Tabellen zusammenfassen (ADO)

Mit der SQL-Anweisung `UNION` können Sie mehrere Tabellen zusammenfassen.

Die nachfolgende Prozedur aus *Listing 226* liest aus den beiden Tabellen KUNDEN und LIEFERANTEN die Informationen ORT, FIRMA und KONTAKTPERSON heraus.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MehrereTabellenAbfragen()
  Dim rst As New ADODB.Recordset

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open "SELECT Ort, Firma, Kontaktperson " & _
    "FROM Kunden UNION SELECT Ort, Firma, " & _
    "Kontaktperson FROM " & _
    "Lieferanten ORDER BY Ort, Firma", _
    CurrentProject.Connection
```

Listing 226: Mehrere Tabellen abfragen

```
    Do Until .EOF
      Debug.Print "Ort: " & .Fields("Ort").Value & _
        "  Firma: " & .Fields("Firma").Value & _
        " Kontaktperson: " & .Fields("Kontaktperson")
      rst.MoveNext
    Loop
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 226: Mehrere Tabellen abfragen (Forts.)

Über das Schlüsselwort `UNION` verbinden Sie die zwei Tabellen KUNDEN und LIEFERANTEN.

Abbildung 131: Das Ergebnis enthält Daten aus zwei Tabellen.

192 Verknüpfte Tabellen abfragen (ADO)

Wenn Sie im Access-Hauptfenster den Dialog BEZIEHUNGEN anzeigen lassen, dann werden Sie feststellen, dass einige Tabellen, wie die Tabelle ARTIKEL und die Tabelle LIEFERANTEN, über die Lieferanten-Nr. verknüpft sind.

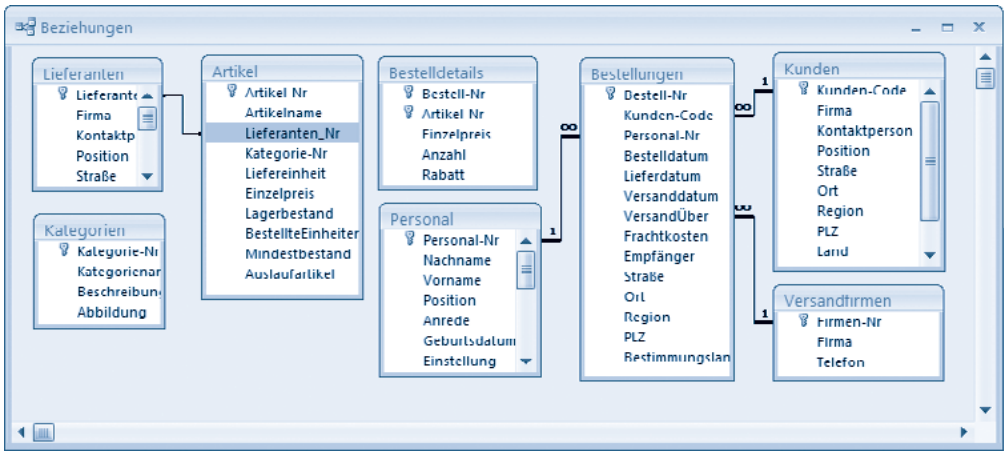


Abbildung 132: Die Tabelle Artikel ist mit der Tabelle Lieferanten verknüpft.

Wenn Sie nun beispielsweise die Tabelle ARTIKEL auslesen, dann bekommen Sie als Lieferant nicht den Namen des Lieferanten, sondern eine Lieferanten-Nr. Mit dieser Nummer kann dann in der Tabelle LIEFERANTEN der Name des Lieferanten ermittelt werden.

Sollen also Informationen aus beiden Tabellen in eine Auswertung einfließen, dann müssen beide Tabellen in die SELECT-Anweisung mit integriert werden. Sehen Sie sich die Prozedur aus *Listing 227* einmal etwas näher an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MehrereVerknüpfteTabellenAbfragen()
  Dim rst As New ADODB.Recordset

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .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 227: Verknüpfte Tabellen abfragen (ADO)

```
      Debug.Print "Kontaktperson: " & .Fields("Kontaktperson") & vbLf
      rst.MoveNext
    Loop
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 227: Verknüpfte Tabellen abfragen (ADO) (Forts.)

Geben Sie bei der SELECT-Anweisung vor jedem Feld auch den Namen der Tabelle an, aus dem das Feld stammt. Nach der FROM-Klausel geben Sie die Namen der beiden Tabellen an, aus denen Sie Daten beziehen möchten. Nach der WHERE-Klausel geben Sie in Form einer Zuweisung über ein Gleichheitszeichen die Namen der Felder an, über die die beiden Tabellen miteinander verknüpft sind. In unserem Fall heißen die Felder, wie üblich, in den beiden verknüpften Tabellen gleich, nämlich LIEFERANTEN_NR.

Abbildung 133: Der Lieferantenname wird korrekt angezeigt.

193 Verknüpfte Tabellen abfragen (DAO)

Die gleiche Aufgabe wie in *Listing 227* wird jetzt in DAO über den Einsatz der Prozedur aus *Listing 228* gelöst.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub MehrereVerknüpfteTabellenAbfragenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  Set db = Application.CurrentDb
  On Error GoTo Fehler

  Set rst = db.OpenRecordset _
    ("SELECT Artikel.Artikelname, Lieferanten.Firma, " & _
    "Lieferanten.Kontaktperson FROM Artikel, Lieferanten " & _
    "WHERE Artikel.Lieferanten_Nr=Lieferanten.Lieferanten_Nr", dbOpenSnapshot)

  With rst
  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 db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 228: Verknüpfte Tabellen abfragen (DAO)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Dann rufen Sie die Methode `OpenRecordset` auf, 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 dem das Feld stammt. Nach der FROM-Klausel geben Sie die Namen der beiden Tabellen an, aus denen Sie Daten abrufen möchten. Nach der WHERE-Klausel geben Sie in Form einer Zuweisung über ein Gleichheitszeichen die Namen der Felder an, über die die beiden Tabellen miteinander verknüpft sind.

In einer Do Until-Schleife durchlaufen Sie so lange alle Datensätze der Tabelle, bis die EOF-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung Fields auf die einzelnen Felder zu. Über die Methode MoveNext gehen Sie nach jedem Schleifendurchlauf zum nächsten Datensatz.

Schließen Sie die Tabelle über die Methode Close und heben Sie die Objektverweise auf.

194 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 229* werden die fünf billigsten Artikel aus dem Sortiment der Tabelle ARTIKEL ermittelt und angezeigt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MinWerteFinden()
  Dim rst As New ADODB.Recordset
  Dim strSql As String

  strSql = _
    "SELECT TOP 5 Einzelpreis, Artikelname, " & _
    "Lagerbestand FROM Artikel ORDER BY Einzelpreis"
  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open strSql, CurrentProject.Connection
    Do Until rst.EOF
      Debug.Print "Artikelname:  " & _
        .Fields("Artikelname").Value & vbLf & _
        "Lagerbestand: "; .Fields("Lagerbestand").Value & _
        vbLf & "Einzelpreis:  " & _
        .Fields("Einzelpreis").Value & vbLf
      rst.MoveNext
```

Listing 229: Minimalwerte aus einer Tabelle ermitteln

```
    Loop
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 229: 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.

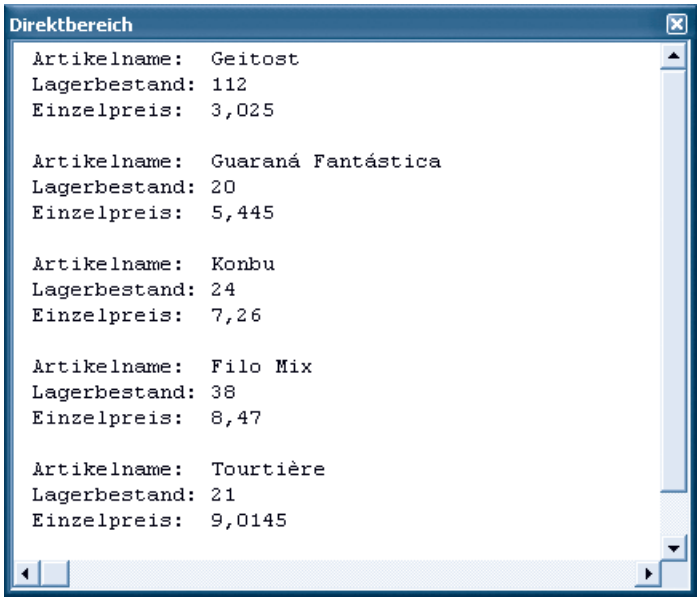


Abbildung 134: Die fünf billigsten Artikel im Sortiment

Top 3 nach Lagerbestand

Sollen hingegen die drei Artikel mit dem größten Lagerbestand ermittelt werden, dann starten Sie die Prozedur aus *Listing 230*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================
```

Listing 230: Die Maximalwerte aus einer Tabelle ermitteln

```
Sub MaxWerteFinden()
  Dim rst As New ADODB.Recordset
  Dim strSql As String

  strSql = _
    "SELECT TOP 3 Einzelpreis, Artikelname, " & _
    "Lagerbestand FROM Artikel ORDER BY Lagerbestand DESC"
  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open strSql, CurrentProject.Connection
    Do Until rst.EOF
      Debug.Print "Artikelname:  " & _
        .Fields("Artikelname").Value & vbLf & _
        "Lagerbestand: "; .Fields("Lagerbestand").Value & _
        vbLf & "Einzelpreis:  " & _
        .Fields("Einzelpreis").Value & vbLf
      rst.MoveNext
    Loop
   .Close
   End With
   Set rst = Nothing
   Exit Sub

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

Listing 230: Die Maximalwerte aus einer Tabelle ermitteln (Forts.)

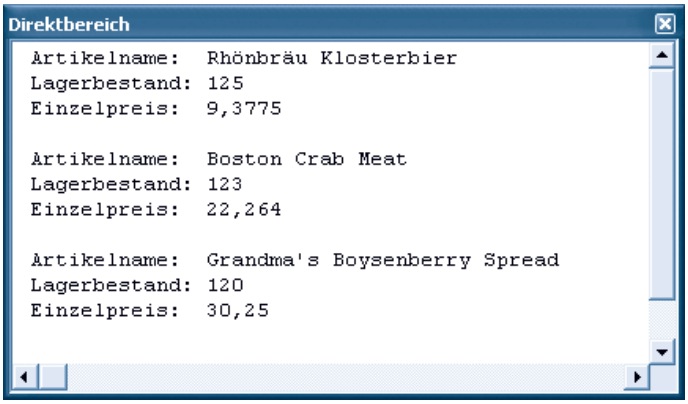


Abbildung 135: Die Lagerhüter im Sortiment

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

In der folgenden Prozedur aus *Listing 231* wird der billigste Artikel der Tabelle ARTIKEL wertmäßig abgefragt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub MinWertFinden()
  Dim rst As New ADODB.Recordset
  Dim strSql As String

  strSql = _
    "SELECT Min(Einzelpreis) FROM Artikel"

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open strSql, CurrentProject.Connection
    .MoveFirst
    MsgBox "Der niedrigste Preis ist " & .Fields(0).Value
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 231: Der niedrigste Preis wird ermittelt.

Fügen Sie die Funktion `Min` gleich nach der Anweisung `SELECT` ein und setzen das Feld in Klammern, dessen niedrigsten Wert Sie ermitteln möchten. Nach der Suche wenden Sie die Methode `MoveFirst` an, um den ersten und normalerweise einzigen Datensatz im Recordset aufzurufen. Über die `Fields`-Auflistung können Sie diesen Wert ausgeben.

Lagerhüter ermitteln

Um den größten Lagerhüter im Sortiment zu ermitteln, starten Sie die Prozedur aus *Listing 232*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MaxWertFinden()
  Dim rst As New ADODB.Recordset
  Dim strSql As String

  strSql = _
    "SELECT Max(Lagerbestand) FROM Artikel"

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open strSql, CurrentProject.Connection
    .MoveFirst
    MsgBox "Die höchste Lagermenge ist " & .Fields(0).Value
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 232: Den größten Lagerhüter ermitteln

Neben den Aggregatfunktionen `Min` und `Max` gibt es in *Tabelle 91* weitere Funktionen, die Sie einsetzen können.

Funktion	Kurzbeschreibung
Avg	Mittelwertberechnung.
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 91: Die verfügbaren Aggregatfunktionen

Lagerbestandsdurchschnitt errechnen

Im folgenden Beispiel aus *Listing 233* wird der durchschnittliche Lagerbestand und der durchschnittliche Preis aller Artikel errechnet und im Direktfenster ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub AVGWerte()
  Dim rst As New ADODB.Recordset
  Dim strSql As String

  strSql = _
    "SELECT AVG(Lagerbestand), AVG(Einzelpreis) FROM Artikel"

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  With rst
    .Open strSql, CurrentProject.Connection
     Debug.Print .GetString
    .Close
  End With
  Set rst = Nothing
  Exit Sub

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

Listing 233: Durchschnittsberechnungen durchführen (ADO)

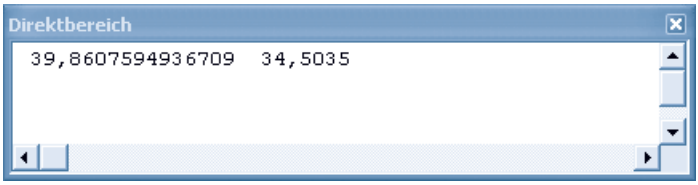


Abbildung 136: Durchschnittlicher Lagerbestand sowie durchschnittlicher Preis

Eine weitere Möglichkeit, Extremwerte zu ermitteln, ist der Einsatz der »D-Funktionen«, die Sie in *Tabelle 92* einsehen können.

Funktion	Kurzbeschreibung
DAvg	Mittelwertberechnung.
DCount	Datensatzanzahl wird ermittelt.
DLookUp	Zeigt einen Wert eines Feldes 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 92: Die D-Funktionen im Überblick

Den ältesten Mitarbeiter ermitteln

Im folgenden Beispiel aus *Listing 234* wird das Geburtsdatum des ältesten Mitarbeiters in der Tabelle PERSONAL ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWertFindenVar2()
  MsgBox "Der älteste Mitarbeiter ist am : " & _
    DMin("Geburtsdatum", "Personal") & _
    " geboren!", vbInformation
End Sub
```

Listing 234: Das am weitesten zurückliegende Geburtsdatum wird über die Funktion Min ermittelt.

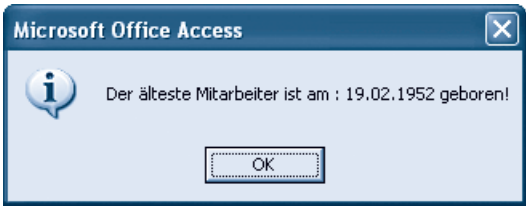


Abbildung 137: Auch Altersberechnungen sind möglich.

Soll das Alter in Jahren berechnet werden, dann erweitern Sie die Prozedur aus *Listing 234* wie in *Listing 204* angegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub MinWertFindenVar2Erweitert()
  MsgBox DateDiff("YYYY", DMin("Geburtsdatum", "Personal"), Date)
End Sub
```

Listing 235: Rechnen ist auch mit DMin möglich.

195 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 Beispiel aus *Listing 236* werden die fünf billigsten Artikel aus der Tabelle ARTIKEL ermittelt und im Direktfenster ausgegeben.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub MinWerteFindenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset _
    ("SELECT TOP 5 Einzelpreis, Artikelname, " & _
    "Lagerbestand FROM Artikel ORDER BY Einzelpreis", dbOpenSnapshot)

  With rst
    Do Until .EOF
    Debug.Print "Artikelname:  " & _
      .Fields("Artikelname").Value & vbLf & _
      "Lagerbestand: "; .Fields("Lagerbestand").Value & _
       vbLf & "Einzelpreis:  " & _
      .Fields("Einzelpreis").Value & vbLf
      .MoveNext
```

Listing 236: Die fünf billigsten Artikel ermitteln (DAO)

```
    Loop
    .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 236: Die fünf billigsten Artikel ermitteln (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist. Dann rufen Sie die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Dabei übergeben Sie die SQL-Anweisung schon beim Öffnen der Tabelle, und zwar als String in der Form `"SELECT TOP 5 Einzelpreis, Artikelname, Lagerbestand FROM Artikel ORDER BY Einzelpreis"`. Danach stehen die fünf billigsten Artikel im Objekt `rst`. 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.

196 Daten gruppieren (ADO)

Um komplexe Tabellen auszuwerten und dabei die Daten nach bestimmten Kriterien 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. Der Aufbau dieser Tabelle können Sie aus *Abbildung 138* entnehmen.

Abbildung 138: Die Ausgangstabelle fürs Gruppieren

Möchten Sie jetzt eine Auswertung haben, die die Summen der Gehälter einer jeden Arbeitsgruppe ermittelt, dann starten Sie die Prozedur aus *Listing 237*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeGruppieren()
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set rst = _
    CurrentProject.Connection.Execute _
    ("SELECT Arbeitsgruppe, sum(Gehalt) FROM Gruppen GROUP BY Arbeitsgruppe" _
     , , adCmdText)

  Debug.Print rst.GetString
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 237: Die Gehälter nach Arbeitsgruppe summieren und gruppieren

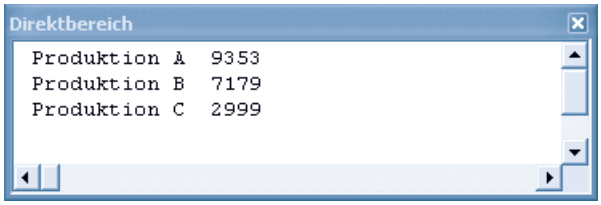


Abbildung 139: Die Gesamtgehälter pro Arbeitsgruppe

Soll das Ergebnis noch weiter eingeschränkt werden, beispielsweise dass nur Arbeitsgruppen aufgelistet werden, die eine Gesamtsumme des Gehalts größer als 3.000 haben, dann integrieren Sie die SQL-Klausel `HAVING`. Die Umsetzung dieser Klausel können Sie in *Listing 238* sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================
```

Listing 238: Die Gruppierung einschränken

```
Sub DatensätzeGruppieren2()
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set rst = _
    CurrentProject.Connection.Execute _
    ("SELECT Arbeitsgruppe, sum(Gehalt) FROM Gruppen " & _
    " GROUP BY Arbeitsgruppe HAVING SUM(Gehalt)> 3000" _
    , , adCmdText)

  Debug.Print rst.GetString
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 238: Die Gruppierung einschränken (Forts.)

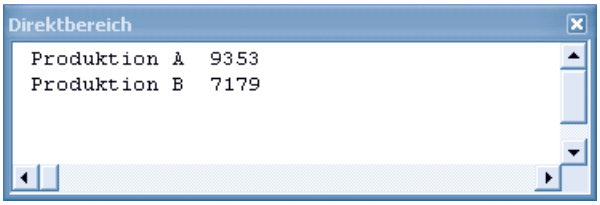


Abbildung 140: Es werden nur bestimmte Arbeitsgruppen angezeigt.

197 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 der Tabelle gesetzt. 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 93: 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 hört am Ende der Tabelle auf, 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 angeben, der anzeigt bei welchem Datensatz die Suche in der Tabelle beginnen soll.

Tabelle 93: Die Argumente der Methode Find (Forts.)

Im folgenden Beispiel aus *Listing 239* wird eine Artikelbezeichnung über eine Inputbox erfasst. Danach öffnen Sie die Tabelle ARTIKEL und durchsuchen sie. Wird der gesuchte Artikel gefunden, dann wird der entsprechende Artikel im Direktfenster ausgegeben.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensatzFinden()
  Dim rst 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 rst = New ADODB.Recordset
  With rst
    .Open Source:="Artikel", _
      ActiveConnection:=CurrentProject.Connection, _
      CursorType:=adOpenKeyset, _
      LockType:=adLockOptimistic

    .Find Criteria:=strEingabe, _
      SearchDirection:=adSearchForward
    If Not .EOF Then
      Do While Not .EOF
```

Listing 239: 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 rst = Nothing
  Exit Sub

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

Listing 239: Datensatz finden (Forts.)

Im ersten Schritt fragen Sie über einen Eingabedialog die Artikelbezeichnung ab, nach der in der Tabelle ARTIKEL gesucht werden soll. Bilden Sie im Anschluss daran den Suchstring. Achten Sie darauf, dass der Suchstring sowohl aus dem eigentlichen Suchbegriff als auch aus dem Feldnamen der Tabelle gebildet werden muss. Dies geschieht in der Zeile `strEingabe = "Artikelname='" & strEingabe & "'"`.

Öffnen Sie die Access-Datenbank mithilfe des Objekts `Connection`. Definieren Sie daraufhin ein `ADODB.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 die `Connection`-Eigenschaft des `CurrentProject`-Objekts.

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 »Schlüsselgruppe«. 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.

Über das Argument `LockType` bestimmen Sie, welches Sperrverfahren beim Öffnen der Tabelle eingesetzt werden 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 für die gesamte Zeit der Bearbeitung für andere Anwender vollständig gesperrt. Dadurch wird auf jeden Fall das erfolgreiche Bearbeiten der Datensätze sichergestellt.
- `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 die Methode `Find` ein, um in der Tabelle nach dem Datensatz zu suchen, der dem angegebenen Kriterium entspricht, indem Sie der Methode unter anderem den Suchstring `strEingabe` übergeben. Ist das Kriterium erfüllt, wird der gefundene Datensatz zum aktuellen Datensatz der Tabelle. Andernfalls wird der Zeiger auf das Ende der Tabelle gesetzt.

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, steht das Suchergebnis in der Objektvariablen `rst`.

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.

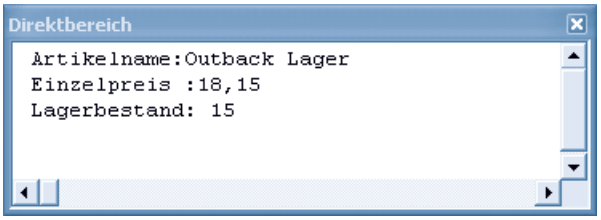


Abbildung 141: Der Datensatz Outback Lager wurde gefunden.

198 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 94* 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 94: Die Unterschiede der Find-Methoden

Ein Suchtreffer

Beim Beispiel aus *Listing 240* wird die Tabelle KUNDEN nach einer bestimmten Kontaktperson durchsucht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeFindenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Kunden", dbOpenDynaset)

  rst.FindFirst "Kontaktperson =  'Victoria Ashworth'"
```

Listing 240: Datensatz finden über DAO

```
  If rst.NoMatch = True Then
    MsgBox "Kontaktperson konnte nicht gefunden werden!"
  Else
    Debug.Print rst.Fields("Firma").Value
    Debug.Print rst.Fields("Kontaktperson").Value
    Debug.Print rst.Fields("Straße").Value
    Debug.Print rst.Fields("Ort").Value
    Debug.Print rst.Fields("Land").Value & vbLf
  End If

  rst.Close
  Set rst = Nothing
  Set db = Nothing
Exit Sub
```

Listing 240: Datensatz finden über DAO (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist. Danach rufen Sie die Methode `OpenRecordset` auf, 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 ein passender Datensatz gefunden werden konnte. Bei einer erfolgreichen Suche greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu und geben diese über die Anweisung `Debug.Print` im Direktfenster der Entwicklungsumgebung aus.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

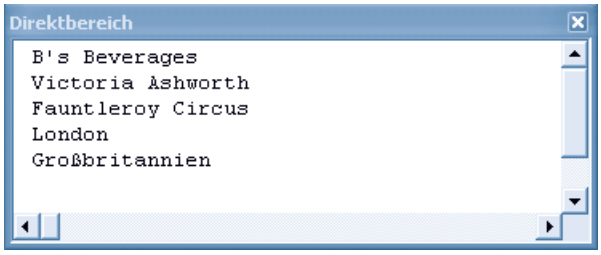


Abbildung 142: Die Kontaktperson konnte gefunden werden.

Mehrere Suchtreffer

Selbstverständlich gibt es auch Suchdurchläufe, bei denen mehrere Datensätze gefunden werden können. So wird beim folgenden Beispiel aus *Listing 241* 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
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub MehrereDatensätzeFindenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Kunden", dbOpenDynaset)

  rst.FindFirst "Land =  'Deutschland'"

  Do Until rst.NoMatch = True
    Debug.Print rst.Fields("Firma").Value
    Debug.Print rst.Fields("Kontaktperson").Value
    Debug.Print rst.Fields("Straße").Value
    Debug.Print rst.Fields("Ort").Value
    Debug.Print rst.Fields("Land").Value & vbLf
    rst.FindNext "Land = 'Deutschland'"
  Loop
  rst.Close
  Set rst = Nothing
  Set db = Nothing
  Exit Sub

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

Listing 241: 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 wird. In diesem Fall meldet die Eigenschaft `NoMatch` den Wert `True`. Innerhalb der Schleife erfragen 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.

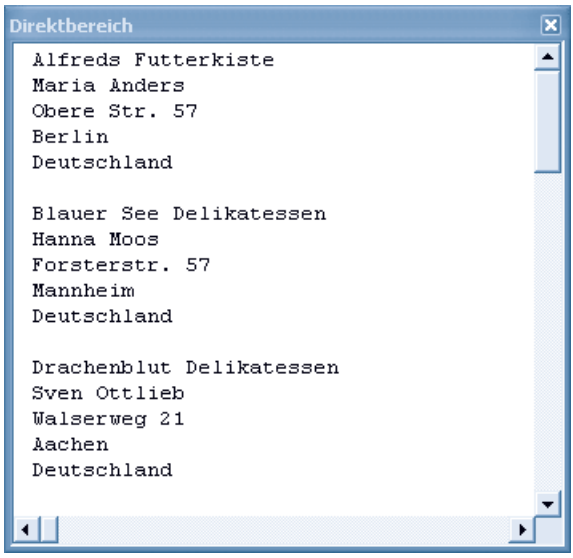


Abbildung 143: Alle Kunden aus Deutschland wurden gefunden.

Mehrere Bedingungen bei der Suche berücksichtigen

Wenn Sie bei einer Suche mehrere Bedingungen gleichzeitig definieren möchten, dann arbeiten Sie mit den Operatoren `AND` oder `OR`.

Im Beispiel aus *Listing 242* werden alle Kunden aus DEUTSCHLAND in der Tabelle KUNDEN gesucht, die gleichzeitig in STUTTGART ansässig sind.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub MehrereDatensätzeFindenDAO2()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Kunden", dbOpenDynaset)

  rst.FindFirst "Land =  'Deutschland' AND Ort = 'Stuttgart'"
```

Listing 242: Mehrere Suchkriterien berücksichtigen (DAO)

```
  Do Until rst.NoMatch = True
    Debug.Print rst.Fields("Firma").Value
    Debug.Print rst.Fields("Kontaktperson").Value
    Debug.Print rst.Fields("Straße").Value
    Debug.Print rst.Fields("Ort").Value
    Debug.Print rst.Fields("Land").Value & vbLf
    rst.FindNext "Land =  'Deutschland' AND Ort = 'Stuttgart'"
  Loop
  rst.Close
  Set rst = Nothing
  Set db = Nothing
  Exit Sub

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

Listing 242: Mehrere Suchkriterien berücksichtigen (DAO) (Forts.)

Über die Methode `FindFirst` können Sie den ersten Datensatz finden, der dem angegebenen Kriterium entspricht. Danach setzen Sie eine `Do Until`-Schleife ein, mit der Sie die Suche so lange wiederholen, bis kein Datensatz mehr gefunden wird, der dem Suchkriterium aus der Methode `FindNext` entspricht. In diesem Fall erhält die Eigenschaft `NoMatch` den Wert `True`.

199 Datensätze verändern (ADO)

Mithilfe der Methode `Update` können Sie eine Änderung an einem Datensatz speichern.

Im folgenden Beispiel aus *Listing 243* werden in der Tabelle GRUPPEN die Gehälter um 2,5 % erhöht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub GehälterAnpassungVornehmen()
  Dim rst As ADODB.Recordset

  Set rst = New ADODB.Recordset
  On Error GoTo Fehler
  rst.Open "Gruppen", CurrentProject.Connection, _
    adOpenKeyset, adLockOptimistic
```

Listing 243: Gehaltsanpassung per Prozedur vornehmen

```
  Do Until rst.EOF
    rst("Gehalt") = rst("Gehalt") * 1.025
    rst.Update
    rst.MoveNext
  Loop
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 243: Gehaltsanpassung per Prozedur 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 in der Tabelle zu aktualisieren. 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, können Sie mithilfe der Methode `Execute` erreichen. Dabei wird dieser Methode eine SQL-Anweisung übergeben. Sehen Sie sich dazu die Prozedur aus *Listing 244* an.

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

Listing 244: Gehaltsanpassung in einer anderen Datenbank vornehmen (ADO)

```
        "C:\Eigene Dateien\Nordwind.mdb"
    .Open
  End With

  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 244: Gehaltsanpassung in einer anderen Datenbank vornehmen (ADO) (Forts.)

Formulieren Sie in der Variablen `strSql` Ihre SQL-Anweisung. Nach dem Öffnen der Datenbank übergeben Sie der Methode `Execute` die Variable `strSql` als Parameter. Die Konstante `RecordsAffected` enthält nach dem Aufruf die Gesamtanzahl der geänderten Sätze.

Im Gegensatz zum vorigen Beispiel, wird hier übrigens über ein `ADODB.Connection`-Objekt eine externe Datenbank geöffnet. Als Provider für die Datenbankverbindung wird der String `"Microsoft.Jet.OLEDB.4.0"` angegeben. Denken Sie daran, dass Sie, sofern Sie mit Access 2007 eine Datei im ACCDB-Format öffnen möchten, Microsoft Office Access Database Engine 2007 verwenden und demnach als Provider den String `"Microsoft.ACE.OLEDB.12.0"` angeben müssen.

200 Datensätze verändern (DAO)

Sollen Datensätze über DAO verändert werden, dann sehen Sie sich das Beispiel aus *Listing 245* an. Bei diesem Beispiel werden alle Artikel der Tabelle ARTIKEL3 im Feld EINZELPREIS um 10% erhöht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeÄndernDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
```

Listing 245: Datenfelder verändern mit DAO

```
  Set rst = db.OpenRecordset("Artikel3", dbOpenDynaset)
  With rst
    .MoveFirst
    Do Until rst.EOF
      rst.Edit
      rst.Fields("Einzelpreis").Value = rst.Fields("Einzelpreis").Value * 1.1
      rst.Update
      .MoveNext
    Loop
    .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 245: Datenfelder verändern mit DAO (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist. Danach rufen Sie die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Rufen Sie die Methode `MoveFirst` auf, um den ersten Satz in der Tabelle zu aktivieren.

In einer `Do Until`-Schleife durchlaufen Sie so lange alle Datensätze der Tabelle, bis die `EOF`-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife rufen Sie die Methode `Edit` auf, um den aktuellen Datensatz zur Änderung zu öffnen. 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 die Änderung im Datensatz zu speichern. Über die Methode `MoveNext` gehen Sie nach jedem Schleifendurchlauf jeweils zum nächsten Datensatz.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

201 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. Bei dieser Angabe müssen Sie den ISO-Code angeben, den Sie in *Tabelle 95* sehen können.

Währung	ISO-Code	Berechnungsgenauigkeit	Anzeige
Belgische Franc	BEF	0	0
Luxemburgische Franc	LUF	0	0
Deutsche Mark	DEM	2	2
Spanische Peseten	ESP	0	0
Französische Franc	FRF	2	2
Irische 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 95: Die Tabelle mit den ISO-Codes (Quelle: Online-Hilfe)

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 6 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
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub EuroUmrechnen()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset

  rst.Open "BestellungenAlt", conn, adOpenKeyset, adLockOptimistic

  Do While Not rst.EOF
    rst![Einzelpreis] = EuroConvert(rst![Einzelpreis], "DEM", "EUR")
    rst.MoveNext
  Loop

  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 246: Währungsumrechnungen durchführen (ADO)

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 dieser Funktion dabei das Datenfeld, das umgerechnet werden soll, sowie die momentan gewünschte Währung.

202 Währungsumrechnungen vornehmen (DAO)

Die Funktion `EuroConvert` kann auch mit DAO eingesetzt werden.

In der folgenden Prozedur aus *Listing 247* werden alle Einzelpreise der Tabelle ARTIKEL1 von DM auf Euro umgestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub EuroUmrechnenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Artikel1", dbOpenTable, dbOpenDynaset)

  With rst
    .MoveFirst
    Do While Not .EOF
      .Edit
      .Fields("Einzelpreis").Value = _
        EuroConvert(.Fields("Einzelpreis").Value, "DEM", "EUR")
      .Update
      .MoveNext
    Loop
    .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 247: Währungsumstellung vornehmen (DAO)

Der Änderungsprozess geht so vor sich, dass zuerst der entsprechende Datensatz über die Methode `Edit` in den Editiermodus versetzt wird. Danach wenden Sie die Funktion `EuroConvert` an, um die Umrechnung im Feld EINZELPREIS durchzuführen. Als Nächstes speichern Sie die Änderung, indem Sie die Methode `Update` aufrufen. In einer `Do until`-Schleife arbeiten Sie so alle Datensätze der Tabelle ab, bis die Eigenschaft `EOF` erfüllt ist.

203 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 248* werden in der Tabelle ARTIKEL alle Artikel im Direktbereich ausgegeben, deren Lagerbestand höher als 20 und deren Einzelpreis größer oder gleich 30 ist.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub DatensätzeFiltern()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  With rst
    .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 rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 248: 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.

Abbildung 144: Alle Artikel, die dem Filterkriterium entsprechen

Filtern nach bestimmten Kunden

In der Prozedur aus *Listing 249* werden aus der Tabelle KUNDEN alle Kunden gefiltert, die aus den USA oder Kanada kommen. Beachten Sie dabei die Schreibweise des Filterkriteriums.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeFiltern2()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  With rst
    .Open "Kunden", conn, adOpenKeyset, _
      adLockOptimistic
    .Filter = "Land = 'USA' OR Land = 'Kanada'"

    Do While Not .EOF
      Debug.Print .Fields("Kontaktperson").Value
```

Listing 249: Das Filtern von alphanumerischen Feldern (ADO)

```
      Debug.Print .Fields("Ort").Value & vbLf
      .MoveNext
    Loop

    .Close
  End With
  conn.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 249: Das Filtern von alphanumerischen Feldern (ADO) (Forts.)

Bei alphanumerischen Feldern müssen Sie darauf achten, dass Sie die Filterkriterien in Apostrophen setzen.

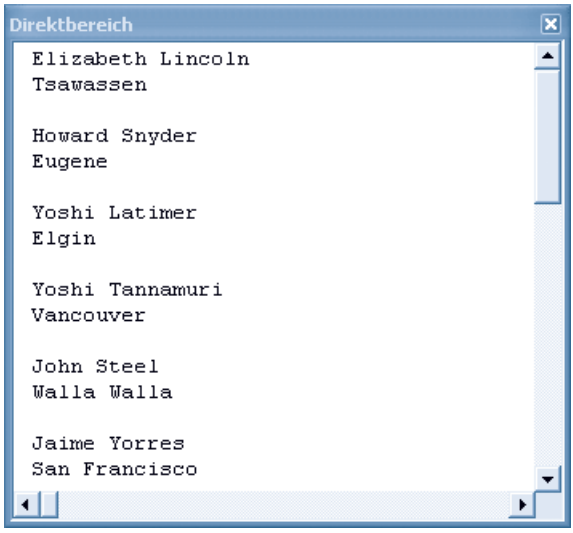


Abbildung 145: Kunden aus den USA oder Kanada

204 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 250* 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
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeFilternDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("SELECT * " & _
    "FROM Artikel WHERE Einzelpreis > 30 AND Lagerbestand > 10", _
      dbOpenSnapshot)

  With rst
    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 db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 250: Datensätze filtern (DAO)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Als Nächstes rufen Sie die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Dabei übergeben Sie beim Öffnen der Tabelle die SQL-Anweisung als String. Als Ergebnis stehen nun alle Datensätze der Tabelle im `DAO.Recordset`-Objekt `rst` und können dort abgegriffen werden.

In einer `Do While`-Schleife durchlaufen Sie so lange alle Datensätze des Recordsets, bis die `EOF`-Eigenschaft (End of File) erfüllt ist. Innerhalb der Schleife greifen Sie über die Auflistung `Fields` auf die einzelnen Felder zu. Über die Methode `MoveNext` rufen Sie nach jedem Schleifendurchlauf den nächsten Datensatz des `DAO.Recordset`-Objekts auf.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

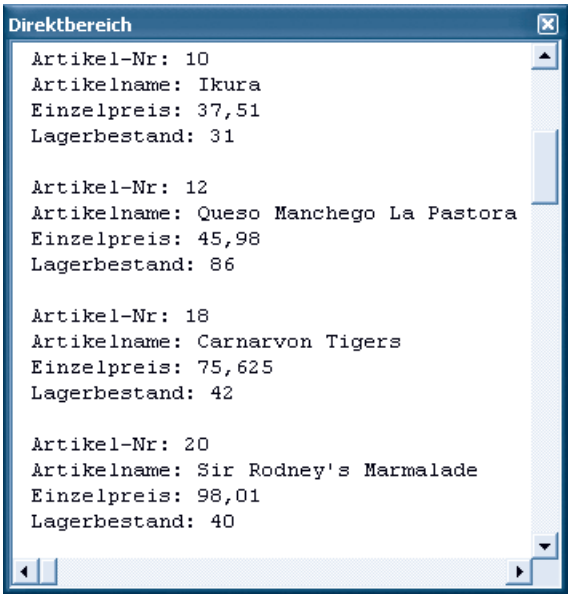


Abbildung 146: Die Daten wurden gefiltert und ausgegeben.

205 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 251* wird die Anzahl der Datensätze der Tabelle KUNDEN ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeZählen()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
```

Listing 251: Datensätze zählen – Variante 1 (ADO)

```
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset

  With rst
    .Open "Kunden", conn, adOpenKeyset, adLockOptimistic
    MsgBox "In der Tabelle befinden sich " & _
      rst.RecordCount & " Datensätze"
    .Close
  End With

  conn.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 251: Datensätze zählen – Variante 1 (ADO) (Forts.)

Zählen über die Funktion DCount

Eine etwas kürzere Variante können Sie über die Funktion `DCount` aus *Listing 252* erreichen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeZählenVar2()
  MsgBox "Es befinden sich : " & _
    DCount("Firma", "Kunden") & _
    " Datensätze in der Tabelle!", vbInformation
End Sub
```

Listing 252: 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 Prozeduren 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 möch-

ten die Anzahl der Kunden bestimmen, die aus Großbritannien oder Frankreich kommen. Sehen Sie sich die Lösung dieser Aufgabe in *Listing 253* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeFilternUndZählen()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset

  With rst
    .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 " & _
      rst.RecordCount & " Datensätze"
    .Close
  End With

  conn.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 253: Datensätze filtern und zählen (ADO)

Nachdem Sie das Filterkriterium eingestellt haben, durchlaufen Sie in einer `Do While`-Schleife alle gefundenen Datensätze und geben diese 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`.

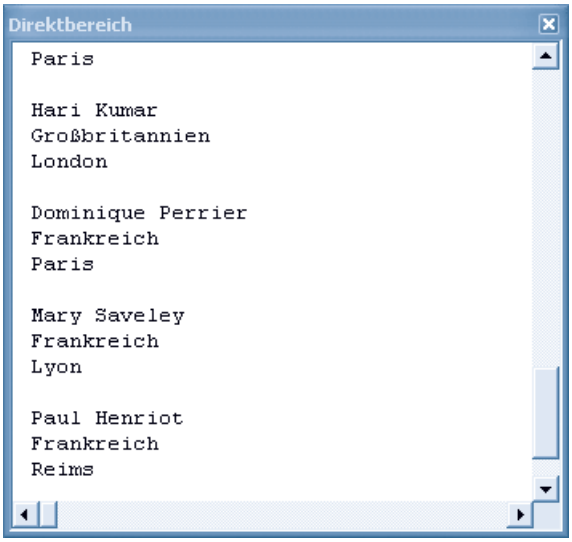


Abbildung 147: Kunden aus Großbritannien und Frankreich

206 Datensätze zählen (DAO)

Auch bei DAO erfolgt die Ermittlung der Anzahl der Datensätze über die Funktion `RecordCount`.

Einfache Zählung vornehmen

Im folgenden Beispiel aus *Listing 254* werden die Datensätze der Tabelle ARTIKEL gezählt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensätzeZählenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Artikel", dbOpenTable, dbReadOnly)

  With rst
    MsgBox "In der Tabelle befinden sich " & _
```

Listing 254: Datensätze zählen (DAO)

```
      .RecordCount & " Datensätze"
    .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 254: Datensätze zählen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Dann rufen Sie die Methode `OpenRecordset` auf, um die Tabelle zu öffnen. Als Nächstes können Sie über die Eigenschaft `RecordCount` die Anzahl der Datensätze erfragen. Am Ende der Prozedur schließen Sie die Tabelle und heben die Objektverweise wieder auf.

Zählung nach Filterung durchführen

Beim folgenden Beispiel aus *Listing 255* werden in der Tabelle KUNDEN erst alle Kunden aus den USA und Kanada herausgefiltert und danach im Direktfenster ausgegeben und gezählt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensätzeFilternUndZählenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler

  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("SELECT * " & _
    "FROM Kunden WHERE Land = 'Kanada' OR Land= 'USA'", dbOpenSnapshot)

  With rst
    Do While Not .EOF
      Debug.Print .Fields("Kontaktperson").Value
      Debug.Print .Fields("Land").Value
```

Listing 255: Datensätze filtern und zählen (DAO)

```
      Debug.Print .Fields("Ort").Value & vbLf
      .MoveNext
    Loop
      MsgBox "In der Tabelle befinden sich " & .RecordCount & " Datensätze"
      .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 255: Datensätze filtern und zählen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Die `CurrentDb`-Methode gibt eine Objektvariable des Typs `Database` zurück, die der Datenbank entspricht, die momentan im Microsoft Access-Fenster geöffnet ist.

Übergeben Sie der Methode `OpenRecordset` beim Öffnen der Tabelle eine SQL-Anweisung in Form eines Strings, über die Sie die Tabelle KUNDEN öffnen und gleichzeitig eine Filterung spezifizieren, über die nur Datensätze ausgewählt werden, bei denen die Kunden aus den USA oder Kanada stammen.

Anschließend können Sie über die Eigenschaft `RecordCount` die Anzahl der Datensätze in der Tabelle ermitteln.

207 Datensätze summieren

Möchten Sie Felder einer Tabelle über mehrere Datensätze hinweg summieren, dann steht Ihnen für diese Aufgabe die Funktion `Dsum` zur Verfügung. Diese Funktion kann sowohl für ADO als auch für DAO eingesetzt werden.

Im Beispiel aus *Listing 256* wird die Tabelle BESTELLDETAILS ausgewertet. Dabei wird der Gesamtwert aller Bestellungen ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================
```

Listing 256: Die Gesamtsumme aller Bestellungen ermitteln

```
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 256: Die Gesamtsumme aller Bestellungen ermitteln (Forts.)

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 Datenfeldes, dessen Summe Sie berechnen möchten. Im *Anhang* können Sie eine Auflistung weiterer D-Funktionen einsehen.

208 Datensätze clonen (ADO)

Die Methode `Clone` erstellt eine Kopie eines `Recordset`-Objekts, die auf dieselben Datensätze des `Recordset`-Objekts verweist. Beachten Sie, dass hierbei lediglich das `Recordset`-Objekt dupliziert wird, nicht jedoch die Datensätze, auf die sich das `Recordset`-Objekt bezieht. Sie können dies nutzen, um beispielsweise im Programm gleichzeitig unterschiedliche Datensätze derselben Tabelle oder Abfrage zu referenzieren. In diesem Beispiel soll Ihnen gezeigt werden, wie Sie in der Lage sind, auch Datensätze zu duplizieren. Die Syntax der Methode `Clone` 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 96: Die Argumente der Methode Clone

Im folgenden Beispiel aus *Listing 257* wird ein bestimmter Datensatz in der Tabelle ARTIKEL gesucht, danach geklont und am Ende der Tabelle unter Vergabe einer neuen Artikel-Nr. eingefügt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================
```

Listing 257: Datensatz clonen und am Ende der Tabelle anhängen (ADO)

```
Sub DatensatzClonen()
  Dim rst1 As New ADODB.Recordset
  Dim rst2 As ADODB.Recordset
  Dim lngNr As Long
  Dim intZ As Integer

  rst1.Open "SELECT * FROM Artikel WHERE Artikelname='Outback Lager'", _
    CurrentProject.Connection, adOpenKeyset, adLockOptimistic

  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
End Sub
```

Listing 257: Datensatz clonen und am Ende der Tabelle anhängen (ADO) (Forts.)

Deklarieren Sie zwei Objektvariable vom Typ `ADODB.Recordset`. Erzeugen Sie für eine der beiden Objektvariablen das entsprechende Objekt mit dem Schlüsselwort `New`. Dies tun Sie am besten, wie im Beispiel dargestellt, schon in der Zeile der Deklaration der Variablen. Dies spart unnötige Codezeilen. Führen Sie beim ersten Objekt eine SQL-Abfrage aus, die aus der Tabelle ARTIKEL alle Datensätze extrahiert, bei denen das Feld ARTIKELNAME den Wert »OUTBACK LAGER« enthält. Rufen Sie hierzu die Methode `Open` auf und übergeben Sie dieser die SQL-Abfrage in Form eines Strings, wie im Beispiel dargestellt. Der gefundene Satz steht danach in der Objektvariablen `rst1` für eine weitere Verarbeitung bereit.

Über den Aufruf der Methode `Clone` wird über die Objektvariable `rst2` eine zweite Referenz auf dieselben Datensätze erstellt, auf die auch `rst1` verweist. Mit dem Aufruf der Methode `AddNew` wird über `rst2` ein neuer Datensatz am Ende der Tabelle angefügt. Im Anschluss daran werden die Feldinhalte des aktuellen Datensatzes von `rst1` auf den aktuellen, neuen Datensatz von `rst2` übertragen. Verdeutlichen Sie sich, dass sich die beiden Recordsets `rst1` und `rst2` auf exakt dieselbe Tabelle beziehen.

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 in der Tabelle gespeichert wird.

Abbildung 148: Der geclonte Datensatz wurde am Ende der Tabelle eingefügt.

209 Datensätze clonen (DAO)

Das Beispiel aus *Listing 257* wird in der Prozedur aus *Listing 258* über den Einsatz von DAO gelöst.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensatzClonenDAO()
  Dim db 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 db = Application.CurrentDb
  Set rst1 = db.OpenRecordset _
    ("SELECT * FROM Artikel WHERE Artikelname='Outback Lager'", dbOpenDynaset)

  Set rst2 = rst1.Clone

  With rst2
    .AddNew
    'Übertragen der Felder
    For intZ = 1 To .Fields.Count - 1
```

Listing 258: Datensatz clonen und am Ende der Tabelle anhängen (DAO)

```
      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 258: Datensatz clonen und am Ende der Tabelle anhängen (DAO) (Forts.)

210 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 zurückkehren. Dabei speichern Sie diese Position in einer Variablen vom Typ `Variant`. Wie das genau funktioniert, sehen Sie in *Listing 259*. 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
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub LesezeichenSetzen()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim varLZ1 As Variant
  Dim varLZ2 As Variant

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
```

Listing 259: Lesezeichen setzen

```
  With rst
    .Open Source:="SELECT Nachname,Vorname FROM Personal " & _
      "ORDER BY Nachname", _
        ActiveConnection:=CurrentProject.Connection, _
        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)
    rst.Bookmark = varLZ1
    Debug.Print "Wieder zurück zu:" & varLZ1
    Debug.Print .Fields("Nachname") & Chr(13)
    .Close
  End With

  conn.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 259: 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 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 `rst.Bookmark = varLZ1` springen Sie direkt wieder auf Ihr vorher gesetztes Lesezeichen und geben die Position und den Nachnamen im Direktfenster aus.

Abbildung 149: Lesezeichen setzen und aktivieren

211 Datensätze löschen (ADO)

Über die Methode `Delete` können Sie Datensätze aus einer Tabelle löschen. Die Syntax dieser Methode lautet:

```
recordset.Delete AffectRecords
```

Argument	Beschreibung
`recordset`	Gibt ein `Recordset`-Objekt zurück
`AffectRecords`	Optional. Hier können die zu löschenden Sätze gezählt werden.

Tabelle 97: Die Argumente der Methode Delete

Im folgenden Beispiel aus *Listing 260* werden alle Datensätze aus der Tabelle ARTIKEL2 entfernt, die einen Lagerbestand von 0 aufweisen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensätzeLöschen()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset

  rst.Open _
    "SELECT * FROM Artikel2 WHERE Lagerbestand = 0", _
    conn, adOpenKeyset, adLockOptimistic
```

Listing 260: Datensätze löschen

```
  Do Until rst.EOF
    rst.Delete
    rst.MoveNext
  Loop
  rst.Close
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 260: 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.

212 Datensätze löschen (DAO)

Das Löschen von Datensätzen erfolgt bei DAO ebenfalls über eine Methode `Delete`.

Bestimmte Kunden löschen

Im Beispiel aus *Listing 261* werden aus der Tabelle KUNDEN2 alle Kunden aus Deutschland gelöscht.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensätzeLöschenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Kunden2", dbOpenDynaset)

  With rst
    .FindFirst "Land =  'Deutschland'"
    Do Until .NoMatch = True
      .Delete
      .FindNext "Land = 'Deutschland'"
```

Listing 261: Datensätze löschen (DAO)

```
    Loop
  .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 261: Datensätze löschen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Rufen Sie dann die Methode `OpenRecordset` auf, um die Tabelle KUNDEN2 zu öffnen. 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 so lange alle Datensätze der Tabelle, 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 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 der Prozedur die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

Lager bereinigen

Im folgenden Beispiel aus *Listing 262* werden in der Tabelle ARTIKEL3 alle Datensätze gelöscht, die einen Lagerbestand von 0 haben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub DatensätzeLöschenDAO2()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset
```

Listing 262: Datensätze mit Nullbestand löschen (DAO)

```
  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Artikel3", dbOpenDynaset)

  With rst
    .MoveFirst

    Do Until rst.EOF
      If rst.Fields("Lagerbestand").Value = 0 Then rst.Delete
      .MoveNext
    Loop
    .Close
  End With

  Set db = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 262: Datensätze mit Nullbestand löschen (DAO) (Forts.)

Deklarieren Sie im ersten Schritt zwei Variable, eine Variable `db` vom Typ `DAO.Database` und eine Variable `rst` vom Typ `DAO.Recordset`. Danach weisen Sie der Objektvariablen `db` über die Anweisung `Set` die aktuelle Datenbank zu. Als Nächstes rufen Sie die Methode `OpenRecordset` auf, um die Tabelle ARTIKEL3 zu öffnen. Wenden Sie die Methode `MoveFirst` an, um den ersten Datensatz in der Tabelle zu aktivieren.

In einer `Do Until`-Schleife durchlaufen Sie so lange alle Datensätze der Tabelle, bis die `EOF`-Eigenschaft (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` rufen Sie nach jedem Schleifendurchlauf jeweils den nächsten Datensatz auf.

Schließen Sie die Tabelle über die Methode `Close` und heben Sie die Objektverweise auf.

213 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 263* wird in der Tabelle PERSONAL ein neuer Datensatz eingefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=====================================================

Sub DatensatzHinzufügenDAO()
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Personal", dbOpenDynaset)

  With rst
    .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 rst = Nothing
  Set db = Nothing
  Exit Sub

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

Listing 263: Neuen Datensatz hinzufügen und füllen (ADO)

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 an das Ende der Tabelle angehängt. 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.

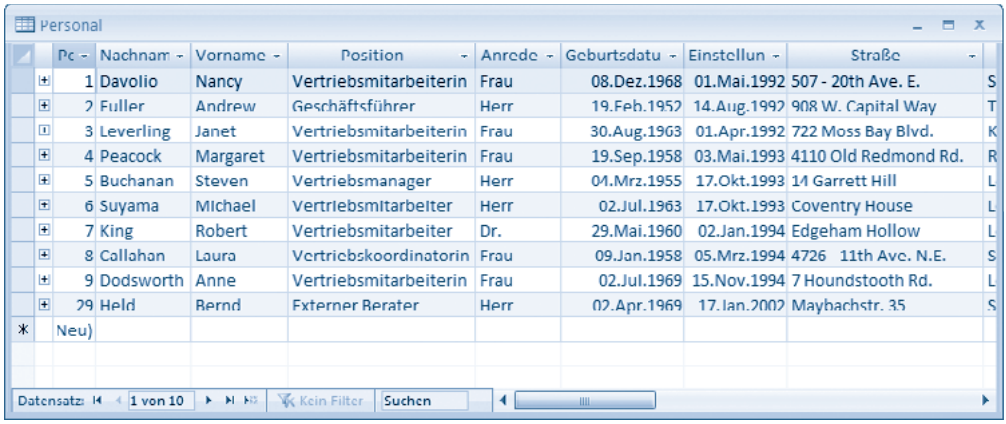


Abbildung 150: Ein Datensatz für den neuen Mitarbeiter wurde am Ende der Tabelle angehängt.

Neue Daten aus Excel beziehen

Beim Beispiel aus *Listing 264* wird die vorherige Aufgabe aus *Listing 263* 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 151* gezeigt aus.

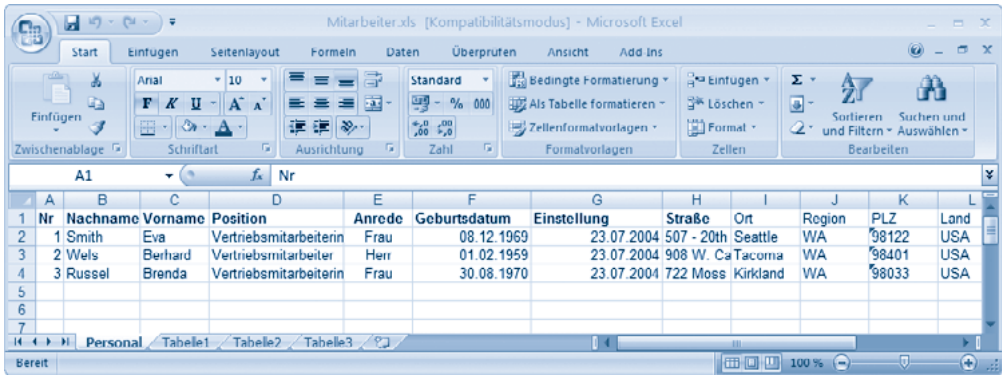


Abbildung 151: Die neuen Mitarbeiter liegen in einer Excel-Tabelle vor.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADO
'=======================================================

Sub DatenAusExcelTabelleHinzufügen()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim objXlApp As Object
  Dim objMappe As Object
  Dim intZ As Integer

  On Error GoTo Fehler
  Set objXlApp = CreateObject("Excel.Application")
  Set objMappe = objXlApp.workbooks.Open(Application.CurrentProject.Path _
    & "\Mitarbeiter.xls")

  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  rst.Open "Personal", conn, adOpenKeyset, _
    adLockOptimistic
  For intZ = 2 To objMappe.sheets("Personal").usedrange.rows.Count
  With rst
    .AddNew
    .Fields("Nachname") = objMappe.sheets(1).cells(intZ, 2).Value
    .Fields("Vorname") = objMappe.sheets(1).cells(intZ, 3).Value
    .Fields("Position") = objMappe.sheets(1).cells(intZ, 4).Value
    .Fields("Anrede") = objMappe.sheets(1).cells(intZ, 5).Value
    .Fields("Geburtsdatum") = objMappe.sheets(1).cells(intZ, 6).Value
    .Fields("Einstellung") = objMappe.sheets(1).cells(intZ, 7).Value
    .Fields("Straße") = objMappe.sheets(1).cells(intZ, 8).Value
    .Fields("Ort") = objMappe.sheets(1).cells(intZ, 9).Value
    .Fields("Region") = objMappe.sheets(1).cells(intZ, 10).Value
    .Fields("PLZ") = objMappe.sheets(1).cells(intZ, 11).Value
    .Fields("Land") = objMappe.sheets(1).cells(intZ, 12).Value
    .Update
  End With
  Next intZ

  rst.Close
  objMappe.Close
  objXlApp.Quit

  Set objMappe = Nothing
  Set objXlApp = Nothing
```

Listing 264: Personaldaten aus einer Excel-Tabelle importieren

```
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

Fehler:
  MsgBox Err.Number & " " & Err.Description
  Set objMappe = Nothing
  Set objXlApp = Nothing
End Sub
```

Listing 264: Personaldaten aus einer Excel-Tabelle importieren (Forts.)

Über die Funktion `CreateObject` erzeugen Sie ein EXCEL-APPLICATION-Objekt. Öffnen Sie die Excel-Arbeitsmappe über die Methode `Open`, arbeiten Sie in einer Schleife Zeile für Zeile der Excel-Tabelle ab und übertragen Sie diese in die Access-Tabelle.

Abbildung 152: Die neuen Daten wurden der Access-Tabelle hinzugefügt.

214 Datensätze hinzufügen (DAO)

Die Neuanlage von Datensätzen funktioniert in DAO ganz ähnlich wie in ADO. Im folgenden Beispiel aus *Listing 265* wird der Tabelle PERSONAL ein neuer Satz hinzugefügt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub DatensatzHinzufügenDAO()
  Dim db As DAO.Database
```

Listing 265: Neuen Datensatz anlegen (DAO)

```
  Dim rst As DAO.Recordset

  On Error GoTo Fehler
  Set db = Application.CurrentDb
  Set rst = db.OpenRecordset("Personal", dbOpenDynaset)

  With rst
    .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 rst = Nothing
  Set db = Nothing
  Exit Sub

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

Listing 265: 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.

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 ein, indem Sie in der Entwicklungsumgebung aus dem Menü Extras den Befehl Verweise auswählen, die Bibliothek aktivieren und mit OK bestätigen.

215 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. Im Falle, dass es sich um eine Tabelle handelt, gibt diese Eigenschaft den Text `Table` zurück.

Im folgenden Beispiel aus *Listing 266* werden die unterschiedlichen Tabellentypen im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabellenAusgeben()
  Dim cat As ADOX.Catalog
  Dim tbi As ADOX.Table

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog
  cat.ActiveConnection = _
    "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source= C:\Eigene Dateien\Nordwind.mdb"
  For Each tbi In cat.Tables
    With tbi
      Debug.Print "Name: " & vbTab & .Name & vbLf & _
        "Typ:" & vbTab & .Type & vbLf
    End With
  Next tbi
  Set tbi = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 266: Alle Tabellentypen ermitteln

Deklarieren Sie im ersten Schritt zwei ADOX-Objektvariable. Im `Catalog`-Objekt sind unter anderem alle Tabellen der Datenbank verzeichnet. Diese Tabellen arbeiten Sie mithilfe einer `For Each Next`-Schleife ab und ermitteln den Namen sowie den Tabellentyp über die Eigenschaften `Name` und `Type`.

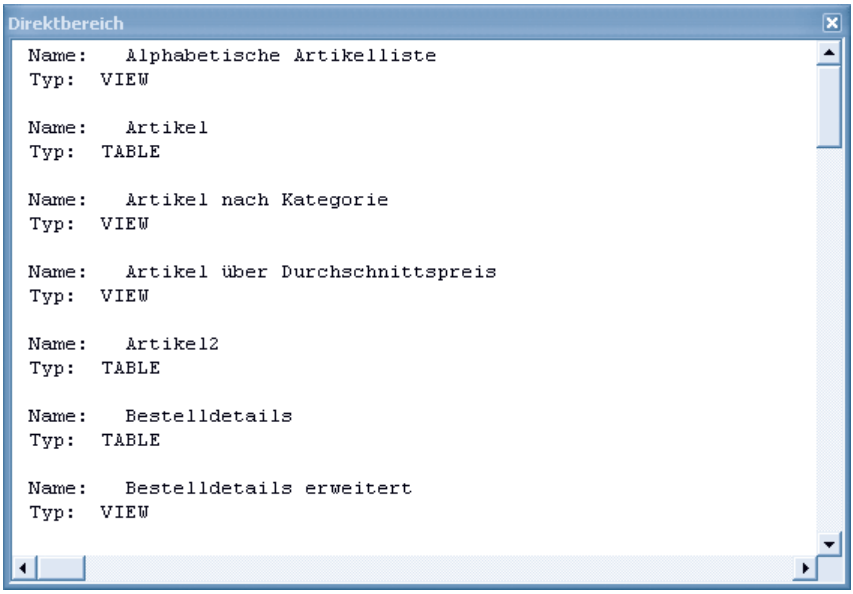


Abbildung 153: Tabelle und Abfragen wurden aufgelistet.

216 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 267* wird das Erstellungsdatum sowie das letzte Änderungsdatum der Tabelle ARTIKEL aus der Datenbank NORDWIND.MDB ermittelt und am Bildschirm angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabellenInfosAusgeben()
  Dim cat As ADOX.Catalog
  Dim tbi As ADOX.Table

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog
  cat.ActiveConnection = _
  "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source= C:\Eigene Dateien\Nordwind.mdb"
  Set tbi = cat.Tables("Artikel")
```

Listing 267: Tabelleninfos ausgeben

```
  With tbi
    MsgBox "Name: " & .Name & vbLf & _
      "Erstellungsdatum: " & .DateCreated & vbLf & _
      "Änderungsdatum: " & .DateModified
  End With
  Set tbi = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 267: Tabelleninfos ausgeben (Forts.)

Definieren Sie im ersten Schritt eine Variable vom Typ `ADOX.Catalog` sowie eine Variable vom Typ `ADOX.Table`. Über die `ADOX.Catalog`-Variable können Sie unter anderem auf alle Tabellen Ihrer Datenbank zugreifen. Danach geben Sie über die Eigenschaft `ActiveConnection` die Datenquelle bekannt.

Weisen Sie der zuvor deklarierten `ADOX.Table`-Variable über die `ADOX.Catalog`-Methode `Tables` die Tabelle ARTIKEL zu. Nun können Sie über die Methoden `Name`, `DateCreated` und `DateModified` die gewünschte Tabelleninformation abfragen.

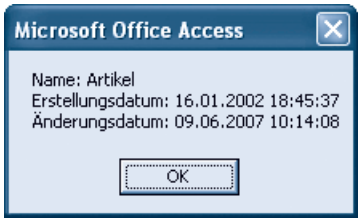


Abbildung 154: Zugriff auf Tabelleneigenschaften über ADOX

217 Tabellenstrukturen auslesen

Um die Struktur der einzelnen Felder einer Tabelle auszulesen, können Sie das Objekt `ADOX.Column` nutzen.

Im Beispiel aus *Listing 268* werden die Felddatentypen der Tabelle ARTIKEL ausgelesen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=======================================================
```

Listing 268: Felddatentypen ermitteln

```
Sub TabellenStrukturenAuslesen()
  Dim cat As New ADOX.Catalog
  Dim tbl As ADOX.Table
  Dim strText As String
  Dim intZ As Integer

  On Error GoTo Fehler
  cat.ActiveConnection = CurrentProject.Connection
  Set tbl = cat.Tables("Artikel")

  With tbl
    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"
        Case Else
          strText = "noch nicht erfasst"
      End Select

      Debug.Print "Felddatentyp: " & _
        .Columns(intZ).Type & " ---> " & strText & vbLf
    Next intZ
  End With
  Set cat = Nothing
  Exit Sub

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

Listing 268: Felddatentypen ermitteln (Forts.)

Mithilfe einer `Select Case`-Anweisung können Sie den numerischen Wert, den Ihnen die Eigenschaft `Type` meldet, in einen Text umwandeln.

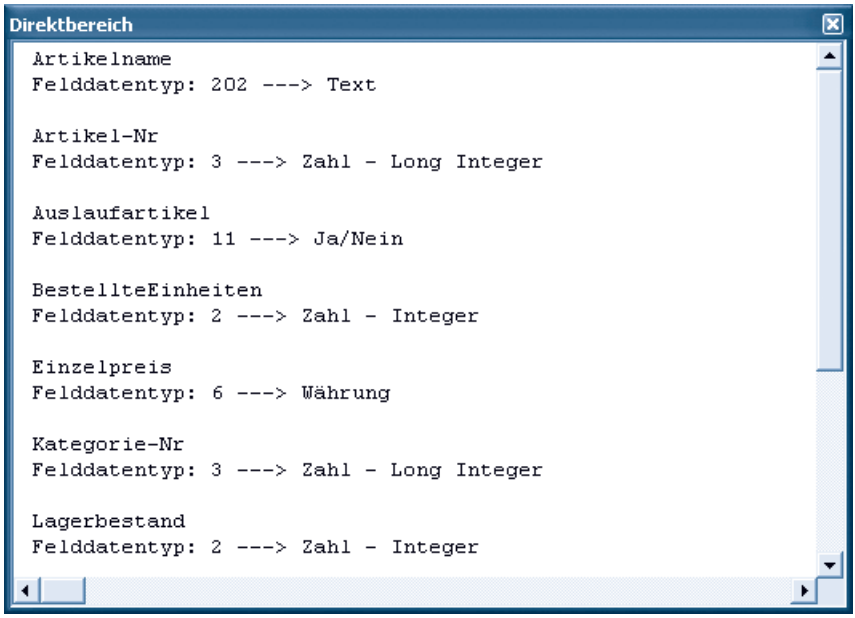


Abbildung 155: Die Felddatentypen wurden ausgelesen.

218 Tabelleneigenschaften abfragen

Um die Struktur der einzelnen Felder einer Tabelle auszulesen, bietet Ihnen das Objekt `ADOX.Column` einige Eigenschaften.

Im folgenden Beispiel aus *Listing 269* lesen Sie alle Datenfelddefinitionen der Tabelle ARTIKEL aus der Datenbank TABELLEN.MDB aus.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=======================================================

Sub FeldInfosAuslesen()
  Dim cat As New ADOX.Catalog
  Dim tbl As ADOX.Table
  Dim intZ As Integer

  On Error GoTo Fehler
  cat.ActiveConnection = CurrentProject.Connection
  Set tbl = cat.Tables("Artikel")
  With tbl
    For intZ = 0 To .Columns.Count - 1
```

Listing 269: Tabelleneigenschaften auslesen

```
      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 cat = Nothing
  Exit Sub

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

Listing 269: Tabelleneigenschaften auslesen (Forts.)

Über die Eigenschaft `Name` können Sie den Namen des Feldes ermitteln. Den Datentyp der Datenfeldes können Sie über die Eigenschaft `Type` herausfinden. Mit der Eigenschaft `Attributes` können Sie bestimmen, ob die Spalte eine feste Länge hat oder 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` deklarieren.

Mit der Eigenschaft `ParentCatalog` können Sie das `Catalog`-Objekt angeben, 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 Datenfeldes – können Sie mit der `Properties`-Auflistung zugreifen.

Geben Sie im Auflistungsobjekt `Tables` an, welche Tabelle Sie auslesen möchten. Diese Bekanntmachung führen Sie mit der Anweisung `Set` durch. Danach nutzen Sie das Auflistungsobjekt `Columns`, 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, da das Objekt `Columns` mit dem Index null beginnt und der höchste Index somit um eins geringer ist, als die von der Eigenschaft `Count` gemeldete Anzahl der Elemente. Innerhalb der Schleife fragen Sie die einzelnen Felder über die vorher beschriebenen Eigenschaften ab und geben diese im Direktfenster aus. Heben Sie am Ende der Prozedur die Objektverweise über das Schlüsselwort `Nothing` wieder auf, um belegten Arbeitsspeicher freizugeben.

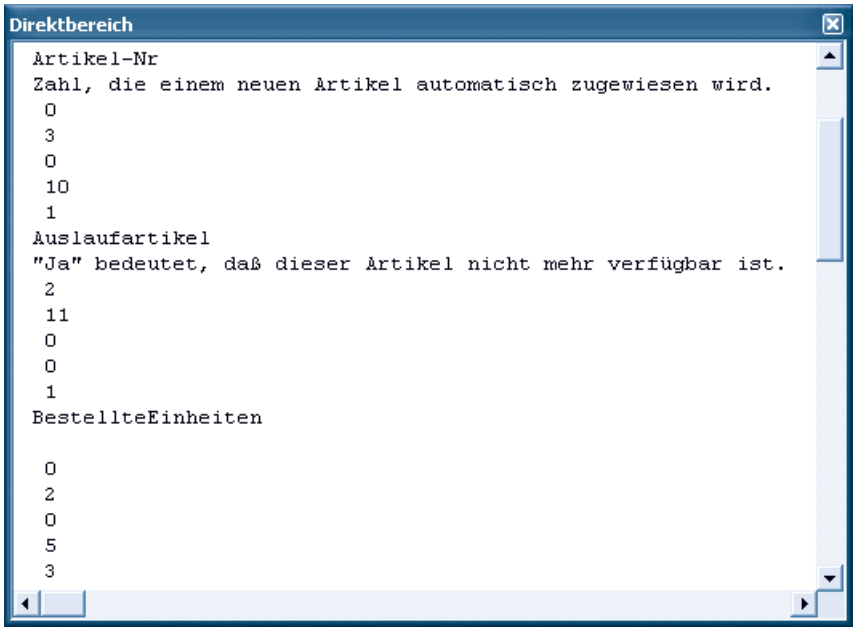


Abbildung 156: Die Tabelleneigenschaften wurden dokumentiert.

219 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.

Mit dem 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 `Variant`-Typ
`adInteger`	`Integer`-Typ
`adBoolean`	Boolescher `Variant`-Typ
`adChar`	Zeichenfolge mit fester Länge

Tabelle 98: Die wichtigsten Datentypen der Methode Append

Konstante	Beschreibung
adVarChar	Zeichenfolge mit variabler Länge
adVarWChar	Zeichenfolge mit Wide-variabler Länge
adLongVarChar	Zeichenfolge mit Long-variabler Länge

Tabelle 98: Die wichtigsten Datentypen der Methode Append (Forts.)

Neue Tabelle in aktueller Datenbank anlegen

Im folgenden Beispiel aus *Listing 270* wird eine neue Tabelle mit dem Namen WAREN erstellt, und es werden dazu einige Felder definiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabelleErstellen()
  Dim cat As ADOX.Catalog
  Dim tbl As ADOX.Table

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog
  cat.ActiveConnection = CurrentProject.Connection
  Set tbl = New ADOX.Table
  With tbl
    .Name = "Waren"
    Set .ParentCatalog = cat
    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
  cat.Tables.Append tbl
  cat.ActiveConnection = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 270: Neue Tabelle mit Feldern erstellen

Abbildung 157: Die neue Tabelle wurde erstellt.

Eine Alternative zur Methode `Append` bietet die Methode `CreateField`.

Neue Tabelle in neuer Datenbank anlegen

Im Beispiel aus *Listing 271* wird eine neue Datenbank angelegt sowie eine Tabelle, die folgende Felder aufweist: FIRMA, NACHNAME, VORNAME, E-MAIL-ADRESSE und TELEFONNUMMER.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub NeueDBAnlegenUndTabelleErstellen()
  Dim AccessObj As Object
  Dim db As Object
  Dim tdf As Object
  Dim varTabfeld As Variant

  Set AccessObj = _
    CreateObject("Access.Application")
  AccessObj.NewCurrentDatabase _
    "C:\Eigene Dateien\Adressen"
```

Listing 271: Neue Datenbank und neue Tabelle anlegen

```
  Set db = AccessObj.CurrentDb

  Set tdf = db.CreateTableDef("Kontakte")
  Set varTabfeld = tdf.CreateField("Firma", 10, 20)
  tdf.Fields.Append varTabfeld

  Set varTabfeld = tdf.CreateField("Nachname", 10, 30)
  tdf.Fields.Append varTabfeld

  Set varTabfeld = tdf.CreateField("Vorname", 10, 20)
  tdf.Fields.Append varTabfeld

  Set varTabfeld = tdf.CreateField("E-Mail", 10, 30)
  tdf.Fields.Append varTabfeld

  Set varTabfeld = tdf.CreateField("Telefon", 10, 25)
  tdf.Fields.Append varTabfeld
  db.TableDefs.Append tdf

  Set AccessObj = Nothing
End Sub
```

Listing 271: Neue Datenbank und neue Tabelle anlegen (Forts.)

Erstellen Sie im ersten Schritt ein Access-Objekt über den Befehl `CreateObject("Access.Application")`. Sie haben dabei prinzipiell auch die Möglichkeit eine Access-Version anzugeben. Hierbei würden Sie dem Befehl `CreateObject` z. B. den String `Access.Application.11` übergeben. Dies würde bedeuten, dass die Access-2003-Objektbibliothek angesprochen wird. Dies funktioniert nur, wenn die entsprechende Access-Version auf dem PC zur Verfügung steht. Übergeben Sie keine Versionsnummer, wird die Objektbibliothek der aktuellen Office-Version angesprochen.

Mit der Methode `NewCurrentDatabase` erstellen Sie eine neue Access-Datenbank. Dabei müssen Sie den Pfad und den Dateinamen angeben. Die Endung des Dateinamens müssen Sie nicht angeben. Diese wird von Access automatisch richtig vergeben. Bei Access 2007 erhält die erstellte Datei demnach die Endung .ACCB, bei früheren Access-Versionen die Endung .MDB. Im nächsten Schritt initialisieren Sie die Objektvariable `db` über die Anweisung `Set` als Referenz auf die neue Datenbank. Setzen Sie für diesen Zweck die Methode `CurrentDb` ein. Erstellen Sie danach im Objekt `tdf` eine neue Tabelle über die Methode `CreateTableDef`. Geben Sie dabei den Namen der Tabelle an.

Legen Sie dann 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 der Prozedur fügen Sie die neue Tabelle über die Zeile

`db.TableDefs.Append tdf` der Datenbank hinzu. Heben Sie zuletzt die Objektverweise über die Anweisung `Set AccessObj = Nothing` auf.

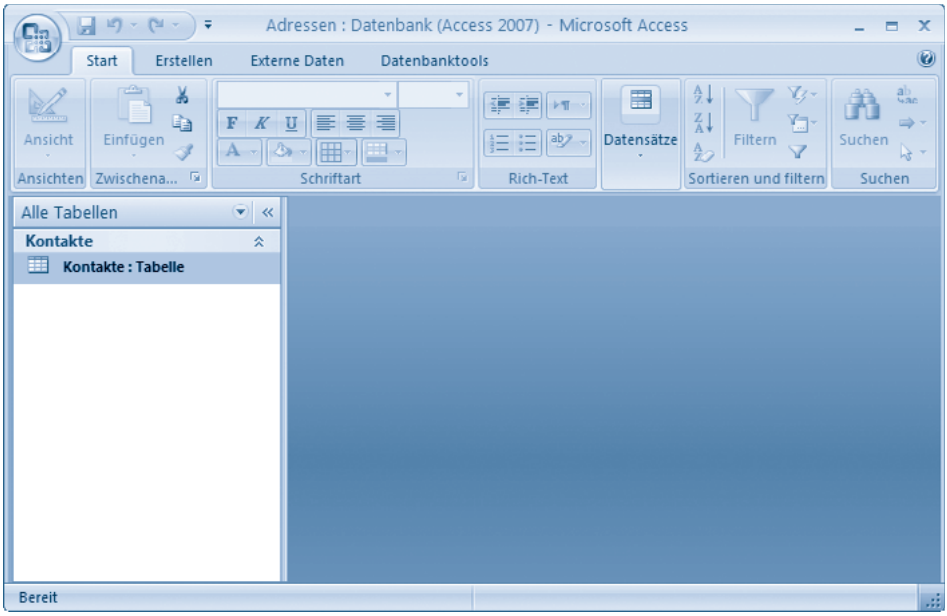


Abbildung 158: Die neue Tabelle wurde erfolgreich in einer neuen Datenbank angelegt.

220 Tabellen verknüpfen

Über die `Properties`-Auflistung können Sie auf Tabelleneigenschaften zugreifen. Unter anderem können Sie diese Auflistung auch nutzen, um externe Tabellen in Ihre Datenbank zu verlinken.

Im folgenden Beispiel aus *Listing 272* wird die externe Tabelle ARTIKEL in die aktuell geöffnete Datenbank unter dem Namen KUNDEN_VERKNÜPFT verknüpft.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=======================================================

Sub TabelleVerknüpfen ()
  Dim cat As New ADOX.Catalog
  Dim tbl As New ADOX.Table

  On Error GoTo Fehler
  cat.ActiveConnection = CurrentProject.Connection
```

Listing 272: Eine Tabelle verknüpfen

```
  tbl.Name = "Kunden_Verknüpft"
  Set tbl.ParentCatalog = cat
  tbl.Properties("Jet OLEDB:Link Datasource") = _
    "C:\Eigene Dateien\Nordwind.mdb"
  tbl.Properties("Jet OLEDB:Remote Table Name") = _
    "Kunden"
  tbl.Properties("Jet OLEDB:Create Link") = True
  cat.Tables.Append tbl

  Set tbl = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 272: Eine Tabelle verknüpfen (Forts.)

Geben Sie im Argument `Link Datasource` den Namen sowie den Pfad der Datenbank an, in dem 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` fügen Sie der Datenbank die Tabellenverknüpfung hinzu.

Abbildung 159: Die Verknüpfung zur externen Tabelle wurde eingefügt.

221 Tabellen löschen

Über den Einsatz der Methode `Delete` können Sie eine Tabelle löschen.

Im Beispiel aus *Listing 273* wird die Tabelle WAREN gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlADOX
'=====================================================

Sub TabelleLöschen()
  Dim cat As ADOX.Catalog
  Dim tbl As ADOX.Table

  Set cat = New ADOX.Catalog
  cat.ActiveConnection = CurrentProject.Connection

  On Error GoTo Fehler
  cat.Tables.Delete "Waren"
  Exit Sub

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

Listing 273: Eine Tabelle entfernen

222 Tabellenfeld anhängen (DAO)

Sie haben jederzeit die Möglichkeit, einer bestehenden Tabelle über eine Prozedur ein weiteres Feld hinzuzufügen. Im folgenden Beispiel aus *Listing 274* wird der Tabelle PERSONAL das Feld ALTER hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub FeldAnhängen()
  Dim db As Database
  Dim tdf As TableDef
  Dim fld As Field

  On Error GoTo Fehler
  Set db = CurrentDb
```

Listing 274: Ein zusätzliches Feld anhängen

```
  Set tdf = db.TableDefs!Personal

  Set fld = tdf.CreateField("Alter")
  fld.Type = dbDouble
  tdf.Fields.Append fld
  Exit Sub

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

Listing 274: Ein zusätzliches Feld anhängen (Forts.)

Mithilfe der Methode `CurrentDb` erhalten Sie eine Referenz auf die aktuelle Datenbank. 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.

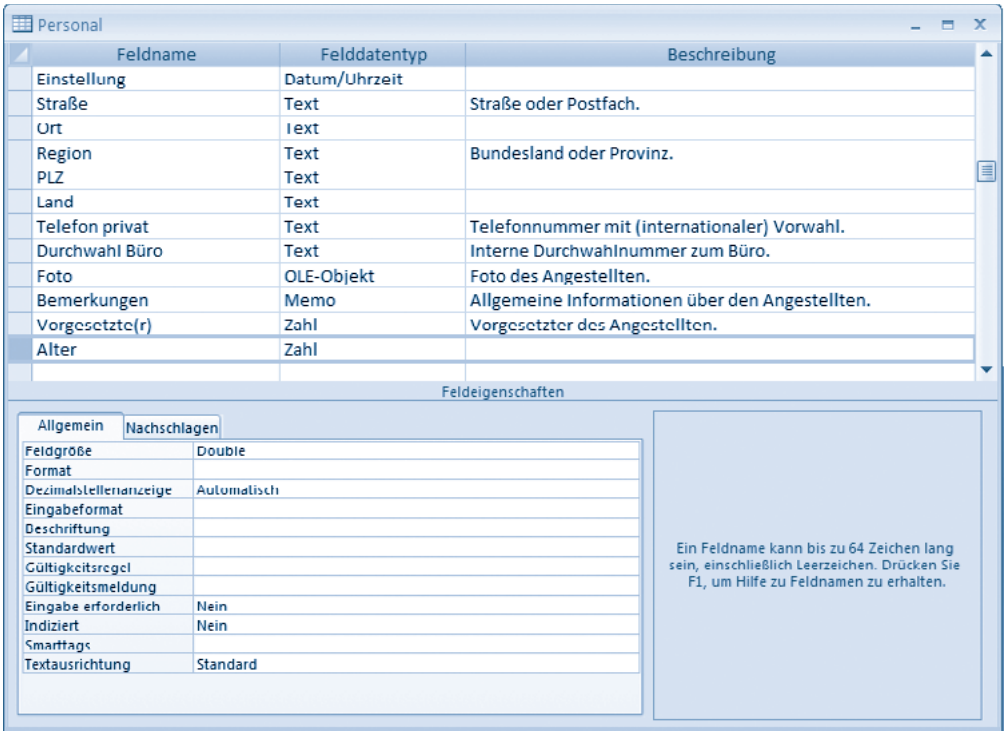


Abbildung 160: Das Feld wurde hinzugefügt.

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 161*). Möchten Sie aber über eine Prozedur 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 die Prozedur aus *Listing 275*.

Abbildung 161: An den Blockpfeilen können verlinkte Tabellen erkannt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub VerlinkteTabellenAnzeigen()
  Dim db As Database
  Dim intZ As Integer
  Dim tdf As TableDef

  On Error GoTo Fehler
```

Listing 275: Ermitteln, welche Tabellen extern verlinkt sind

```
  Set db = CurrentDb
  For intZ = 0 To db.TableDefs.Count - 1
    Set tdf = db.TableDefs(intZ)
    If Len(tdf.Connect) > 0 Then
      Debug.Print tdf.Name
    End If
  Next intZ

  Set db = Nothing
  Exit Sub

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

Listing 275: 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 geben sie einen Wert > 0 zurück. In diesem Fall schreiben Sie diese Tabellen über den Befehl `Debug.Print` in den Direktbereich.

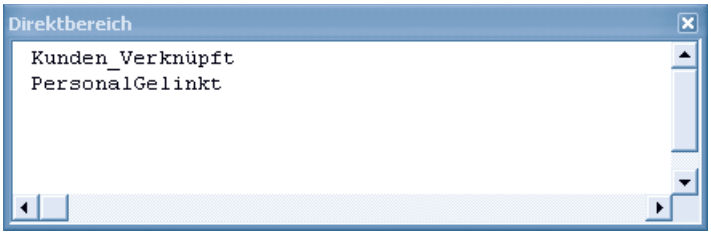


Abbildung 162: Die verlinkten Dateien wurden ausgegeben.

223 Verknüpfte Tabellen aktualisieren

Möchten Sie alle verlinkten Tabellen der aktuellen Datenbank aktualisieren, können Sie diese Aufgabe über die Prozedur aus *Listing 276* lösen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=====================================================

Sub VerlinkteTabellenAktualisieren()
  Dim db As Database
```

Listing 276: Verlinkte Tabellen aktualisieren

```
  Dim intZ As Integer
  Dim tdf As TableDef

  On Error GoTo Fehler
  Set db = CurrentDb

  For intZ = 0 To db.TableDefs.Count - 1
  Set tdf = db.TableDefs(intZ)
  If Len(tdf.Connect) > 0 Then
    tdf.RefreshLink
  End If
  Next intZ
  Set db = Nothing
  Exit Sub

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

Listing 276: Verlinkte Tabellen aktualisieren (Forts.)

Wenden Sie die Methode `RefreshLink` an, um die verknüpften Tabellen zu aktualisieren.

224 Verknüpfungsadresse anpassen

Ändert sich der Pfad bzw. das Laufwerk einer Datenbank, dann können Sie über die Prozedur aus *Listing 277* diese Pfade leicht anpassen. Hier soll das alte Verzeichnis C:\EIGENE DATEIEN in C:\DATEN geändert werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Tabellen.mdb
' Modul      mdlDAO
'=======================================================

Sub PfadVerlinkterTabellenÄndern()
  Dim db As Database
  Dim tdf As TableDef

  Set db = CurrentDb

  On Error GoTo Fehler
  For Each tdf In db.TableDefs
    If Len(tdf.Connect) > 0 Then
      tdf.Connect = ";DATABASE=" & "C:\Daten\Nordwind.mdb"
```

Listing 277: Pfade von verknüpften Tabellen anpassen

```
      tdf.RefreshLink
    End If
  Next tdf
  Set db = Nothing
  Exit Sub

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

Listing 277: Pfade von verknüpften Tabellen anpassen (Forts.)

Mithilfe der Methode `RefreshLink` können Sie den Pfad der verlinkten Tabellen anpassen.

Abfragen programmieren

Im vorherigen Kapitel haben Sie komplexere Auswertungen von Tabellen vorgenommen und die Ergebnisse davon dann im Direktfenster der Entwicklungsumgebung ausgegeben. Wenn Sie Abfragen programmieren oder auch manuell erstellen, dann können Sie die Ergebnisse in Abfragetabellen (engl. Views) speichern. Wie Sie Abfragen per VBA erstellen, können Sie in diesem Kapitel nachschlagen.

Bei der Programmierung von Abfragen stehen 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.

Abfragen mithilfe von DoCmd

Über das Objekt `DoCmd` können Sie unter anderem auch Abfragen durchführen. Die Methode, um eine Abfrage zu starten, lautet: `OpenQuery`.

225 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.

Tabelle 99: Die Argumente der Methode OpenQuery

Argument	Beschreibung
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 99: Die Argumente der Methode OpenQuery (Forts.)

Im folgenden Beispiel aus *Listing 278* wird die Abfrage mit dem Namen ARTIKEL NACH KATEGORIE ausgeführt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 278: Abfrage starten

Die Methode OpenQuery ruft die in der Rubrik ABFRAGEN gespeicherte Abfrage ARTIKEL NACH KATEGORIE auf, die ihrerseits auf die hinterlegten Tabellen ARTIKEL und KATEGORIE zugreift und das Abfrageergebnis in einer Tabelle darstellt.

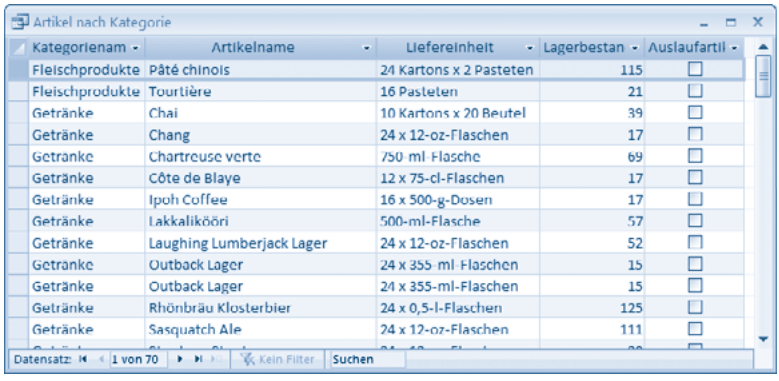


Abbildung 163: Die Abfrage wurde ausgeführt und geöffnet.

226 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 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 100: 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.

227 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
```

Die Anweisung `UPDATE` gibt bekannt, dass eine Tabelle aktualisiert wird. Mit `SET` geben Sie Datenfeldbezeichnung und neuen Inhalt des Datenfeldes bekannt. Über `WHERE` können Sie eine oder mehrere Bedingungen angeben, nach welchen Sätzen in der Tabelle gesucht werden soll.

Im Beispiel aus *Listing 279* 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
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
```

Listing 279: 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 279: Mit UPDATE Bestände initialisieren (Forts.)

Über die Methode `SetWarnings`, die hier auf den Wert `False` gesetzt wird, können Standardmeldungen, wie z. B. die Aktualisierungsmeldung, unterdrückt werden. Übergeben Sie danach der Methode `RunSql` die SQL-Anweisung.

Abbildung 164: Der Bestand wurde für alle Artikel auf 0 gesetzt.

Im nächsten Beispiel aus *Listing 280* wird ein Lieferantenwechsel vorgenommen. Dabei soll der Lieferant NEW ORLEANS CAJUN DELIGHTS (Lieferanten-Nr. = 2) in Zukunft stärker eingesetzt werden und die Lieferanten GRANDMA KELLY'S HOMESTEAD (Lieferanten-Nr. = 3) und TOKYO TRADERS (Lieferanten-Nr. = 4) ablösen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub LieferantenWechsel()
  Dim strSQL As String

  strSQL = "UPDATE Artikel2 " & _
```

Listing 280: Lieferantenwechsel durchführen

```
      "SET Artikel2.Lieferanten_Nr = 2 " & _
      "WHERE Artikel2.Lieferanten_Nr = 3 OR Artikel2.Lieferanten_Nr = 4"

  On Error GoTo Fehler
  DoCmd.SetWarnings False
  DoCmd.RunSQL strSQL
  DoCmd.SetWarnings True
  Exit Sub

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

Listing 280: Lieferantenwechsel durchführen (Forts.)

228 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 die SQL-Anweisung `INSERT INTO` geben Sie bekannt, dass Sie in einer Tabelle einen Satz anfügen möchten. Über die SQL-Anweisung `IN` teilen Sie den Namen der Zieldatenbank mit. Geben Sie danach über die Anweisung `Select` an, welche Datenfelder Sie an die Tabelle 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. Sie können `INSERT INTO` auch folgendermaßen einsetzen:

```
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 über die Anweisung `Select` die Datenfelder der Tabelle 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 Kommata voneinander ab.

Im Beispiel aus *Listing 281* wird der Tabelle PERSONAL ein neuer Mitarbeiter hinzugefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
```

Listing 281: Neuen Datensatz mit INSERT INTO hinzufügen

```
Sub MitarbeiterHinzufügen()
  Dim strSQL As String

  On Error GoTo Fehler
  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 281: Neuen Datensatz mit INSERT INTO hinzufügen (Forts.)

Über die Anweisung `INSERT TO` geben Sie bekannt, dass Sie in die Tabelle PERSONAL einen neuen Datensatz einfügen möchten. Geben Sie danach die Datenfelder der Tabelle 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 Kommata voneinander ab.

Personal

Personal-N	Nachnam	Vorname	Position	Anrede	Geburtsdatu	Einstellun	Straße
4	Peacock	Margaret	Vertriebsmitarbeiterin	Frau	19.Sep.1958	03.Mai.1993	4110 Old Redm
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 Hous
7	King	Robert	Vertriebsmitarbeiter	Dr.	29.Mai.1960	02.Jan.1994	Edgeham Hollo
8	Callahan	Laura	Vertriebskoordinatorin	Frau	09.Jan.1958	05.Mrz.1994	4726 - 11th Ave
9	Dodsworth	Anne	Vertriebsmitarbeiterin	Frau	02.Jul.1969	15.Nov.1994	7 Houndstooth
21	Smith	Eva	Vertriebsmitarbeiterin	Frau	08.Dez.1969	23.Jul.2004	507 - 20th Ave.
22	Wels	Berhard	Vertriebsmitarbeiter	Herr	01.Feb.1959	23.Jul.2004	908 W. Capital
23	Russel	Brenda	Vertriebsmitarbeiterin	Frau	30.Aug.1970	23.Jul.2004	722 Moss Bay E
26	Held	Bernd	Externer Berater	Herr	02.Apr.1969		
(Neu)							

Datensatz: 13 von 13 | Kein Filter | Suchen

Abbildung 165: Der neue Datensatz wurde am Ende der Tabelle eingefügt.

Im folgenden Beispiel aus *Listing 282* wird die Tabelle NEUEKUNDEN aus der Datenbank ABFRAGEN.MDB in die Tabelle KUNDEN der Datenbank NORWIND.MDB geschrieben. Sehen Sie sich zunächst die Tabelle NEUEKUNDEN in *Abbildung 166* an.

Abbildung 166: Diese Kunden sollen in eine bereits bestehende Tabelle überführt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 282: Neue Kunden einlesen

Abbildung 167: Die neuen Kunden wurden übertragen und einsortiert.

229 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 283* werden aus der Tabelle LIEFERANTEN alle Lieferanten entfernt, die aus Japan kommen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 283: Datensätze löschen mit DELETE

Geben Sie bei der WHERE-Klausel das Kriterium in Apostrophen an, sofern es sich um alphanumerische Vergleichsfelder handelt. Die Apostrophen 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. Die Prozedur für diesen Zweck sehen Sie in *Listing 284*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================
```

Listing 284: Länder mit dem Anfangsbuchstaben »F« löschen

```
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 284: Länder mit dem Anfangsbuchstaben »F« löschen (Forts.)

Vorsicht ist beim nächsten Beispiel aus *Listing 285* geboten. Die Aufgabe besteht darin, alle Milchprodukte aus der Tabelle ARTIKEL2 zu entfernen. Diese Produkte werden durch die Kategorie-Nr. 4 repräsentiert.

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

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

Listing 285: Datensätze über eine Nummer löschen

Enthält der Feldname Sonderzeichen wie beispielsweise das Minuszeichen, dann müssen Sie diesen in eckige Klammern setzen.

230 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'
```

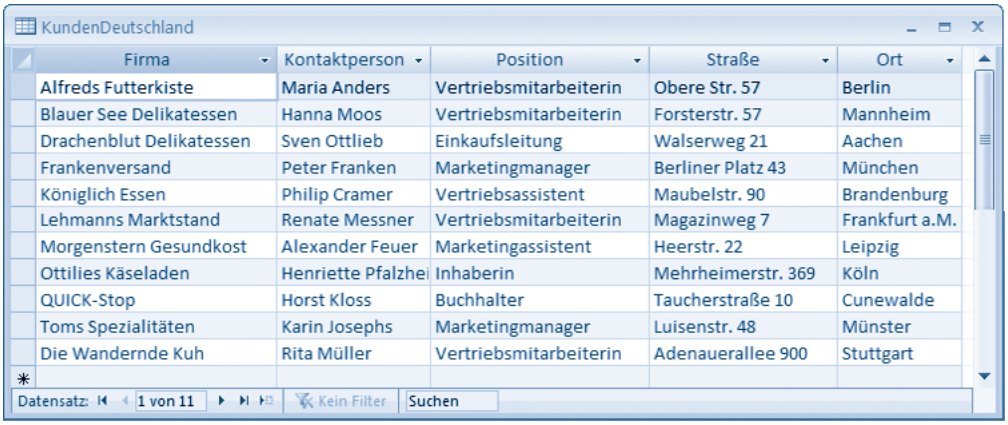
Über die SQL-Anweisung SELECT wählen Sie die Felder aus, die Sie in der neuen Tabelle speichern möchten. Bei der Vergabe der Feldnamen können Sie bis zu 64 Zeichen für einen Namen verwenden. Über INTO geben Sie den Namen der Tabelle an, die angelegt werden soll. Danach legen Sie mit der SQL-Anweisung FROM fest, woher die Daten stammen. Mit der WHERE-Anweisung können Sie die zu übertragenden Datensätze einschränken.

Im folgenden Beispiel aus *Listing 286* werden aus der Tabelle KUNDEN alle Kunden ermittelt, die aus Deutschland stammen und in die Tabelle KUNDENDEUTSCHLAND geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 286: 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, dass nur alle Kunden aus Deutschland übernommen werden sollen.


KundenDeutschland

Firma	Kontaktperson	Position	Straße	Ort
Alfreds Futterkiste	Maria Anders	Vertriebsmitarbeiterin	Obere Str. 57	Berlin
Blauer See Delikatessen	Hanna Moos	Vertriebsmitarbeiterin	Forsterstr. 57	Mannheim
Drachenblut Delikatessen	Sven Ottlieb	Einkaufsleitung	Walserweg 21	Aachen
Frankenversand	Peter Franken	Marketingmanager	Berliner Platz 43	München
Königlich Essen	Philip Cramer	Vertriebsassistent	Maubelstr. 90	Brandenburg
Lehmanns Marktstand	Renate Messner	Vertriebsmitarbeiterin	Magazinweg 7	Frankfurt a.M.
Morgenstern Gesundkost	Alexander Feuer	Marketingassistent	Heerstr. 22	Leipzig
Ottilies Käseladen	Henriette Pfalzhei	Inhaberin	Mehrheimerstr. 369	Köln
QUICK-Stop	Horst Kloss	Buchhalter	Taucherstraße 10	Cunewalde
Toms Spezialitäten	Karin Josephs	Marketingmanager	Luisenstr. 48	Münster
Die Wandernde Kuh	Rita Müller	Vertriebsmitarbeiterin	Adenauerallee 900	Stuttgart

Datensatz: 1 von 11 | Kein Filter | Suchen

Abbildung 168: Alle Kunden aus Deutschland in einer separaten Tabelle

231 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` geben Sie den Namen der neuen Tabelle an, die angelegt werden soll. Direkt hinter der Anweisung `FROM` geben Sie den Namen der ersten Tabelle an. Auf die Anweisung `INNER JOIN` folgt der Name der Vergleichstabelle, die mit der ersten Tabelle abgeglichen werden soll. Nach der Anweisung `ON` erfassen Sie das Vergleichskriterium.

Im folgenden Beispiel aus *Listing 287* 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

Arti	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lager
1	CHAI	Exotic Liquids	Gewürze	10 Kartons x 20 Beut	19,60 €	
2	CHANG	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	20,69 €	
3	ANISEED SYRUP-2	Exotic Liquids	Gewürze	12 x 550-ml-Flascher	10,89 €	
(Neu)					0,00 €	

Datensatz: 1 von 3 | Kein Filter | Suchen

Abbildung 169: Vergleichsartikel in der Tabelle Artikelvgl

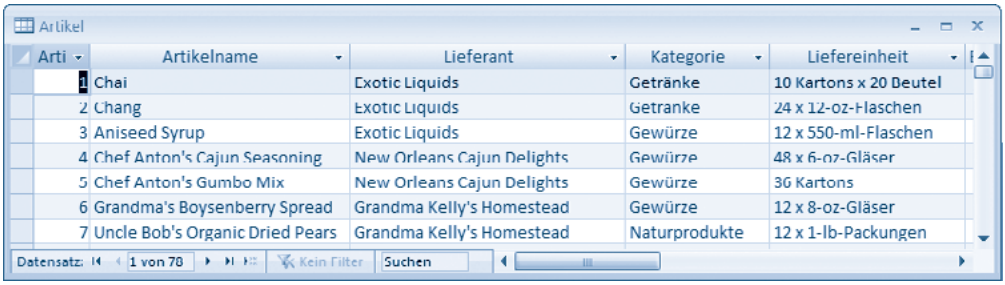


Abbildung 170: Die Ausgangstabelle Artikel

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 287: 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 171: Nur zwei Artikel sind identisch.

Im folgenden Beispiel aus *Listing 288* werden die Tabellen BESTELLUNGEN und KUNDEN miteinander verarbeitet. Über den gemeinsamen eindeutigen Schlüssel KUNDEN-CODE können Sie diese Aufgabe mit dem Einsatz der SQL-Anweisung `INNER JOIN` lösen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 288: 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 beziehen 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.

Abbildung 172: Alle Bestellungen aus den Tabellen Bestellungen und Kunden zusammengestellt

Soll zusätzlich noch eine weitere Abfrage integriert werden, beispielsweise alle Frachtkosten über 50 Euro, dann starten Sie die Prozedur aus *Listing 289*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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
  DoCmd.RunSQL strSQL
```

Listing 289: Eine weitere Einschränkung wurde hinzugefügt.

```
  DoCmd.SetWarnings True
  Exit Sub

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

Listing 289: Eine weitere Einschränkung wurde hinzugefügt. (Forts.)

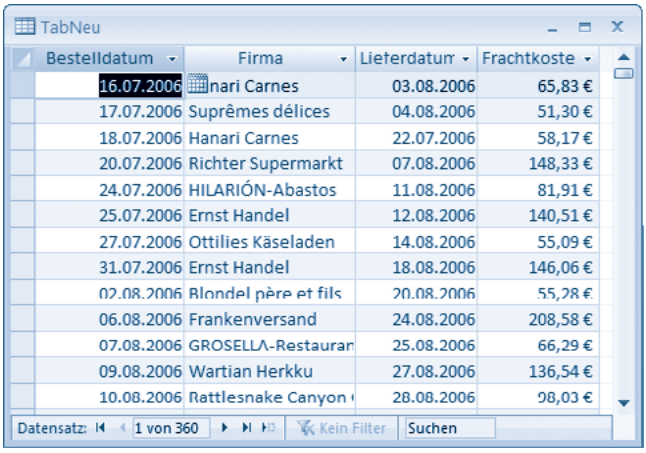


Abbildung 173: Nur Datensätze mit Frachtkosten über 50 € werden berücksichtigt.

232 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 290* werden die beiden Tabellen ARTIKEL und ARTIKELNEU kombiniert und nach dem Artikelnamen sortiert.

Abbildung 174: Die Tabelle mit den neuen Artikeln

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=======================================================

Sub TabellenZusammenführen()
  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 290: Tabellen konsolidieren mit UNION

In der Prozedur aus *Listing 290* wurden alle Datenfelder der beide Tabellen in der Ergebnistabelle gespeichert. Sollen jedoch nur bestimmte Felder übertragen werden, dann starten Sie die Prozedur aus *Listing 291*. In dieser Prozedur werden nur die Felder ARTIKELNAME, EINZELPREIS und LAGERBESTAND in die Tabelle ERGEBNISTAB übertragen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=======================================================
```

Listing 291: Eine Auswahl einiger Felder beider Tabellen in Zieltabelle übertragen

```
Sub TabellenZusammenführenAuswahl()
  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 291: Eine Auswahl einiger Felder beider Tabellen in Zieltabelle übertragen (Forts.)

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` geben Sie die Namen der Tabellen an, die Sie vereinen möchten. Über die `ORDER`-Klausel sortieren Sie die Ergebnistabelle nach dem Artikelnamen.

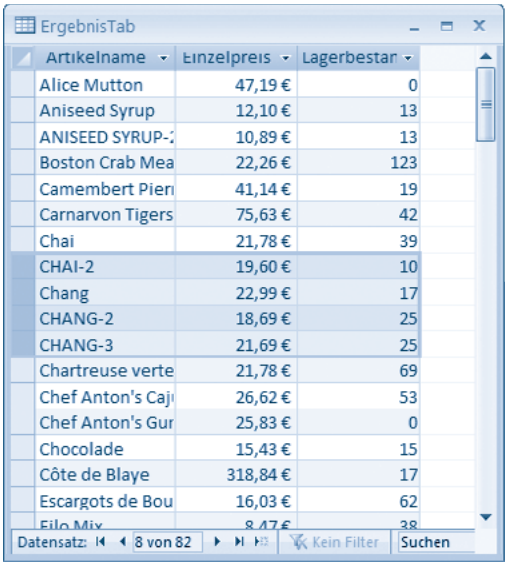


Abbildung 175: Es wurden nur die spezifizierten Felder übertragen.

233 Abfragen mit Berechnungen ausführen

Wie schon im vorherigen Kapitel angesprochen, können mathematische Funktionen wie Summe, Mittelwert und andere aus der *Tabelle 101* in SQL-Anweisungen eingebaut werden.

Funktion	Beschreibung
SUM	Summe
AVG	Mittelwert
MIN	Kleinster Wert
MAX	Größter Wert
COUNT	Anzahl
STDEV	Standardabweichung
VAR	Varianz
FIRST	Erster Satz
LAST	Letzter Satz

Tabelle 101: Mathematische Funktionen für SQL-Anweisungen

Im folgenden Beispiel aus *Listing 292* sollen die Bestellungen pro Hersteller aus der Tabelle BESTELLUNGEN gezählt und in einer Ergebnistabelle ausgegeben werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=======================================================

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
```

Listing 292: Datensätze zählen über die Funktion COUNT

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

Listing 292: 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 176: Die Anzahl der Bestellungen pro Kunde wird dargestellt.

Im nächsten Beispiel aus *Listing 293* werden aus der Tabelle ARTIKEL die Lagermengen pro Hersteller summiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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]"
```

Listing 293: Lagermengen pro Hersteller summieren

```
  DoCmd.SetWarnings False
  DoCmd.RunSQL strSQL
  DoCmd.SetWarnings True
  Exit Sub

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

Listing 293: Lagermengen pro Hersteller summieren (Forts.)

Über die Funktion SUM werden alle Lagerbestände pro Lieferant ermittelt. Dabei werden die summierten Lagerbestände im Feld SUMBESTAND der Tabelle ERGEBNISLAGER ausgegeben.

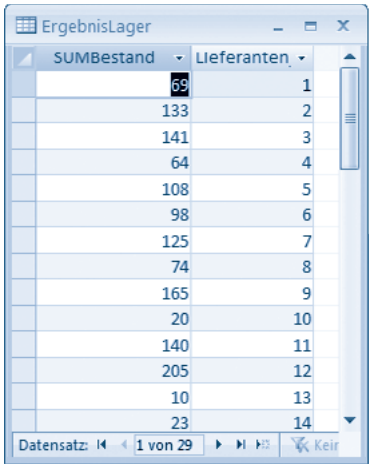


Abbildung 177: Die Lagerbestände der einzelnen Lieferanten

234 ALTER (Tabellenänderungsabfrage)

Um eine bereits vorhandene Tabelle zu ändern, können Sie die SQL-Anweisung ALTER TABLE einsetzen. Dabei müssen Sie genau angeben, welche Änderungen Sie an der Tabelle vornehmen möchten. Dafür stehen Ihnen weitere SQL-Anweisungen 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 Tabellenname SQLAnw Name Datentyp
```

Im Argument `Tabellenname` geben Sie den Namen der Tabelle an, in der Sie eine der vier Operationen durchführen möchten. Über die SQL-Anweisung `SQLAnw` legen Sie fest, welche Aktion Sie ausführen möchten. Im Argument `Name` geben Sie den Namen der/des Spalte/Indexes an. Im letzten Argument `Datentyp` geben Sie einen Datentyp gemäß *Tabelle 102* 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 102: Die zur Verfügung stehenden Datentypen

Beim folgenden Beispiel aus *Listing 294* wird der Tabelle ARTIKEL ein zusätzliches Feld mit dem Namen ALTERNATIVARTIKEL hinzugefügt. Das Feld hat den Datentyp `Text` und eine Feldgröße von 100 Zeichen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub SpalteEinfügen()
  Dim strSQL As String

  On Error GoTo Fehler
  strSQL = "ALTER TABLE Artikel ADD COLUMN AlternativArtikel Text(100)"

  DoCmd.SetWarnings False
  DoCmd.RunSQL strSQL
  DoCmd.SetWarnings True
  Exit Sub
```

Listing 294: Eine Spalte hinzufügen über ADD COLUMN

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

Listing 294: Eine Spalte hinzufügen über ADD COLUMN (Forts.)

Im nächsten Beispiel aus *Listing 295* wird die Spalte AlternativArtikel aus der Tabelle Artikel wieder gelöscht.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 295: Eine Spalte löschen über DROP COLUMN

235 CREATE TABLE (Tabelle erstellen)

Verwenden Sie die SQL-Anweisung `CREATE TABLE`, um eine neue Tabelle in Ihrer Datenbank anzulegen. Dabei bestimmen Sie den Namen, den Feldtyp und die Größe des Feldes. Die so erstellte Tabelle ist vorerst aber noch leer. Die Syntax dieser Anweisung lautet:

```
CREATE TABLE Tabellenname (XY Text(30), YY Text(10))
```

Im Argument `Tabellenname` 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 296* wird eine neue Tabelle mit dem Namen KundenAkt angelegt, die folgende Felder enthalten soll:

- Firma (Textfeld mit max. 30 Zeichen)
- Kontaktperson (Textfeld mit max. 30 Zeichen)

- ORT (Textfeld mit max. 20 Zeichen)
- LAND (Textfeld mit max. 20 Zeichen)

```
'=================================================
' Auf CD      Buchdaten\Beispiele
' 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 296: Eine Tabelle erstellen über CREATE TABLE

Denken Sie daran, am Ende der Prozedur mithilfe der Methode `RefreshDataBaseWindow` die Ansicht zu aktualisieren, da sonst die neue Tabelle nicht angezeigt wird. Diese Methode simuliert das Drücken der Taste F5.

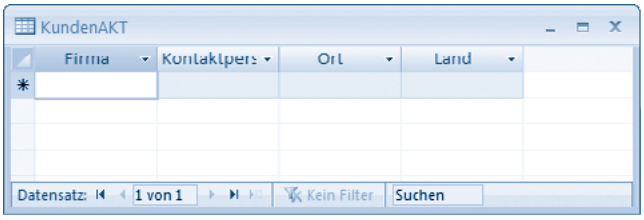


Abbildung 178: Die neue Tabelle wurde angelegt.

Im folgenden Beispiel aus *Listing 297* wird eine neue Tabelle mit dem Namen ALTERNATIVARTIKEL angelegt und das Datenfeld LAGERBESTAND so definiert, dass eine Eingabe erforderlich ist, das Feld also nicht den Wert `NULL` haben darf.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 297: Neue Tabelle mit Muss-Feldern anlegen

236 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 Tabellenname (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 298* wird zunächst eine neue Tabelle mit dem Namen ARTIKELNEU2 erstellt und danach ein Datenfeld mit einem eindeutigen Index ausgestattet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 298: Neue Tabelle mit Index über CREATE UNIQUE INDEX erstellen

237 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 Tabellenname
```

Im Argument `NameIndex` geben Sie den Namen des Indexes an, den Sie entfernen möchten. Im Argument `Tabellenname` geben Sie den Namen der Tabelle an, die den Index enthält.

Im folgenden Beispiel aus *Listing 299* wird der eindeutige Index des Datenfeldes ART-NR der Tabelle ARTIKELNEU2 entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 299: Einen Index über DROP INDEX entfernen

238 DROP TABLE (Tabelle löschen)

Verwenden Sie die SQL-Anweisung `DROP TABLE`, um eine Tabelle aus Ihrer Datenbank zu löschen. 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 300* wird die Tabelle ARTIKELNEU2 aus der Datenbank entfernt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDoCmd
'=====================================================

Sub TabelleEntfernenAbfrage()
  Dim strSQL As String

  On Error GoTo Fehler
  strSQL = "DROP TABLE ArtikelNeu2"
```

Listing 300: Eine Tabelle über DROP TABLE löschen

```
  DoCmd.SetWarnings False
  DoCmd.RunSQL strSQL
  DoCmd.SetWarnings True
  Application.RefreshDatabaseWindow
  Exit Sub

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

Listing 300: Eine Tabelle über DROP TABLE löschen (Forts.)

Eigene Abfragen

Auf den folgenden Seiten finden Sie Beispiele, wie Sie eigene Abfragen definieren und speichern können.

239 Abfrage erstellen und speichern (ADO)

Im folgenden Beispiel aus *Listing 301* wird die Tabelle ARTIKEL ausgewertet, indem alle Artikel mit einem Einzelpreis größer als 20 ermittelt und mit der Abfrage gespeichert werden.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADO
'=====================================================

Sub AbfrageErstellen()
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set rst = _
    CurrentProject.Connection.Execute _
    ("CREATE VIEW ErsteView (Artikelname, Einzelpreis, Lagerbestand) " & _
    "AS SELECT Artikelname, Einzelpreis, Lagerbestand FROM Artikel " & _
    "WHERE Einzelpreis > 20", , adCmdText)

  Set rst = Nothing
  Exit Sub

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

Listing 301: Eine Abfrage erstellen (ADO)

Über die SQL-Anweisung `CREATE VIEW` können Sie eine Abfrage erstellen und speichern. Geben Sie hinter dieser Anweisung den Namen der Abfrage an. Danach führen Sie die Datenfelder, getrennt durch Kommas, auf, 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.

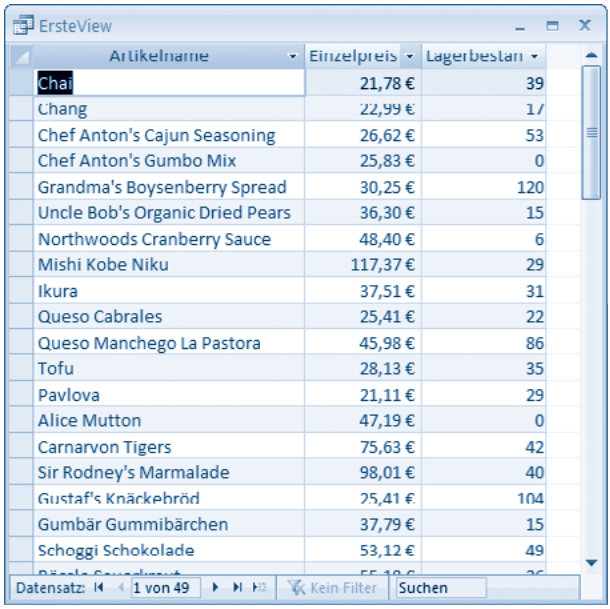


Abbildung 179: Die erste Abfrage wurde erstellt.

240 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 302* werden alle Daten aus der Tabelle KUNDEN extrahiert, bei denen die Kunden aus Deutschland kommen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlDAO
'=======================================================

Sub AbfrageErstellenDAO()
  Dim dbs As DAO.Database
  Dim qdf As DAO.QueryDef
```

Listing 302: Eine Abfrage erstellen (DAO)

```
  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 302: Eine Abfrage erstellen (DAO) (Forts.)

Deklarieren Sie zu Beginn der Prozedur aus *Listing 302* zwei Objektvariable. 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 der Datenbank entspricht, die gerade geöffnet ist.

Der Objektvariablen `qdf` weisen Sie die neue Abfrage zu, die Sie über die Methode `CreateQueryDef` erzeugen. 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 zu bestimmen, was die Abfrage genau machen soll. Nach dem Ausführen dieser Methode wird die Abfrage automatisch erzeugt und gespeichert.

Heben Sie am Ende der Prozedur über das Schlüsselwort `Nothing` Objektverweise auf.

NeueDAOAbfrage

Kunden-(	Firma	Kontaktperson	Position	Straße	Ort	Regio	PLZ	Land
ALFKI	Alfreds Futterkiste	Maria Anders	Vertriebsmitarbeiterin	Obere Str. 57	Berlin		12209	Deutschland
BLAUS	Blauer See Delikatessen	Hanna Moos	Vertriebsmitarbeiterin	Forsterstr. 57	Mannheim		68306	Deutschland
DRACD	Drachenblut Delikatesse	Sven Ottlieb	Einkaufsleitung	Walserweg 21	Aachen		52066	Deutschland
FRANK	Frankenversand	Peter Franken	Marketingmanager	Berliner Platz 43	München		80805	Deutschland
KOENE	Königlich Essen	Philip Cramer	Vertriebsassistent	Maubelstr. 90	Brandenburg		14776	Deutschland
LEHMS	Lehmanns Marktstand	Renate Messner	Vertriebsmitarbeiterin	Magazinweg 7	Frankfurt a.M.		60528	Deutschland
MORGK	Morgenstern Gesundkost	Alexander Feuer	Marketingassistent	Heerstr. 22	Leipzig		04179	Deutschland
OTTIK	Ottilies Käseladen	Henriette Pfalzhei	Inhaberin	Mehrheimerstr. 369	Köln		50739	Deutschland
QUICK	QUICK-Stop	Horst Kloss	Buchhalter	Taucherstraße 10	Cunewalde		01307	Deutschland
TOMSP	Toms Spezialitäten	Karin Josephs	Marketingmanager	Luisenstr. 48	Münster		44087	Deutschland
WANDK	Die Wandernde Kuh	Rita Müller	Vertriebsmitarbeiterin	Adenauerallee 900	Stuttgart		70563	Deutschland

Datensatz: 1 von 11 | Kein Filter | Suchen

Abbildung 180: Alle Kunden aus Deutschland sind über eine Abfrage schnell verfügbar.

241 Abfragen über das Katalog-Objekt realisieren

Möchten Sie eine Abfrage erstellen und diese dann auch als Abfrage in Access speichern, dann müssen Sie in der Entwicklungsumgebung unter dem Menü EXTRAS und dem Befehl VERWEISE die Bibliothek MICROSOFT ADO EXT. <X.Y> FOR DDL AND SECURITY einbinden. <X.Y> steht hierbei für die Versionsnummer, die von der Access-Version abhängig ist, die Sie einsetzen. Bei Access 2007 ist die Version dieser Bibliothek beispielweise 2.8. Die genaue Bezeichnung ist dann MICROSOFT ADO EXT. 2.8 FOR DDL AND SECURITY.

Über die Eigenschaft `CommandText` können Sie die SQL-Anweisung formulieren. Dafür stehen zwei verschiedene Syntaxen zur Verfügung:

- `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 der 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. Gleich danach legen Sie fest, um welche Art der Dateneingabe es sich dabei handeln soll.

242 Normale Abfragen erstellen

Bei der Prozedur aus *Listing 303* wird zunächst eine recht einfache Abfrage erstellt, die alle Artikel aus der Tabelle ARTIKEL 1:1 übernimmt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub AbfrageAnlegen1()
  Dim cat As New ADOX.Catalog
  Dim cmd As ADODB.Command

  On Error GoTo Fehler
  cat.ActiveConnection = CurrentProject.Connection
  Set cmd = New ADODB.Command

  cmd.CommandText = "SELECT * FROM Artikel"
  cat.Procedures.Append "ArtikelAbfr1", cmd
  Set cmd = Nothing
```

Listing 303: Abfrage erstellen und speichern

```
  Set cat = Nothing
  Exit Sub

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

Listing 303: 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 die Abfrage konkret machen soll. Diese Information haben Sie bereits als SQL-Anweisung in der Variablen `cmd` bekannt gegeben. Vergessen Sie nicht, die Objekte nach ihrem Gebrauch wieder freizugeben.

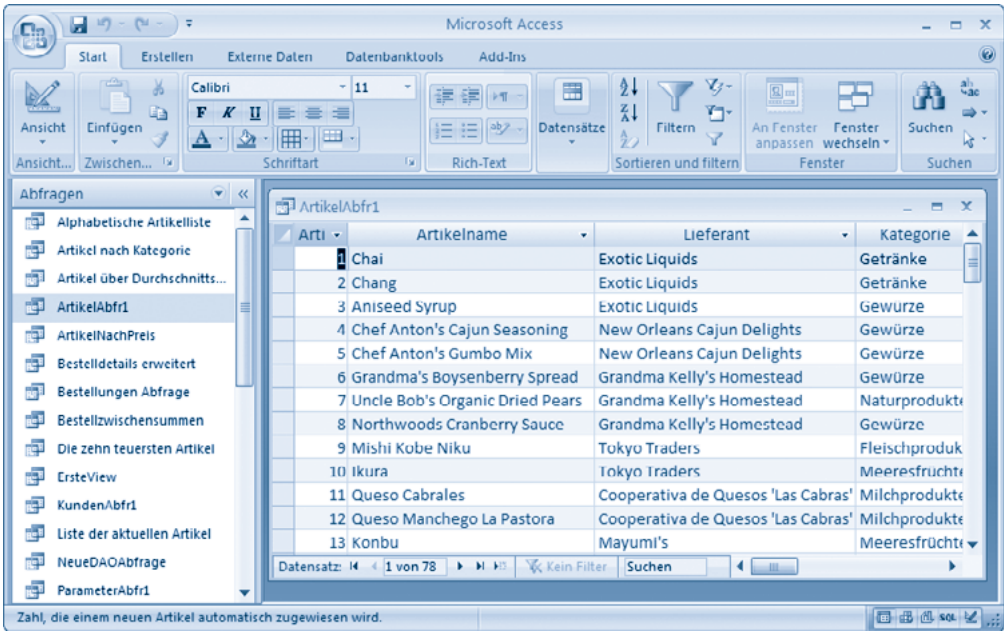


Abbildung 181: Die Abfrage ArtikelAbfr1 wurde angelegt und kann per Klick gestartet werden.

Kundenabfrage durchführen

Bei der folgenden Prozedur aus *Listing 304* wird die Tabelle KUNDEN ausgewertet. Dabei sollen alle Kunden aus Berlin und München ermittelt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub AbfrageAnlegen2()
  Dim cat As New ADOX.Catalog
  Dim cmd As ADODB.Command

  On Error GoTo Fehler
  cat.ActiveConnection = _
    CurrentProject.Connection
  Set cmd = New ADODB.Command

  With cmd
    .CommandText = "SELECT Firma, Kontaktperson, " & _
      "Straße, Ort FROM Kunden " & _
      "WHERE Ort = 'Berlin' OR Ort = 'München'"
  End With

  cat.Procedures.Append "KundenAbfr1", cmd
  Application.RefreshDatabaseWindow

  Set cmd = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 304: Alle Kunden aus bestimmten Orten auflisten

Abbildung 182: Die Ortsabfrage wurde erzeugt.

Lieferanten- und Kundenabfrage ausführen

Im folgenden Beispiel aus *Listing 305* werden alle Kunden und Lieferanten, die aus Deutschland, Spanien oder Frankreich kommen in einer Abfrage verarbeitet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub AbfrageAnlegen3()
  Dim cat As New ADOX.Catalog
  Dim cmd As ADODB.Command

  On Error GoTo Fehler
  cat.ActiveConnection = _
  CurrentProject.Connection
  Set cmd = New ADODB.Command

  With cmd
    .CommandText = "SELECT Land, Ort, Firma, " & _
      "Kontaktperson AS Ansprechpartner " & _
      "FROM Kunden WHERE Land = 'Deutschland' " & _
      "UNION ALL SELECT Land, Ort, Firma, " & _
      "Kontaktperson FROM Lieferanten " & _
      "WHERE Land = 'Deutschland' OR " & _
      "Land = 'Spanien' OR Land = 'Frankreich' " & _
      "ORDER BY Ansprechpartner"
  End With

  cat.Procedures.Append "KundenUndLieferantenAbfr1", cmd
  Application.RefreshDatabaseWindow
  Set cmd = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 305: Mehrere Tabellen auswerten

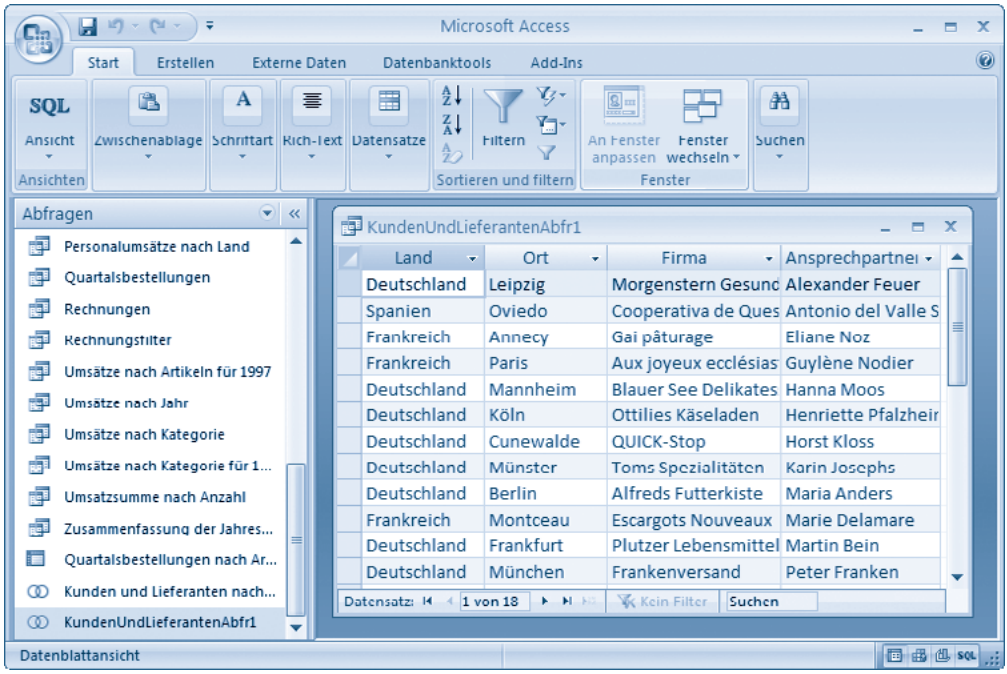


Abbildung 183: Eine UNION-Abfrage wurde erstellt (siehe Ring-Symbole).

243 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 306* 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.

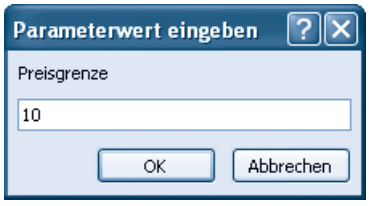


Abbildung 184: Alle Artikel über 10 € sind gefordert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=====================================================

Sub ParameterAbfrageAnlegen()
  Dim cat As New ADOX.Catalog
  Dim cmd As ADODB.Command

  On Error GoTo Fehler
  cat.ActiveConnection = _
  CurrentProject.Connection
  Set cmd = New ADODB.Command

  With cmd
    .CommandText = "PARAMETERS [Preisgrenze] MONEY; " & _
      "SELECT * FROM Artikel WHERE Einzelpreis > [Preisgrenze]"
  End With

  cat.Procedures.Append "ParameterAbfr1", cmd
  Application.RefreshDatabaseWindow

  Set cmd = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 306: Eine Parameterabfrage erstellen – Datentyp Money (ADOX)

Über das Schlüsselwort `PARAMETERS` geben Sie an, dass es sich um eine Abfrage handeln soll, bei der 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. Gleich danach legen Sie fest, um welche Art der Dateneingabe es sich dabei handeln soll. Da es sich hier um einen Preis handelt, geben Sie hier den Datentyp `MONEY` an.

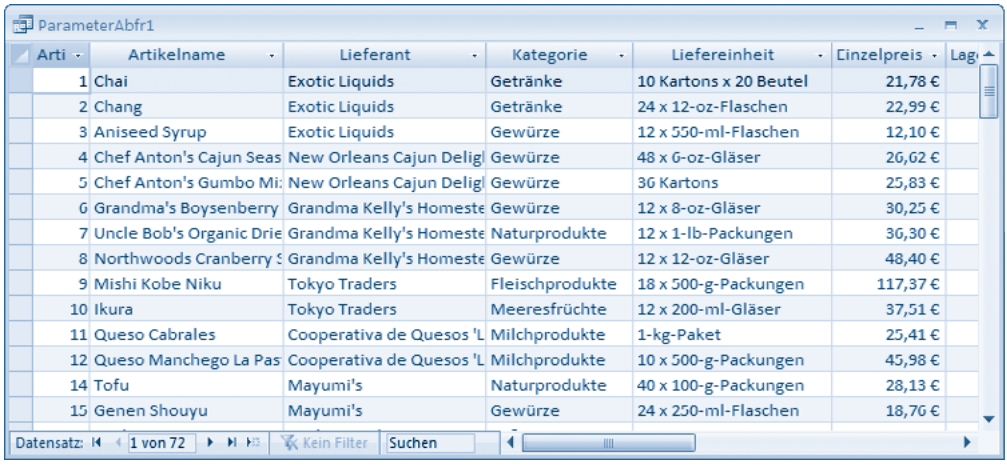


Abbildung 185: Alle Artikel mit einem Einzelpreis über 10 € werden ausgegeben.

Lieferantenabfrage nach Land

Beim nächsten Beispiel aus *Listing 307* wird die Tabelle LIEFERANTEN ausgewertet. Dabei sollen alle Lieferanten aus einem bestimmten Land ermittelt und angezeigt werden. Die Eingabe des gewünschten Landes wird über eine Inputbox vorgenommen. Als Ergebnis werden alle Felder der Tabelle LIEFERANTEN angefordert.

Abbildung 186: Die Eingabe des gewünschten Landes


```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfragen.mdb
' Modul      mdlADOX
'=======================================================

Sub ParameterAbfrageAnlegen2()
  Dim cat As New ADOX.Catalog
  Dim cmd As ADODB.Command

  On Error GoTo Fehler
  cat.ActiveConnection = _
  CurrentProject.Connection
  Set cmd = New ADODB.Command
```

Listing 307: Eine Parameterabfrage erstellen – Datentyp Text (ADOX)

```
  With cmd
    .CommandText = "PARAMETERS [Land Eingabe] Text; " & _
    "SELECT * FROM Lieferanten WHERE Land = [Land Eingabe]"
  End With

  cat.Procedures.Append "ParameterAbfr2", cmd
  Application.RefreshDatabaseWindow

  Set cmd = Nothing
  Set cat = Nothing
  Exit Sub

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

Listing 307: Eine Parameterabfrage erstellen – Datentyp Text (ADOX) (Forts.)

Geben Sie beim Schlüsselwort `PARAMETERS` den Datentyp `Text` an.

Abbildung 187: Alle Lieferanten aus Deutschland stehen zur Verfügung.

244 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

In der folgenden Prozedur aus *Listing 308* wird eine Parameterabfrage auf Basis der Tabelle KUNDEN erstellt. Der Anwender soll dabei nach dem Start der Parameterabfrage das Land eingeben, auf dessen Basis die Kundenliste erstellt werden soll. Im Ergebnis sollen die Felder LAND, ORT und KONTAKTPERSON aufgeführt werden.

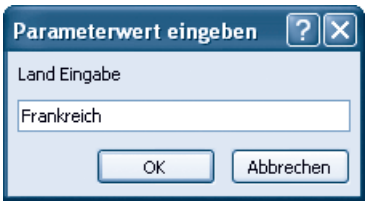


Abbildung 188: Als Parameter wird Frankreich erfasst.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 308: Eine Parameterabfrage anlegen (DAO)

Über das Schlüsselwort `PARAMETERS` geben Sie an, dass es sich um eine Abfrage handeln soll, bei der 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 der Dateneingabe es sich dabei handeln soll. Da es sich hier um einen Text handelt, können Sie hier den Datentyp `Text` angeben.

Abbildung 189: Nur Kunden aus Frankreich werden ausgegeben.

Artikel in einem Preiskorsett ausgeben

Im nächsten Beispiel aus *Listing 309* wird die Tabelle ARTIKEL über eine Parameterabfrage ausgewertet. Dabei sollen alle Artikel ausgegeben werden, die in einem bestimmten Preisniveau liegen. Sie müssen daher zwei Parameter definieren. Im Ergebnis sollen die Felder ARTIKELNAME, EINZELPREIS und LAGERBESTAND aufgeführt werden.

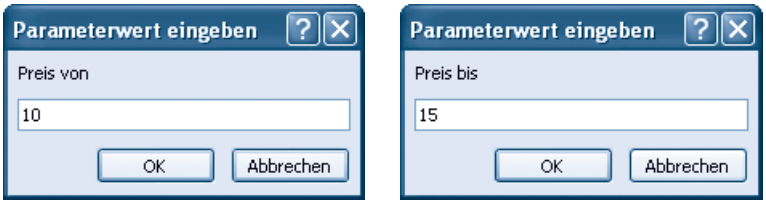


Abbildung 190: Die beiden Parameter werden nacheinander eingegeben.

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

Listing 309: Eine Abfrage mit zwei Parametern anlegen (DAO)

```
  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

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

Listing 309: 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 191: Alle Artikel, die im Preis zwischen 10 und 15 € liegen, werden angezeigt.

Datumsabfrage von Bestellungen

Als letztes Beispiel für Parameterabfragen mit DAO wird in der Prozedur aus *Listing 310* 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.

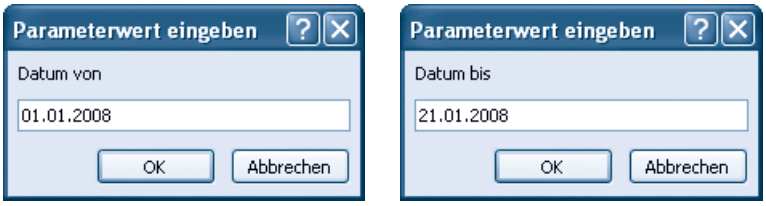


Abbildung 192: Eine Datumsspanne wird über zwei Parameter abgefragt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 310: Eine Zeitspanne wird ausgewertet (DAO).

Die Datumsspanne können Sie über die SQL-Syntax `Bestelldatum BETWEEN [Datum von] AND [Datum bis]` spezifizieren. Da es sich hier um Datumswerte handelt, geben Sie den Datentyp `DateTime` an.

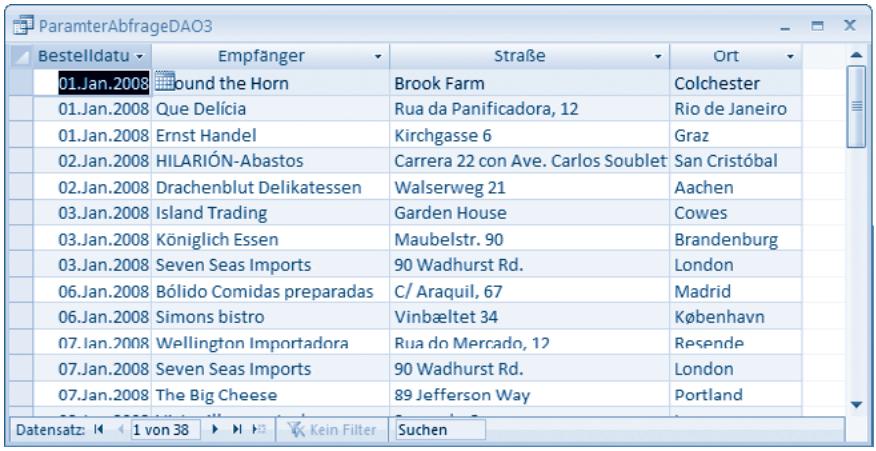


Abbildung 193: Alle Bestellungen in einem bestimmten Zeitrahmen werden ausgegeben.

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.

Das Meldungsfenster MsgBox

In den vorherigen Kapiteln wurde die Methode `MsgBox` bereits häufiger eingesetzt. Diese Methode wird z. B. dazu verwendet, um den Anwender über ein Ergebnis einer Prozedur zu informieren oder auch um eine Warnmeldung auf dem Bildschirm anzuzeigen. Dabei können Sie das Aussehen dieser Maske in vielen Punkten bestimmen. Die Syntax dieser Methode lautet:

```
MsgBox(prompt[, buttons] [, title] [, helpfile, context])
```

Argument	Beschreibung
`prompt`	Erforderlich. Es besteht aus einem 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. Die möglichen Varianten können Sie aus der folgenden Tabelle 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 103: 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.
`vbRetryCancel` oder `5`	Zeigt die Schaltflächen Wiederholen und Abbrechen an.
`vbCritical` oder `16`	Zeigt Meldung mit Stopp-Symbol an.
`vbQuestion` oder `32`	Zeigt Meldung mit Fragezeichensymbol an.

Tabelle 104: Schaltflächen-Symbole für MsgBoxen

Konstante oder Wert	Beschreibung
vbExclamation oder 48	Zeigt Meldung mit Ausrufezeichensymbol an.
vbInformation oder 64	Zeigt Meldung 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 zuerst auf das Meldungsfeld 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 HILFE-Schaltfläche hinzu.

Tabelle 104: Schaltflächen-Symbole für MsgBoxen (Forts.)

Die vorige Tabelle besteht aus verschiedenen Gruppen. Die Gruppe mit den Werten 0 bis 5 beschreibt die Anzahl und den Typ der angezeigten Schaltflächen. Die Gruppe mit den Werten 16, 32, 48, 64 beschreibt die Symbolart. Die dritte Gruppe, 0, 256, 512, legt die Standardschaltfläche fest. Die Gruppe aus den Werten 0 und 4096 legt fest, in welcher Form das Dialogfeld gebunden ist. Sie können bei Aufruf der Funktion MsgBox für das Argument buttons Werte aus den einzelnen Gruppen addieren, dürfen von jeder Gruppe aber nur einen Wert verwenden.

Abhängig davon, welche Schaltfläche der Anwender im Meldungsfeld anklickt, sollen unterschiedliche Aktionen folgen. Wird z. B. die Schaltfläche ABBRECHEN angeklickt, muss die Prozedur 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 105: Rückgabewerte für Schaltflächen

Im folgenden Beispiel aus *Listing 311* soll vor der Löschung eines Formulars nachgefragt werden, ob diese Aktion wirklich durchgeführt werden soll.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=====================================================

Sub RückfrageEinholen()
  Dim intS As Integer

  intS = MsgBox("Wollen Sie das Formular Artikel wirklich löschen?", _
    vbOKCancel + vbQuestion, "Löschenrückfrage")
  If intS = vbOK Then
    DoCmd.DeleteObject acForm, "Artikel"
  Else
    MsgBox "Formular nicht gelöscht!"
  End If
End Sub
```

Listing 311: Die Rückfrage vor dem Löschen eines Formulars

Über die Variable `intS` wird festgehalten, welche Schaltfläche im Meldungsfenster angeklickt wurde. Im Falle, dass dabei der Wert `vbOK` zurückgegeben wird, wurde die Schaltfläche OK angeklickt. Wenn dem so ist, dann löschen Sie mithilfe der Methode `DeleteObject` das Formular ARTIKEL.

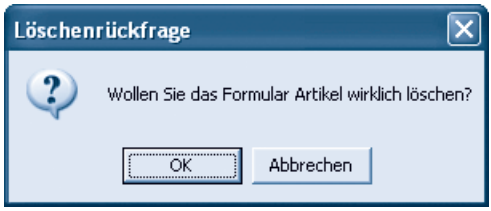


Abbildung 194: Die MsgBox wird angezeigt.

Im nächsten Beispiel aus *Listing 312* wird ein mehrzeiliger Meldungsdialog angezeigt. In diesem Beispiel wird nur eine OK-Schaltfläche angezeigt. Zusätzlich wird das Ausrufezeichen als Symbol angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=====================================================
```

Listing 312: Mehrere Zeilen im Meldungsfenster anzeigen

```
Sub MehrzeiligeMsgBoxAnzeigen()
  MsgBox "Hallo Anwender " & Environ("username") _
    & vbLf & "Heute ist der " & Date _
    & vbLf & "Genau " & Time & " Uhr!", _
    vbOKOnly + vbInformation, "Information"
End Sub
```

Listing 312: 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.

Beim Aufruf der Funktion `MsgBox` könnten Sie statt des Arguments `vbOKOnly + vbInformation` natürlich auch einfach `vbInformation` schreiben, d. h., Sie könnten `vbOKOnly` einfach weglassen. Dies können Sie tun, da die Konstante `vbOKOnly` ohnehin den Wert `0` hat.

Abbildung 195: Eine Bildschirmmeldung mit drei Zeilen

Der Eingabedialog Inputbox

Mithilfe der Methode `InputBox` versetzen Sie den Anwender in die Lage, einzelne Eingaben in einer Maske vorzunehmen. Diese Funktion eignet sich hervorragend für kleinere Aufgaben, 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. Es besteht aus einem 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 106: Die Argumente der Methode InputBox

Argument	Beschreibung
`left` und `top`	Optional. Legt die Position auf dem Bildschirm fest, an der das entsprechende Dialogfeld angezeigt werden soll. So wird beim Argument `left` der horizontale Abstand des linken Rands des Dialogfeldes vom linken Rand des Bildschirms festgelegt. Beim Argument `top` wird der vertikale Abstand des oberen Rands des Dialogfeldes vom oberen Rand des Bildschirms festgelegt.
`helpfile` und `context`	Optional. Diese Argumente werden eingesetzt, wenn Sie auf einen Hilfetext im Eingabefenster verweisen möchten.

Tabelle 106: Die Argumente der Methode InputBox (Forts.)

Im folgenden Beispiel aus *Listing 313* wird eine Inputbox dazu benützt, um eine Eingabe zu erzwingen. Sollte der Anwender keine Eingabe vornehmen, dann wird die Prozedur sofort beendet.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 313: Eine Eingabe über InputBox auswerten

Die FileDialog Standarddialoge

Das `FileDialog`-Objekt kann eingesetzt werden, um Standarddialoge wie den ÖFFNEN- oder SPEICHERN-Dialog aufzurufen. Als Dialoge stehen Ihnen die folgenden vier Standarddialoge zur Verfügung:

Dialogtyp	Beschreibung
msoFileDialogFilePicker	Dateiauswahl
msoFileDialogFolderPicker	Ordnerauswahl
msoFileDialogOpen	Dialog ÖFFNEN
msoFileDialogSaveAs	Dialog SPEICHERN UNTER

Tabelle 107: Die vier Standarddialoge

Im folgenden Beispiel aus *Listing 314* wird der Dialog DATEIAUSWAHL angezeigt und darin Bilder in der Miniaturansicht angeboten. Um das Objekt `FileDialog` nutzen zu können, aktivieren Sie in der Entwicklungsumgebung unter EXTRAS/VERWEISE die Bibliothek MICROSOFT OFFICE <X.Y> OBJECT LIBRARY. <X.Y> steht hierbei für die Version der Bibliothek. Diese hängt von der eingesetzten Office-Version ab. Bei Office 2007 ist es die Version `12.0`.

Damit Sie das Objekt vom Typ `InternetExplorer` nutzen können, um später die ausgewählten Bilder im Internet Explorer anzeigen zu können, aktivieren Sie in der Entwicklungsumgebung unter EXTRAS/VERWEISE die Bibliothek MICROSOFT INTERNET CONTROLS.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=======================================================

Sub DateiAuswahlDialogAnzeigen()
  Dim dlg As FileDialog
  Dim varArrAuswahl 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 varArrAuswahl In .SelectedItems
        Set ie = New InternetExplorer
        ie.Visible = True
        ie.Navigate varArrAuswahl
```

Listing 314: Den Dateiauswahl-Dialog anzeigen

```
      Next
    Else
      MsgBox "Sie haben Abbrechen gedrückt!"
    End If
  End With
End Sub
```

Listing 314: Den Dateiauswahl-Dialog anzeigen (Forts.)

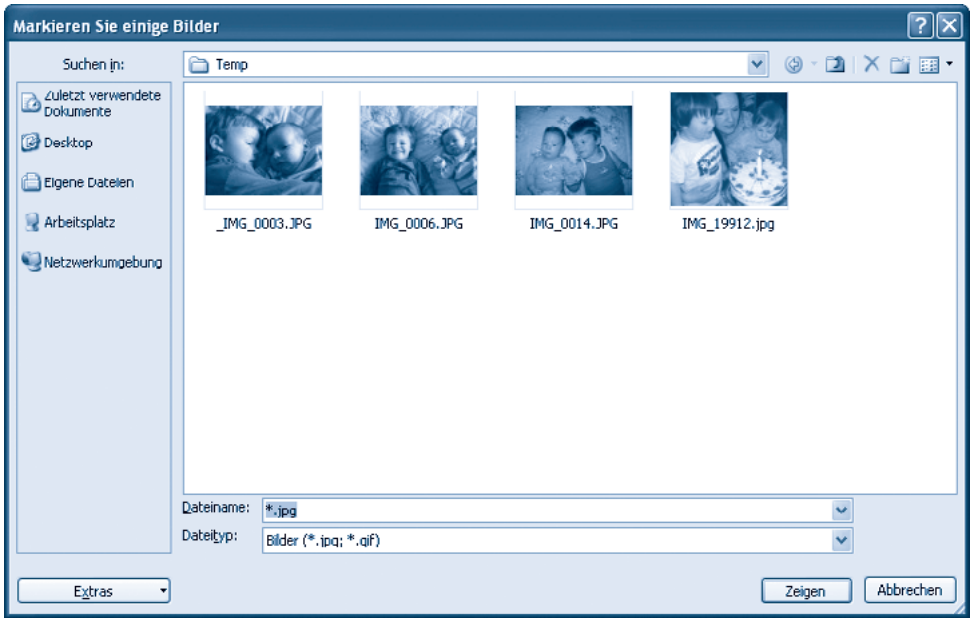


Abbildung 196: Der Standarddialog zur Dateiauswahl wird benutzerdefiniert angezeigt.

Setzen Sie die Eigenschaft `AllowMultiSelect` auf den Wert `True`, wenn der Benutzer die Möglichkeit haben soll, mehrere Dateien in einem Dateidialogfeld auszuwählen. Über die Eigenschaft `ButtonName` können Sie den angezeigten Text einer Aktionsschaltfläche des Dialoges festlegen. Standardmäßig wird diese Eigenschaft auf den Standardtext für den Typ des Dateidialogfeldes 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 Drop-down 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
`msoFileDialogViewSmallIcons`	Ansicht KLEINE SYMBOLE
`msoFileDialogViewThumbnail`	MINIATURANSICHT
`msoFileDialogViewWebView`	WEBANSICHT

Tabelle 108: Die Ansichtstypen für Dateien

Über die Eigenschaft `Title` können Sie den Titel eines Dialoges 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.

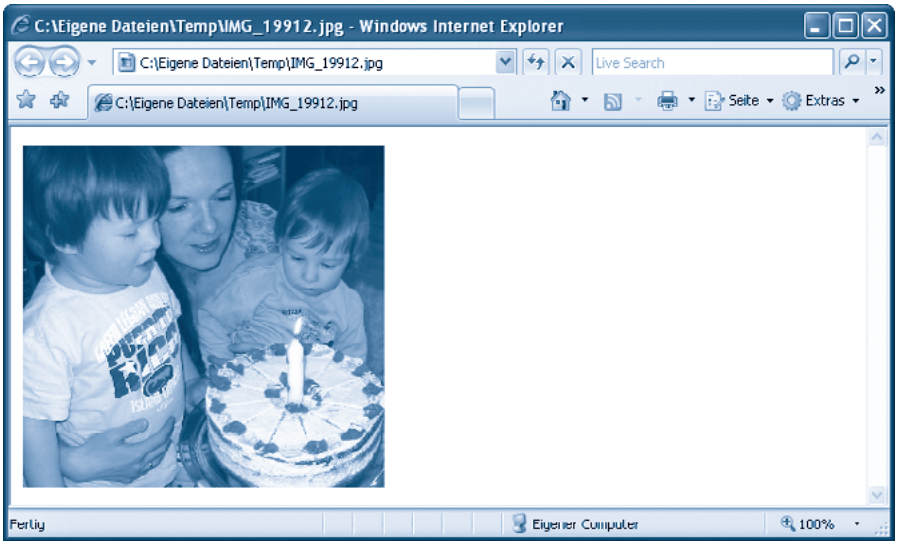


Abbildung 197: Die ausgewählten Bilder werden im Internet Explorer angezeigt.

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 den Befehl DRUCKEN aus dem Menü DATEI über eine Prozedur auswählen. Fügen Sie dazu die Prozedur aus *Listing 315* in der Entwicklungsumgebung von Access ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlDialog
'=====================================================

Sub DialogDrucken()
  DoCmd.OpenReport "Rechnung", acViewPreview
  On Error GoTo Fehler
  DoCmd.RunCommand acCmdPrint
  Exit Sub

Fehler:
End Sub
```

Listing 315: Einen integrierten Dialog aufrufen

Wenden Sie im ersten Schritt das Objekt `DoCmd` an, um Zugriff auf weitere Methoden in Access zu erlangen. Eine Methode davon ist die Methode `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 Drucker-Einstellungen über den Dialog DRUCKEN vornehmen.

Um nun den Dialog DRUCKEN richtig abfangen zu können, müssen Sie dafür sorgen, dass es beim Abbruch dieses Dialoges über die Schaltfläche ABBRECHEN nicht zu einem Laufzeitfehler kommt. Dazu integrieren Sie eine `On Error`-Klausel vor dem Aufruf des Dialoges und verzweigen im Fehlerfall (wenn die Schaltfläche ABBRECHEN angeklickt bzw. die Taste ESC gedrückt wird) zum Paragraphen `Fehler`. Alle weiteren Schaltflächen des Dialoges DRUCKEN können wie gewohnt bedient werden und müssen nicht weiter abgefangen werden.

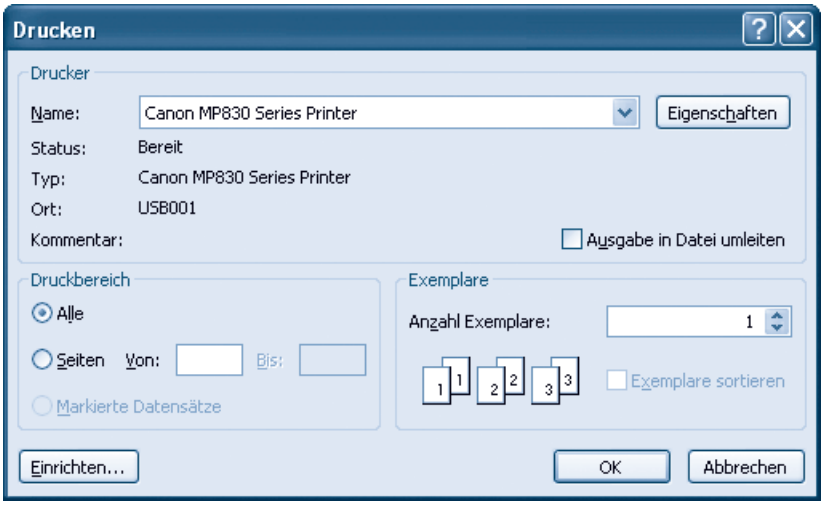


Abbildung 198: Der integrierte Dialog Drucken wurde aufgerufen.

Formulare

Möchten Sie in Access mit Formularen arbeiten, um sich beispielsweise die Eingabe von Daten zu erleichtern oder Suchfunktionen einzusetzen, dann können Sie Ihre eigenen Formulare entwerfen. 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. Bitte beachten Sie, dass bei Access 2007 ein wenig anders vorgegangen wird als bei den Vorgängerversionen:

1. Starten Sie den Formular-Assistenten. Bei Access 2007 wählen Sie hierzu aus der Registerkarte ERSTELLEN in der Gruppe FORMULARE im Modul WEITERE FORMULARE das Element FORMULAR-ASSISTENT.

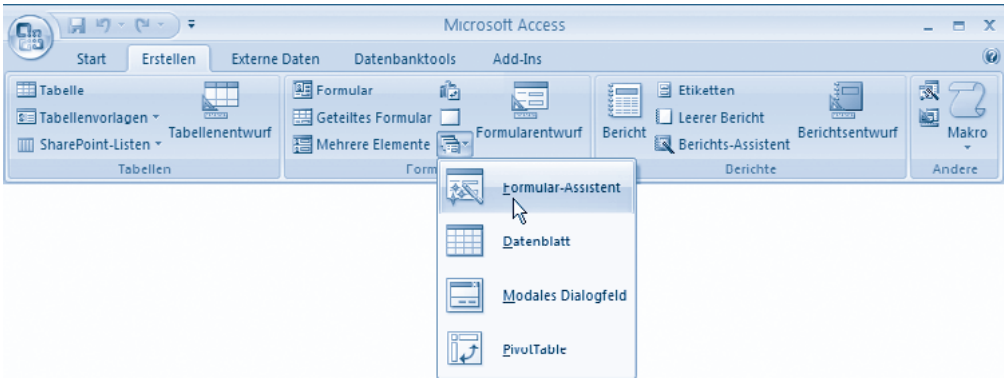


Abbildung 199: Den Formular-Assistenten starten

Bei früheren Access-Versionen begeben Sie sich in das Datenbankfenster und wählen aus dem Menü ANSICHT den Befehl DATENBANKOBJEKTE/FORMULARE. Anschließend klicken Sie auf die Schaltfläche NEU. Bei dem dann angezeigten Dialog NEUES FORMULAR markieren Sie im Listenfeld den Eintrag FORMULAR-ASSISTENT und wählen aus dem Drop-down-Feld die Tabelle oder die Abfrage aus, auf deren Grundlage Sie das Formular erstellen möchten. Stellen Sie dort jetzt einmal die Tabelle ARTIKEL ein und klicken Sie danach auf OK.

2. Der FORMULAR-ASSISTENT startet:

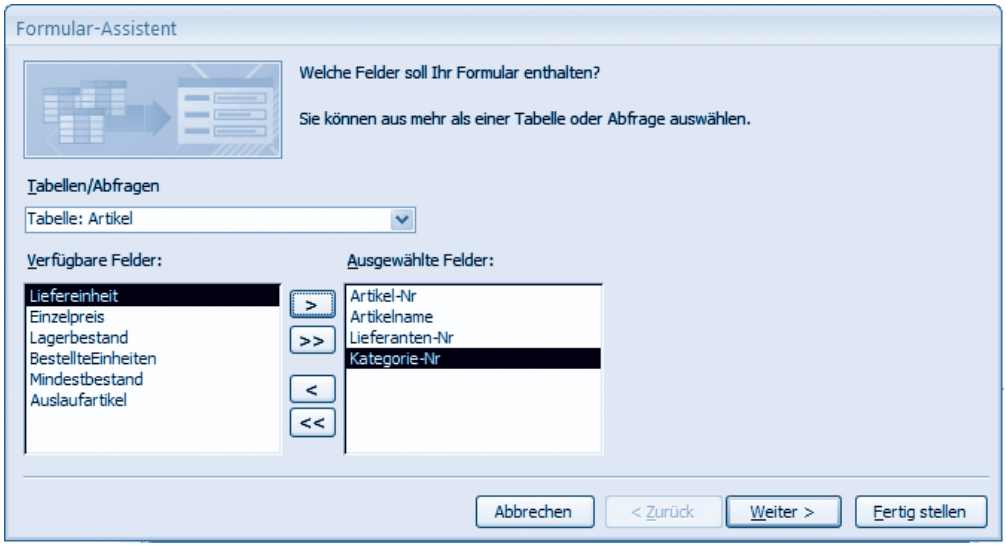


Abbildung 200: Dem Formular die Felder hinzufügen

3. Bei Access 2007 wählen Sie erst hier die Tabelle oder Abfrage aus, auf deren Grundlage Sie das Formular erstellen möchten. Bei sämtlichen Access-Versionen markieren Sie hier die einzelnen Datenfelder, die Ihr Formular anzeigen soll, und klicken Sie auf die Pfeilsymbole, um die Felder einzufügen.
4. Klicken Sie danach auf WEITER.

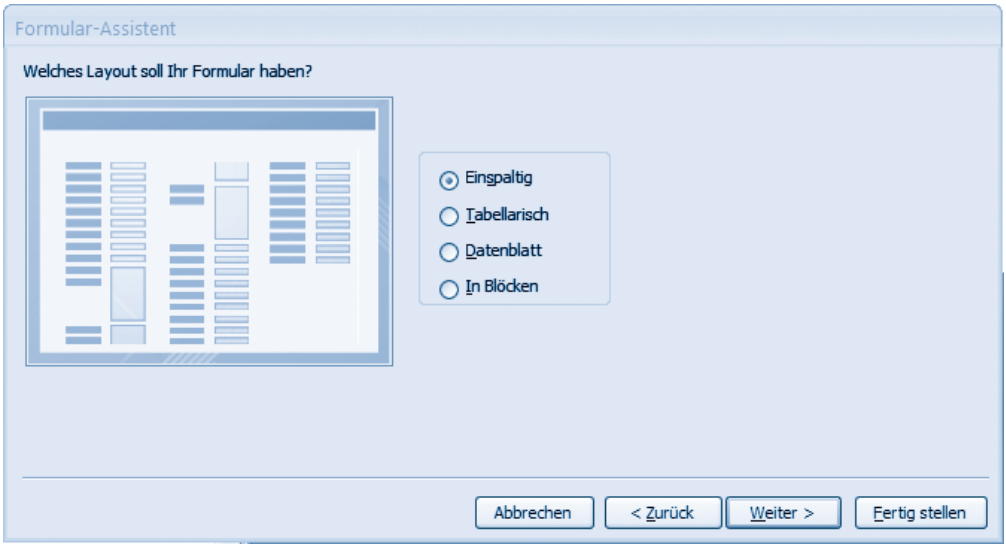


Abbildung 201: Layout des Formulars festlegen

5. Bestimmen Sie das Layout Ihres Formulars, indem Sie eine der angebotenen Optionen auswählen.
6. Klicken Sie danach auf WEITER, um zum nächsten Schritt des Assistenten zu gelangen.

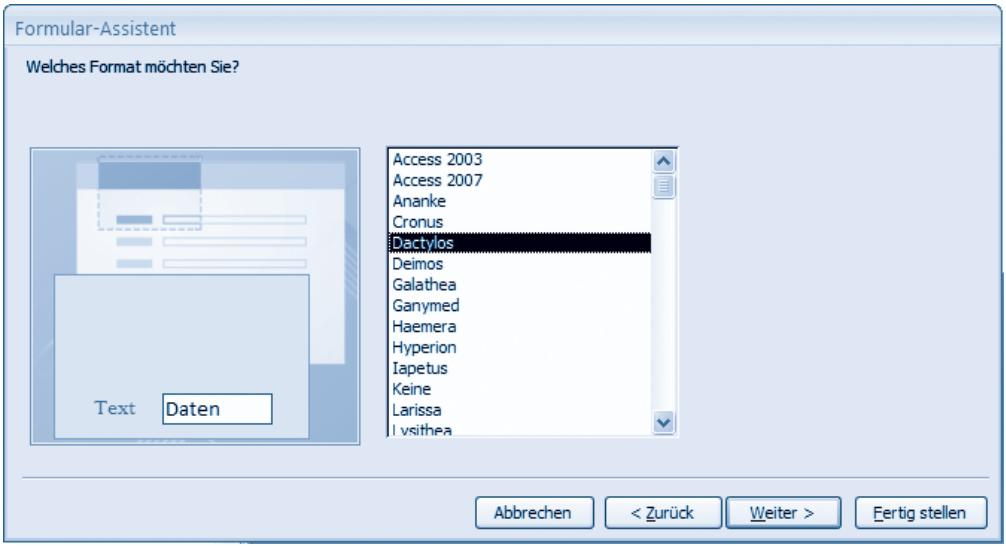


Abbildung 202: Das Format des Formulars festlegen

7. Wählen Sie das gewünschte Format für das Formular aus und betrachten das Resultat im linken Vorschaufenster.
8. Klicken Sie danach auf WEITER.
9. Geben Sie dem Formular im letzten Schritt des Assistenten einen Namen und lassen die Einstellungen wirksam werden, indem Sie auf die Schaltfläche FERTIG STELLEN klicken. Das Formular wird gleich danach standardmäßig aufgerufen.

Abbildung 203: 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 in der Objektliste auf den Eintrag FORMULAR. Klicken Sie danach auf das Symbol NEU, wählen im Dialogfeld NEUES FORMULAR im Listenfeld den Eintrag ENTWURFSANSICHT, wählen die Tabelle bzw. Abfrage aus, auf die sich das Formular beziehen soll, und bestätigen über die Schaltfläche OK.

245 Die Steuerelemente für Formulare

Sobald Sie in die Entwurfsansicht eines Formulars wechseln, wird bei Access-Versionen vor Access 2007 die Symbolleiste TOOLBOX eingeblendet. Diese TOOLBOX enthält die Steuerelemente für Ihr Formular. Bei Access 2007 finden Sie Steuerelemente auf der Registerkarte ENTWURF in der Gruppe STEUERELEMENTE. Die Registerkarte ENTWURF steht übrigens erst zur Verfügung, wenn Sie ein Formular in der Entwurfsansicht öffnen.

Entnehmen Sie der folgenden Tabelle die zur Verfügung stehenden Steuerelemente. Bitte beachten Sie, dass sich die dargestellten Elemente auf die Version Access 2007 beziehen. In früheren Access-Versionen können bestimmte Steuerelemente entweder anders heißen oder eventuell ganz fehlen. Außerdem beschränkt sich diese Liste auf die wichtigsten Funktionen

Element	Beschreibung
	Über das Symbol AUSWÄHLEN können Sie Formularfelder auswählen, um sie beispielsweise zu verschieben.
	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 Feldes (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 Wertes, 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. Es kann immer nur ein Zustand vorherrschen. 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. Bei aktiviertem 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. Bei aktiviertem 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, das 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.

Tabelle 109: Die Steuerelemente für Formulare

Element	Beschreibung
	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 mehrere Einträge ausgewählt werden können.
	BEFEHLSSCHALTFLÄCHEN bieten Ihnen eine Möglichkeit, Aktionen auszuführen, indem Sie einfach auf die Schaltflächen klicken.
	Das Symbol BILD setzen Sie dann 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 UNGEBUNDENES OBJEKTFELD kommt dann zum Einsatz, wenn Sie ein OLE-Objekt in Ihrem Formular anzeigen möchten, das keine Bindung zu Datenfeldern innerhalb der Datenbank besitzt. Dabei kann es sich beispielsweise um eine externe Excel-Tabelle handeln, die Sie anzeigen möchten.
	Das Symbol GEBUNDENES OBJEKTFELD setzen Sie ein, um OLE-Objekte anzuzeigen, die Teil der Tabelle oder Abfrage sind, auf der das Formular basiert. Dies kann beispielsweise eine Reihe von Bildern sein. Jedes Mal, wenn Sie den Datensatz wechseln, wird ein neues Bild angezeigt.
	Mithilfe des Symbols SEITENUMBRUCH legen Sie bei mehrseitigen Formularen den Seitenumbruch fest.
	Das Symbol REGISTERSTEUERELEMENT kann dafür 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 und dabei einen räumlichen Trenneffekt 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 DIAGRAMM EINFÜGEN können Sie Daten in Form eines Diagramms darstellen.
	Über das Symbol HYPERLINK EINFÜGEN erstellen Sie eine Verknüpfung zu einer Datei oder einer Webseite.

Tabelle 109: Die Steuerelemente für Formulare (Forts.)

Element	Beschreibung
	Über das Symbol ACTIVE-X-STEUERELEMENT EINFÜGEN haben Sie Zugriff auf viele weitere Steuerelemente. Welche Steuerelemente Ihnen hier zur Verfügung stehen, ist stark davon abhängig, welche Software-Komponenten auf Ihrem PC installiert sind. Viele dieser Software-Komponenten sind nicht Teil Ihrer Microsoft Office-Installation, sondern wurden von anderen Programmen installiert und stehen im Windows-System als Bibliotheken anderen Programmen wie z. B. Access zur Verfügung. Soll Ihr erstelltes Formular auf anderen PC-Systemen fehlerfrei funktionieren, und Sie verwenden Steuerelemente aus diesem Untermenü, müssen Sie sicherstellen, dass auf dem Ziel-PC-System die entsprechenden Software-Komponenten ebenfalls installiert sind. Es erfordert manchmal eine etwas gründlichere Recherche, um herauszufinden, aus welchen Software-Komponenten bestimmte Steuerelemente stammen und wie Sie diese Software-Komponenten auf anderen PC-Systemen installiert bekommen. In aller Regel hilft Ihnen da eine Suche im Internet da recht schnell weiter, wo Sie oft in Foren entsprechende Antworten finden.

Tabelle 109: Die Steuerelemente für Formulare (Forts.)

246 Formulare auslesen

Um auf einzelne Steuerelemente in Formularen zuzugreifen, müssen Sie erst einmal wissen, wie Sie diese ansprechen können. Über die Auflistung `Controls` können Sie alle Steuerelemente auf einem Formular auslesen.

Das folgende Beispiel aus *Listing 316* 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 die Eigenschaft `Name` ein, um die Namen der Steuerelemente zu ermitteln.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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
  Set frm = Nothing
End Sub
```

Listing 316: Steuerelemente auslesen

Deklarieren Sie zunächst zwei Objektvariable, eine von Typ `Form` und eine vom Typ `Controls`. In der Variablen `frm` vom Typ `Form` 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 diesen im Direktfenster der Entwicklungsumgebung aus.

Abbildung 204: Alle Steuerelemente des Formulars Artikel

Um zu ermitteln, um welchen Steuerelementtyp es sich handelt, setzen Sie die Eigenschaft `ControlType` ein. Diese Eigenschaft gibt einen Indexwert zurück. Hierbei haben Textfelder den Index `109`, Beschriftungsfelder den Wert `100`, Schaltflächen den Wert `104` und Kontrollkästchen den Wert `106`.

Im folgenden Beispiel aus *Listing 317* werden die Steuerelementtypen des Formulars ARTIKEL identifiziert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=====================================================

Sub SteuerelementtypenErmitteln()
  Dim frm As Form
  Dim ctl As Control

  Set frm = Form_Artikel
```

Listing 317: Identifikation der Steuerelemente

```
  For Each ctl In frm.Controls
    Debug.Print "Typ: " & ctl.ControlType & " Name: " & ctl.name
  Next ctl
  Set frm = Nothing
End Sub
```

Listing 317: Identifikation der Steuerelemente (Forts.)

Mithilfe der Eigenschaft `ControlType` können Sie die einzelnen Steuerelementtypen im Formular unterscheiden.

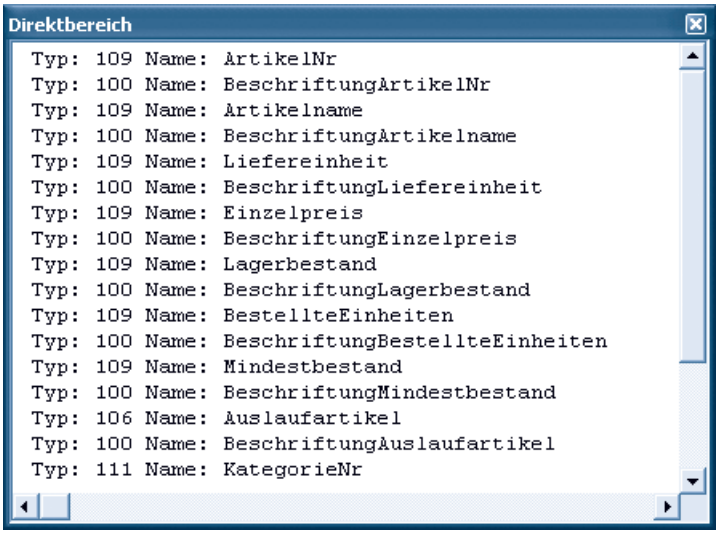


Abbildung 205: Der Typ der Steuerelemente wird ermittelt.

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

Tabelle 110: Die Konstanten, um die Steuerelemente zu identifizieren

Konstante	Steuerelement
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 110: Die Konstanten, um die Steuerelemente zu identifizieren (Forts.)

Im folgenden Beispiel aus *Listing 318* sollen nur die Textfelder des Formulars ARTIKEL ermittelt und im Direktfenster ausgegeben werden.

```
'======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlForm
'======================================================

Sub NurTextfelderAusgeben()
  Dim frm As Form
  Dim ctl As Control

  Set frm = Form_Artikel
  For Each ctl In frm.Controls
    If ctl.ControlType = acTextBox Then
      Debug.Print "Typ: " & ctl.ControlType & " Name: " & ctl.name
    End If
  Next ctl
  Set frm = Nothing
EndSub
```

Listing 318: Nur Textfelder über die Konstante acTextBox ermitteln

Die Eigenschaft `ControlType` kann auch direkt über Konstanten, die Sie in *Tabelle 110* sehen können, abgefragt werden.

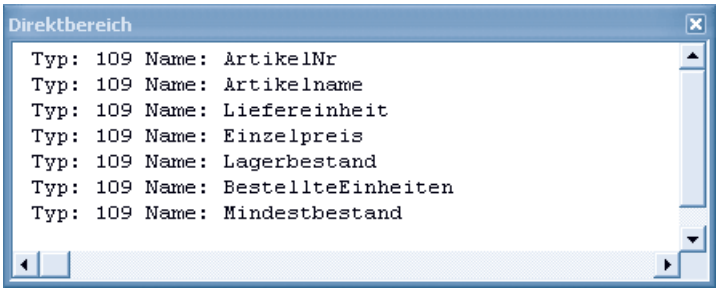


Abbildung 206: Es werden nur Textfelder ausgegeben.

247 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.
`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.

Tabelle 111: Die Argumente der Methode OpenForm

Argument	Beschreibung
	▶ 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 wieder weiterarbeiten, wenn Sie das Formular wieder geschlossen haben.
ÖffnenArgumente	Optional. Dabei können dem Formular bereits beim Öffnen bestimmte Werte bzw. Einstellungen mitgegeben werden.

Tabelle 111: Die Argumente der Methode OpenForm (Forts.)

Beim folgenden Beispiel aus *Listing 319* wird das Formular ARTIKEL in der Dialogansicht aufgerufen und ein Filter voreingestellt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Modul      mdlForm
'=======================================================

Sub FormularAufrufen()
  On Error GoTo Fehler
  DoCmd.OpenForm "Artikel", acNormal, , "Einzelpreis > 50", , acDialog
  Exit Sub

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

Listing 319: Formular öffnen und Filter einstellen

Übergeben Sie der Methode OpenForm den Namen des Formulars, das Sie öffnen möchten, den Anzeigemodus und eine Filter-Bedingung.

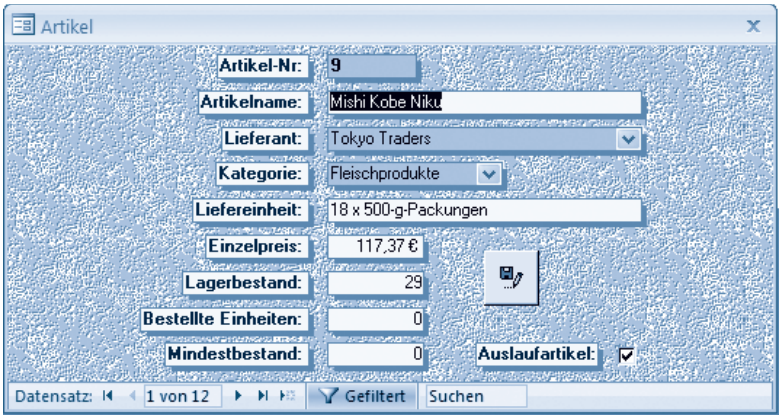


Abbildung 207: Das Formular mit eingestelltem Filter öffnen

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 112: Die Argumente der Methode Close

In der Prozedur aus *Listing 320* wird das Formular ARTIKEL geschlossen. Vorher erfolgt jedoch eine Prüfung, ob das betreffende Formular überhaupt geöffnet ist.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 320: Formular schließen

Über die Funktion `IsLoaded` können Sie feststellen, ob ein bestimmtes Formular geöffnet ist. Wenn ja, dann meldet diese Funktionen den Wert `True` zurück. In diesem Fall wenden Sie die Methode `Close` an, um das Formular zu schließen.

248 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 Tab von einem Formularfeld zum anderen springen. Möchten Sie rückwärtsspringen, drücken Sie die Tastenkombination Umschalt + Tab. Standardmäßig ist diese Reihenfolge schon vorgegeben. Die Aktivierreihenfolge 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 aus dem Kontextmenü den Befehl AKTIVIERREIHENFOLGE.

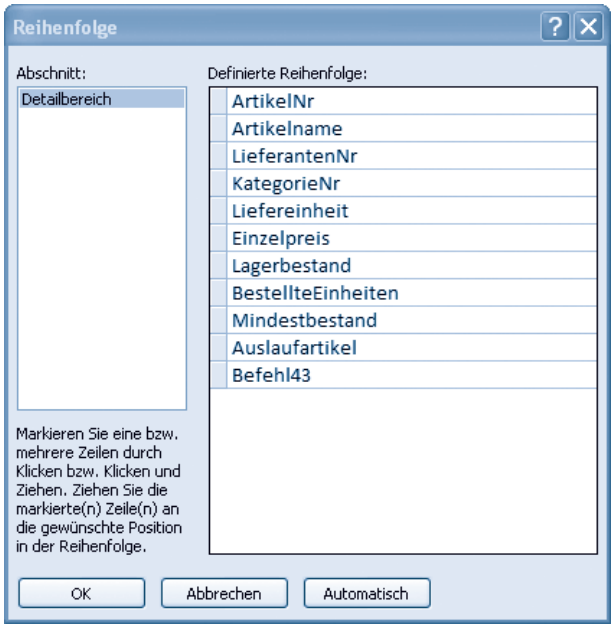


Abbildung 208: Aktivierreihenfolge festlegen

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 einsetzen, klicken Sie auf die jeweilige graue Schaltfläche und ziehen diese an die gewünschte Stelle.
5. Bestätigen Sie Ihre Einstellung mit OK.

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 Tab die einzelnen Felder anspringen. Um wieder zum Formularentwurf zurückzugelangen, klicken Sie das Formular mit der rechten Maustaste an und wählen im Kontextmenü den Befehl ENTWURFSANSICHT aus.

249 Formularhintergrund einstellen

Die Hintergrundfarbe für Formulare kann in Access eingestellt werden. Per VBA-Code stellen Sie diese 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. Der Wert 0 zeigt an, dass diese Farbkomponente nicht vorhanden ist. Der Wert 255 zeigt dagegen an, dass diese Farbkomponente mit ihrer höchsten Intensität verwendet wird.

Im Formular aus *Listing 321* wird nach jedem Klicken auf die Schaltfläche eine andere Hintergrundfarbe eingestellt. Die Auswahl der Farbe wird jeweils nach dem Zufallsprinzip ermittelt. Dabei werden die verwendeten `RGB`-Werte in der Titelleiste des Formulars ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Textfelder
'=====================================================

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 321: Hintergrundfarbe des Formulars festlegen

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 Zusammensetzung der `RGB`-Funktion geben Sie über die Eigenschaft `Caption` des `Form` Objekts aus.

Abbildung 209: Zufallsfarbe per Mausklick einstellen

250 Formular ohne Navigationsschaltflächen aufrufen

Möchten Sie verhindern, dass ein Anwender die Navigationsschaltflächen benutzen kann, dann können Sie die Anzeige dieser Schaltflächen unterdrücken, indem Sie das Formularereignis des Formulars `Form_Load` einsetzen, das automatisch ausgeführt wird, wenn das Formular angezeigt wird. Dort blenden Sie die Schaltflächen einfach 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.

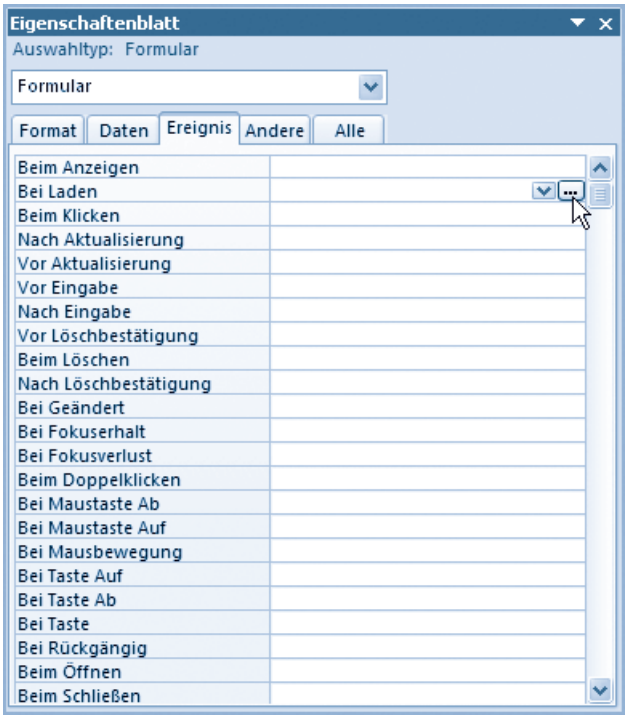


Abbildung 210: Ein Formularereignis einstellen

5. Setzen Sie den Mauszeiger in das Feld BEI 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 mit OK.

Ergänzen Sie den noch leeren Ereignisrahmen wie in *Listing 322* angezeigt:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel2
'=======================================================

Private Sub Form_Load()
  With Me
    .NavigationButtons = False
    .RecordSelectors = False
  End With
End Sub
```

Listing 322: Navigationsschaltflächen ausblenden

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 Dialograndausblenden, dann weisen Sie der Eigenschaft `RecordSelectors` ebenso den Wert `False` zu.

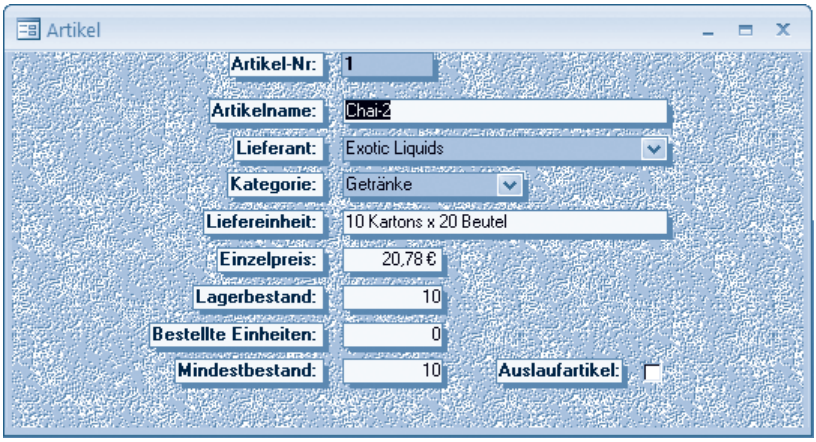


Abbildung 211: Die Steuerungsschaltflächen wurden ausgeblendet.

Hinweis

Im *Kapitel Die Ereignisprogrammierung* dieses Buches können Sie noch mehr Beispiele zu Ereignissen nachschlagen.

Textfelder

Die wichtigsten Felder in einem Formular sind Textfelder, über die Eingaben vorgenommen werden. Für diese Felder können Sie die Hintergrundfarbe sowie die Schriftfarbe bestimmen und bestimmte Spezialeffekte einstellen. Auf den folgenden Seiten können Sie typische Anwendungsbeispiele für den Einsatz von Textfeldern nachschlagen.

251 Hintergrund und Schriftfarbe anpassen

Im folgenden Beispiel aus *Listing 323* werden alle Textfelder des Formulars ARTIKEL1 mit der Hintergrundfarbe GELB sowie der Schriftfarbe ROT ausgestattet.

```
'======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'======================================================
```

Listing 323: Hintergrundfarbe Gelb und Schriftfarbe Rot

```
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

  Set frm = Nothing
End Sub
```

Listing 323: Hintergrundfarbe Gelb und Schriftfarbe Rot (Forts.)

Über die Eigenschaft `BackColor` können Sie die Hintergrundfarbe festlegen. Mithilfe der Eigenschaft `ForeColor` können Sie die Schriftfarbe bestimmen.

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	Rot-Wert	Grün-Wert	Blau-Wert
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 113: Die Standardfarben über die Funktion RGB

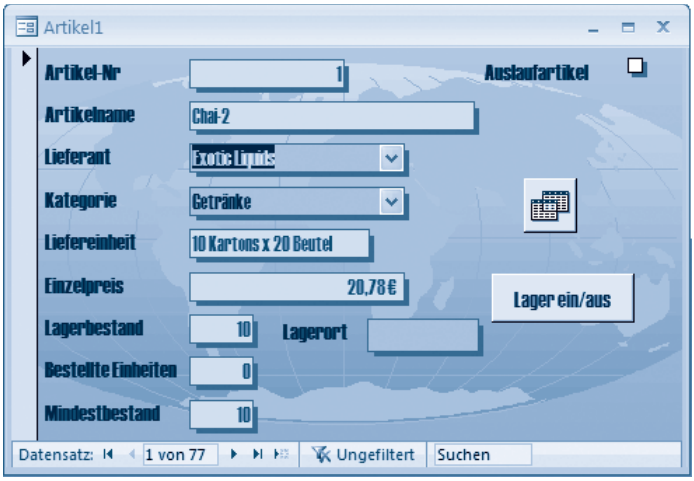


Abbildung 212: Die Formatierungen der Felder wurden vorgenommen.

252 Startposition des Cursors festhalten

Mithilfe der Eigenschaft `SelStart` können Sie die Startposition des markierten Textabschnitts eines Textfeldes festlegen. Sollte noch kein Text eingefügt worden sein, legen Sie über diese Eigenschaft die Einfügemarke des Mauszeigers fest. Standardmäßig wird beim Aktivieren eines Textfeldes immer der komplette Text markiert.

Folgende alternativen 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 114: Alternative Markierungsformen für Textfelder

253 Auswahl aufheben

Standardmäßig wird in einem Textfeld beim Anspringen immer der ganze Text markiert. Diese Markierung können Sie aufheben, indem Sie die Eigenschaft `SelStart` sowie die Eigenschaft `SelLenght` auf den Wert `0` setzen.

Im folgenden Beispiel aus *Listing 324* wird das Ereignis aus *Listing 322* 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
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Form_Load()
 With Me
  .NavigationButtons = False
  .RecordSelectors = False
  .Artikelname.SetFocus
  .Artikelname.SelStart = 0
  .Artikelname.SelLength = 0
 End With
End Sub
```

Listing 324: Die Standardmarkierung in Textfeldern aufheben

Über die Methode `SetFocus` setzen Sie den Mauszeiger in das gewünschte Textfeld.

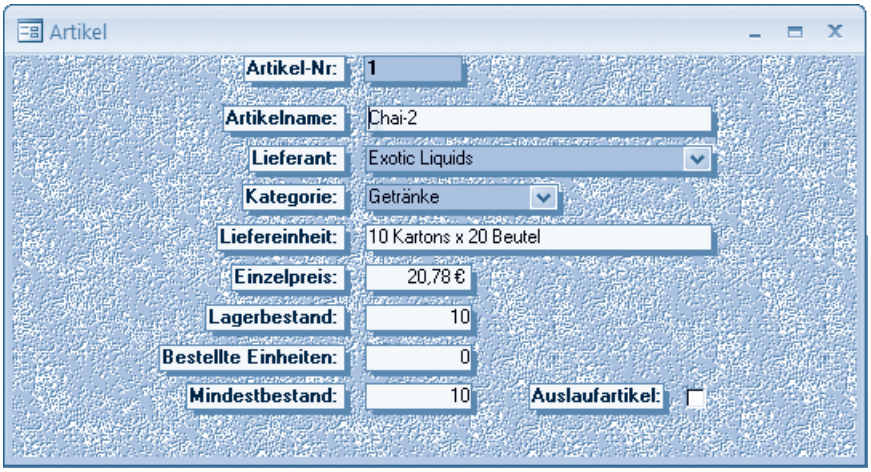


Abbildung 213: Der Mauszeiger steht zu Beginn des Textfeldes Artikelname.

254 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 325* wurde im Formular ARTIKEL1 eine Schaltfläche eingefügt, die beim Anklicken die aktuellen sowie die geänderten Werte in das Direktfenster der Entwicklungsumgebung schreibt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 325: Alte und neue Werte festhalten

Die Eigenschaft `ControlType` gibt Auskunft darüber, ob es sich um ein Textfeld handelt. Wenn ja, dann lässt sich über den Einsatz der Eigenschaft `OldValue` der »alte« Wert vor der Änderung eines Textfeldes festhalten. Den aktuellen Wert fragen Sie über die Eigenschaft `Value` ab.

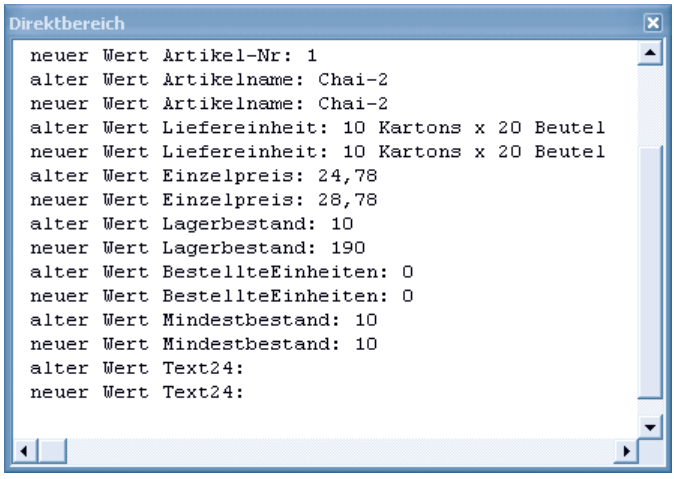


Abbildung 214: Einzelpreis und Lagerbestand wurden geändert.

255 QuickInfo hinzufügen

Setzen Sie die Eigenschaft `ControlTipText` ein, um einem Textfeld eine QuickInfo hinzuzufügen.

Im folgenden Beispiel aus *Listing 326* wird jedem Textfeld des Formulars ARTIKEL1 eine QuickInfo zugewiesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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
  Set frm = Nothing
End Sub
```

Listing 326: 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.

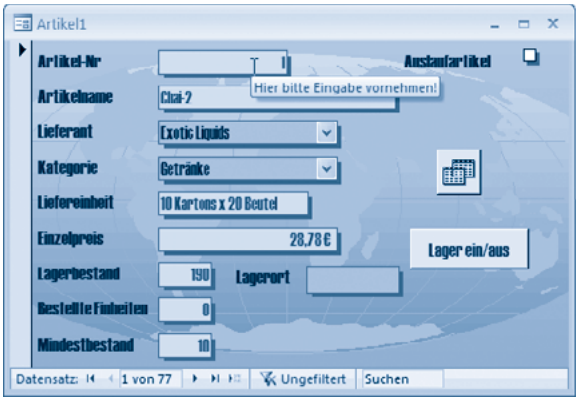


Abbildung 215: QuickInfos hinzufügen

256 Textfelder sperren

Setzen Sie die Eigenschaft `Enabled` auf `False`, um ein Textfeld zu deaktivieren. Es wird dann ausgegraut im Formular angezeigt und kann vom Anwender nicht editiert und auch nicht markiert werden.

Setzen Sie die Eigenschaft `Locked` auf `True`, um ein Textfeld auf andere Art vor Veränderungen zu schützen. Das Textfeld sieht unverändert aus. Der Cursor kann im Textfeld positioniert werden, der Inhalt des Textfeldes kann markiert werden. Das Textfeld kann den Fokus erhalten und Ereignisse auslösen. Eine Änderung des Textfeld-Inhalts durch den Anwender ist jedoch auch hier nicht möglich.

Im folgenden Beispiel aus *Listing 327* wird für das Feld ARTIKELNAME im Formular ARTIKEL1 die Eigenschaft `Enabled` auf `False` gesetzt. Für das Feld EINZELPREIS wird die Eigenschaft `Locked` auf `True` gesetzt. Beide Aufgaben führen Sie am besten gleich direkt beim Öffnen des Formulars durch. In beiden Fällen ist eine Änderung durch den Anwender nicht möglich. Während das Feld ARTIKELNAME ausgegraut erscheint und nicht anwählbar ist, zeigt das Feld EINZELPREIS optisch keine Veränderungen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=======================================================

Private Sub Form_Open(Cancel As Integer)
  On Error GoTo Fehler
  Me!Artikelname.Enabled = False
  Me!Einzelpreis.Locked = True
  Exit Sub

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

Listing 327: Das Feld Artikelname wurde gesperrt.

Soll eine Sperre von allen Textfeldern des Formulars erreicht werden, dann passen Sie die Prozedur aus *Listing 327* wie in *Listing 328* gezeigt an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel
'=======================================================
```

Listing 328: Alle Textfelder eines Formulars werden gesperrt.

```
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
    End If
  Next ctl
  Exit Sub

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

Listing 328: Alle Textfelder eines Formulars werden gesperrt. (Forts.)

Das Sperren von Textfeldern kann beispielsweise auch abhängig vom Anwender vorgenommen werden. Im *Kapitel Die Ereignisprogrammierung* dieses Buches können Sie nachschlagen, wie Sie mithilfe von Ereignissen diesbezüglich eine anwenderorientierte Lösung mit Textfeldern aufsetzen können.

257 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 329* wurde in das Formular ARTIKEL1 eine Umschaltfläche eingefügt, um dynamisch Felder ein- und auszublenden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

 Sub Umschaltfläche23_Click()
'Textfelder ein- und ausblenden

  If Umschaltfläche23.Value = 0 Then
    Me!Lagerbestand.Visible = False
    Me!BestellteEinheiten.Visible = False
    Me!Mindestbestand.Visible = False
    Me!Auslaufartikel.Visible = False
  Else
```

Listing 329: Textfelder ein- und ausblenden

```
    Me!Lagerbestand.Visible = True
    Me!BestellteEinheiten.Visible = True
    Me!Mindestbestand.Visible = True
    Me!Auslaufartikel.Visible = True
  End If
End Sub
```

Listing 329: Textfelder ein- und ausblenden (Forts.)

Das Ein- und Ausblenden der Textfelder richtet sich nach dem Status der Schaltfläche, je nachdem, ob diese gedrückt ist oder nicht.

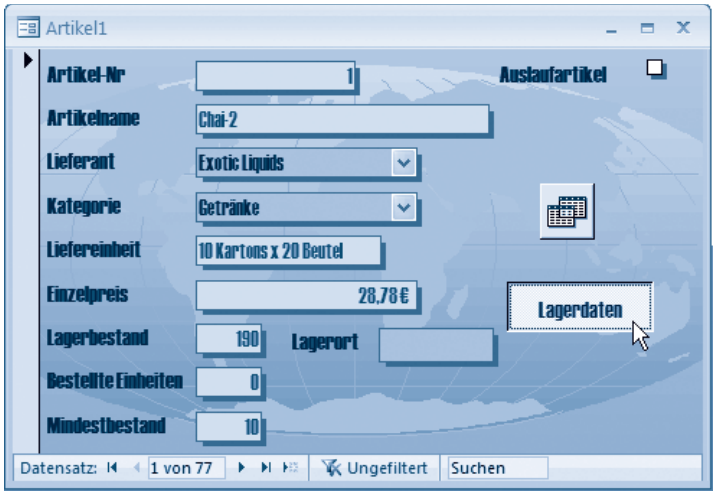


Abbildung 216: Lagerdaten eingeblendet

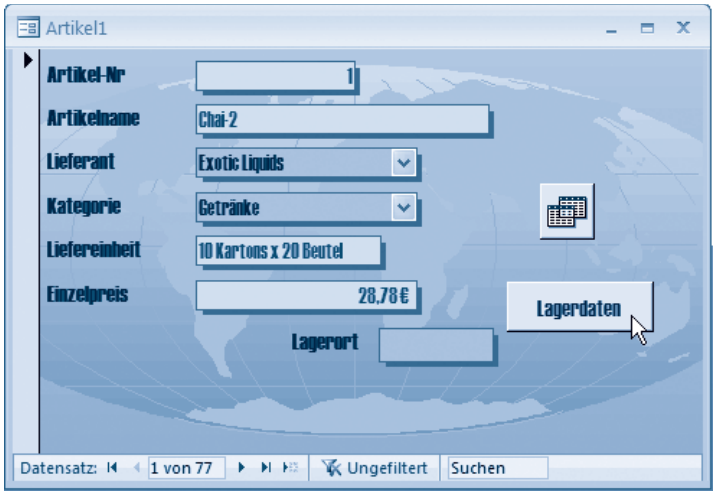


Abbildung 217: Lagerdaten ausgeblendet

258 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.

Beim folgenden Beispiel aus *Listing 330* wurde im ARTIKEL1 das Textfeld LAGERORT eingefügt, das über das Ereignis `Change` überwacht wird. In diesem Feld dürfen nur maximal zehn Zeichen erfasst werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Artikel1
'=====================================================

Private Sub Text24_Change()
  If Len(Text24.Text) > 10 Then
    MsgBox "Es dürfen nur 10 Zeichen eingegeben werden!"
    Text24.Text = Left(Text24.Text, 10)
    Text24.SelStart = 10
  End If
End Sub
```

Listing 330: Die Eingabe auf zehn Zeichen beschränken

Die Eingabelänge des Textfeldes wird bei jeder Eingabe über die Funktion `Len` neu gemessen. Beträgt die Eingabe über zehn Zeichen, dann wird automatisch eine Kürzung vorgenommen, indem mithilfe der Funktion `Left` vom linken Rand des Textfeldes genau zehn Zeichen übertragen werden. Die Zeile `Text24.SelStart = 10` setzt den Cursor wieder ans Ende des Textfeldes.

259 Textfelder überwachen

Soll eine Änderung an einem Textfeld überwacht werden, dann können Sie dies über eine Ereignisprozedur umsetzen. Im folgenden Beispiel aus *Listing 331* wird im Formular ARTIKEL1 das Feld BESTAND überwacht. Um dies zu realisieren, 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
' 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 331: Lagerbestandsänderung festhalten

Die Ereignisprozedur wird einmalig aufgerufen, wenn eine Änderung im Textfeld begonnen wird. Über die Funktion `Environ` und die Konstante `Username` können Sie feststellen, wer soeben die Änderung vornimmt. Die Funktionen `Date` und `Time` liefern aktuelles Datum und Uhrzeit. Über den Befehl `Me.Lagerbestand.Value` fragen Sie den Lagerbestand vor der Änderung ab.

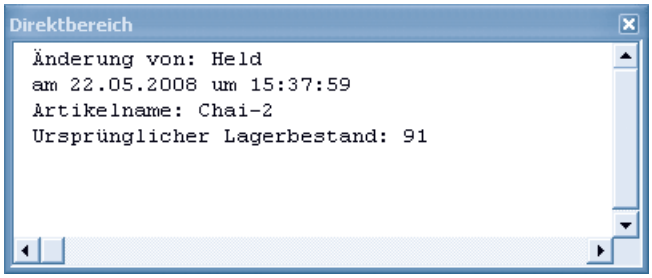


Abbildung 218: Das Feld Lagerbestand wurde überwacht.

260 Textfelder initialisieren

Bei ungebundenen Formularen mit diversen Textfeldern können Sie eine Funktion schreiben, die alle Textfelder in einem Aufwasch initialisiert.

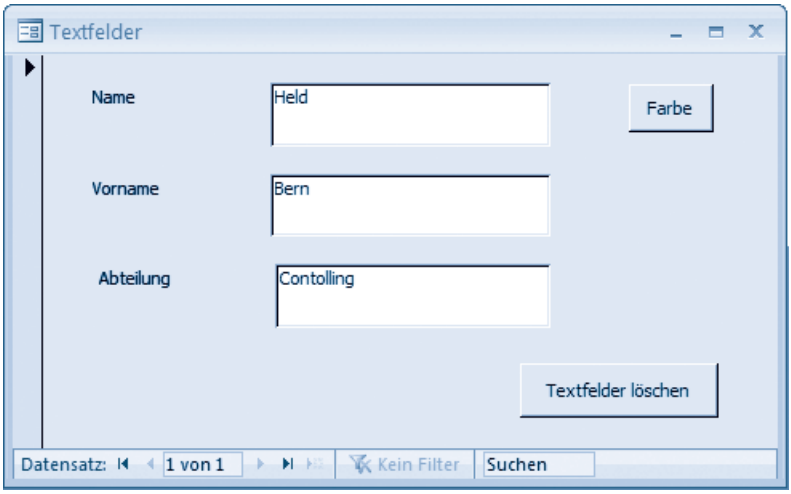


Abbildung 219: Die Textfelder sollen geleert werden.

Um alle Textfelder in einem Formular zu leeren, starten Sie die Prozedur aus *Listing 332*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Textfelder
'=======================================================

Private Sub Befehl6_Click()
  'Textfelder löschen
  Dim ctl As Control

  For Each ctl In Me.Controls
    If TypeName(ctl) = "TextBox" Then ctl.Value = ""
  Next ctl
End Sub
```

Listing 332: 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.

261 Textfelder prüfen

Beim Verlassen eines Textfeldes können Sie überprüfen, ob eine richtige Eingabe vorgenommen wurde. Im Beispiel aus *Listing 333* wird überprüft, ob überhaupt eine Eingabe vorgenommen wurde. Wenn ja, dann erfolgt die zweite Prüfung, ob ein alphanumerischer Eintrag vorgenommen wurde.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 333: Überprüfung des Textfeldes

Prüfen Sie im ersten Schritt, ob überhaupt eine Eingabe gemacht wurde. Wenn nicht, geben Sie eine Meldung aus und beenden Sie die Prozedur mit der Anweisung `Exit Sub`. Wurde eine Eingabe vorgenommen, überprüfen Sie mithilfe der Funktion `IsNumeric`, ob es sich dabei um einen gültigen alphanumerischen Wert handelt. Gibt der Anwender versehentlich eine Zahl ein, dann geben Sie eine Meldung auf dem Bildschirm aus. Durch die Anweisung `Cancel = True` erreichen Sie, dass der Anwender die Textbox nicht verlassen kann, solange ein Eintrag im Textfeld numerisch ist.

Listenfelder

Die wichtigsten Eigenschaften, Funktionen und Methoden für Listenfelder werden jetzt vorgestellt. 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 ein.

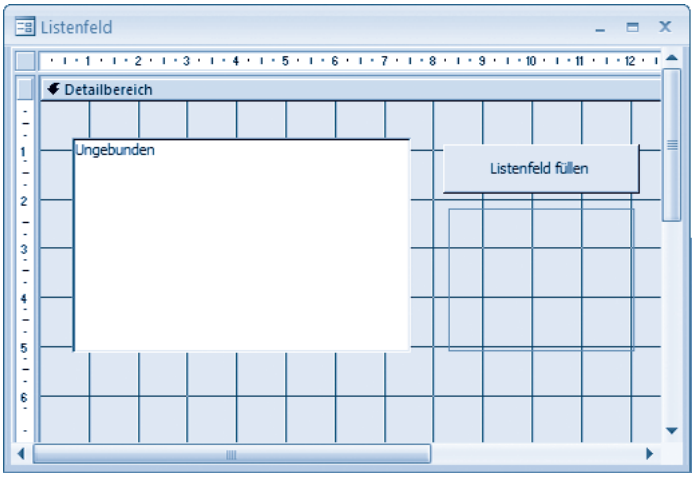


Abbildung 220: Die Ausgangssituation

262 Listenfeld füllen

Über die Methode `AddItem` können Sie einem Listenfeld einen Eintrag hinzufügen. Die Syntax lautet:

```
Ausdruck.AddItem(Item, Index)
```

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 115: Die Argumente der Methode AddItem

Im Beispiel aus *Listing 334* wird im Formular LISTENFELD der Schaltfläche eine Prozedur zugewiesen, welche die Namen sowie die Pfade aller Bilddateien aus einem Verzeichnis im Listenfeld einfügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl2_Click()
  'Bilddateien einlesen
```

Listing 334: Listenfeld mit Namen von Bilddateien füllen

```
  Dim strDatei As String
  Dim strSuchVerz As String

  strSuchVerz = "C:\Eigene Dateien\Bilder\"
  strDatei = Dir(strSuchVerz & "*.jpg")
  Do Until strDatei = ""
    Me!Liste0.AddItem strSuchVerz & strDatei
    strDatei = Dir
  Loop
End Sub
```

Listing 334: Listenfeld mit Namen von Bilddateien füllen (Forts.)

Zur Suche der Bilddateien verwenden Sie die Funktion `Dir`. Speichern Sie den Pfad des Verzeichnisses, in dem Sie suchen möchten, zunächst in einer String-Variablen `strSuchVerz`. Diesen Pfad benötigen Sie später noch. Übergeben Sie der Funktion `Dir` beim ersten Aufruf den Pfad des Verzeichnisses, in dem Sie suchen möchten, verknüpft mit einem Suchpattern in der Form `*.JPG`. In unserem Beispiel erhält die Funktion `Dir` genau genommen den String `C:\Eigene Dateien\Bilder\*.JPG` und sucht demnach nach allen Dateien mit der Endung `.JPG` im Verzeichnis `C:\Eigene Dateien\Bilder`.

Innerhalb einer Schleife rufen Sie wiederholt die Funktion `Dir` ohne Parameter auf, bis diese einen leeren String zurückliefert, was bedeutet, dass keine weitere Datei mehr gefunden wurde, die der Suchspezifikation entspricht.

Innerhalb der Schleife werden die einzelnen Dateinamen über die Methode `AddItem` ins Listenfeld eingefügt. Hier kommt die Variable `StrSuchVerz` zum Einsatz. Da Ihnen die Funktion `Dir` Dateinamen ohne Pfad zurückliefert, können Sie zusammen mit dem zuvor abgespeicherten Suchverzeichnis für jede gefundene Datei den kompletten Dateipfad konstruieren.

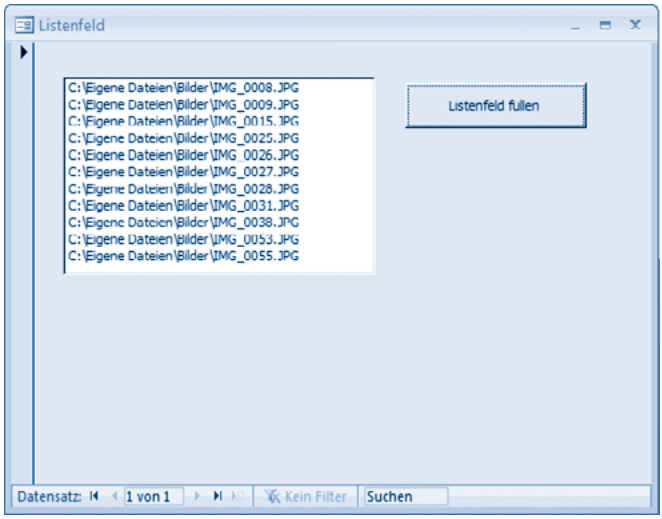


Abbildung 221: Das Listenfeld wurde gefüllt.

263 Listenfeldauswahl verarbeiten

Im nächsten Schritt soll bei der Auswahl eines Listenfeldeintrages das entsprechende Bild im Kleinformat in das Steuerelement BILD eingefügt werden. Die Auswahl eines Bildes wird über das Ereignis `Click` des Listenfeldes ausgelöst. Bevor Sie aber an die Programmierung des Ereignisses gehen, gehen Sie wie folgt vor, um das BILD-Steuerelement für diese Aufgabe vorzubereiten:

1. Klicken Sie das BILD-Steuerelement in der Entwurfsansicht mit der rechten Maustaste an und wählen 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 aus.
5. Beenden Sie diese Festlegung und speichern Sie die Änderungen.

Stellen Sie nun das `Click`-Ereignis des Listenfeldes wie in *Listing 335* gezeigt ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=======================================================

Private Sub Liste0_Click()
  'Ausgewähltes Bild übertragen
  Dim intIdx As Integer

  intIdx = Me!Liste0.ListIndex
  Bild4.Picture = Me!Liste0.ItemData(intIdx)
End Sub
```

Listing 335: 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 `intIdx`.

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 116: Die Argumente der Eigenschaft ItemData

Übergeben Sie den kompletten Pfad des Bildes aus dem Listenfeld direkt dem BILD-Steuerelement, indem Sie der Eigenschaft `Picture` diese Information geben.

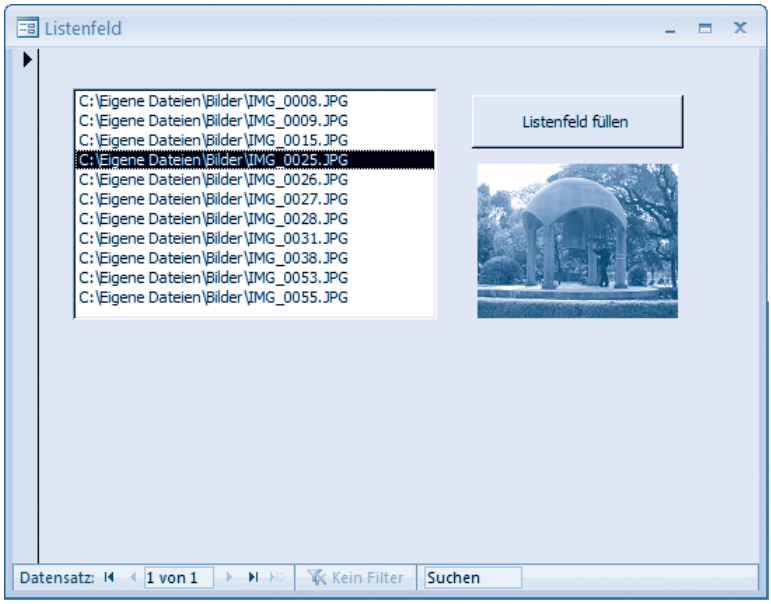


Abbildung 222: Bild nach Klick im Kleinformat anzeigen

264 Listenfeldeinträge zählen

Mit der `ListCount`-Eigenschaft können Sie feststellen, wie viele Zeilen ein Listenfeld oder der Listenfeldteil eines Kombinationsfeldes enthält.

Der erste Eintrag im Listenfeld fängt mit der 0 an.

265 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. In der nächsten Proze-

dur aus *Listing 336* wird jeweils der markierte Listeneintrag aus dem Listenfeld sowie die Grafikdatei auf der Festplatte entfernt.

```
'=================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=================================================

Private Sub Befehl5_Click()
  'Ausgewählten Eintrag löschen
  Dim intIdx As Integer

  On Error GoTo Fehler
  intIdx = Me!Liste0.ListIndex

  Kill Me!Liste0.ItemData(intz)
  Me!Liste0.RemoveItem (intIdx)
  Exit Sub

Fehler:
  MsgBox Err.Number & " " & Err.Description
End Su b
```

Listing 336: 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 `Remove Item`, 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 117: 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.

266 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 Kontextmenü, wechseln zur Registerkarte ANDERE und aktivieren im Feld MEHRFACHAUSWAHL den Eintrag ERWEITERT. Sie können dann durch gleichzeitiges Drücken der Strg-Taste und mit der Maustaste mehrere Einträge auswählen.

Bei der folgenden Lösung aus *Listing 337* wurde eine weitere Schaltfläche im Formular eingefügt und das `Click`-Ereignis eingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl6_Click()
  'Ausgewählte Einträge im Listenfeld löschen
  Dim intZeile As Integer
  Dim aint() As Integer
  Dim intZ As Integer

  If Me!Liste0.ItemsSelected.Count = 0 Then
    Exit Sub
  Else
    ReDim aint(Me!Liste0.ItemsSelected.Count)
    For intZeile = 0 To Me.Liste0.ListCount - 1
      If Me!Liste0.Selected(intZeile) Then
        aint(intZ) = intZeile
        intZ = intZ + 1
      End If
    Next
    For intZ = Me!Liste0.ItemsSelected.Count - 1 To 0 Step -1
      Me!Liste0.RemoveItem aint(intZ)
    Next
  End If
End Sub
```

Listing 337: Alle markierten Listenfeldeinträge entfernen.

Da in diesem Fall mehrere Einträge im Listenfeld markiert werden können, definieren Sie zu Beginn der Prozedur eine Array-Variable `aint`. 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 als 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 `avar` gefüllt, entfernen Sie die entsprechenden Listeneinträge in einer Schleife. Sie müssen dabei bei den höheren Indizes anfangen. Zum Entfernen verwenden Sie die Methode `RemoveItem`.

267 Alle Listenfeldeinträge aus-/abwählen

Mithilfe der `ItemsSelected`-Eigenschaft können Sie ermitteln, welche Einträge bei einer Mehrfachauswahl im Listenfeld markiert sind.

Im folgenden Beispiel aus *Listing 338* werden alle markierten Einträge im Listenfeld abgewählt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl7_Click()
  'Alle Einträge demarkieren
  Dim var As Variant
  Dim ctl As Control

  Set ctl = Me!Liste0
  For Each var In ctl.ItemsSelected
    ctl.Selected(var) = False
  Next var
End Sub
```

Listing 338: 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 339* zur Verfügung stellen, mit der man sehr schnell alle Listenfeldeinträge markieren kann.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Listenfeld
'=====================================================

Private Sub Befehl8_Click()
  'Alle Einträge markieren
```

Listing 339: Alle Listenfeldeinträge markieren

```
  Dim i As Integer

  For i = 0 To Liste0.ListCount
    Me!Liste0.Selected(i) = True
  Next i
End Sub
```

Listing 339: 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.

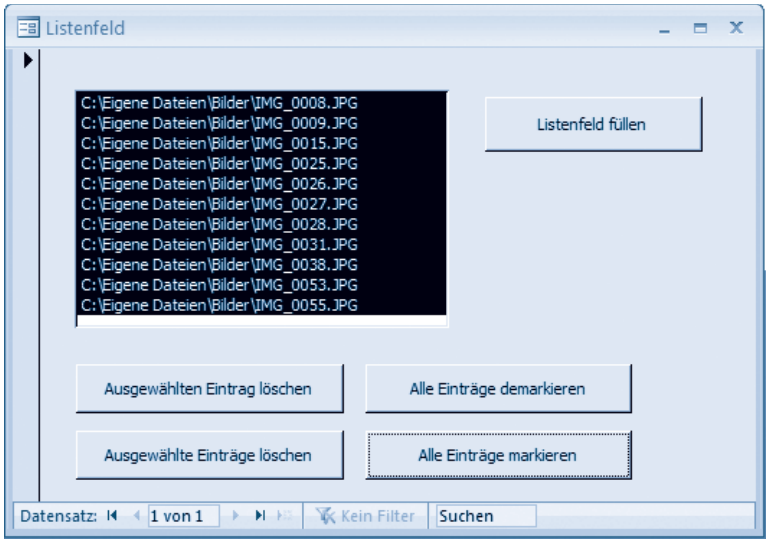


Abbildung 223: Alle Listenfeldeinträge blitzschnell markieren

268 Mehrspaltiges Listenfeld füllen

Beim folgenden Beispiel aus *Listing 340* werden die in der Datenbank vorhandenen Tabellen sowie Abfragen in ein mehrspaltiges Listenfeld eines Formulars eingelesen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_ListenfeldMehrspaltig
'=======================================================

Private Sub Befehl2_Click()
  'Tabellen und Abfragen in mehrspaltiges Listenfeld einlesen
```

Listing 340: Ein mehrspaltiges Listenfeld füllen

```
  Dim cat As New ADOX.Catalog
  Dim tbi As ADOX.Table

  cat.ActiveConnection = CurrentProject.Connection

  For Each tbi In cat.Tables
    With tbi
      Me!Liste0.AddItem .name & ";" & .Type
    End With
  Next

  Set cat = Nothing
End Sub
```

Listing 340: 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.

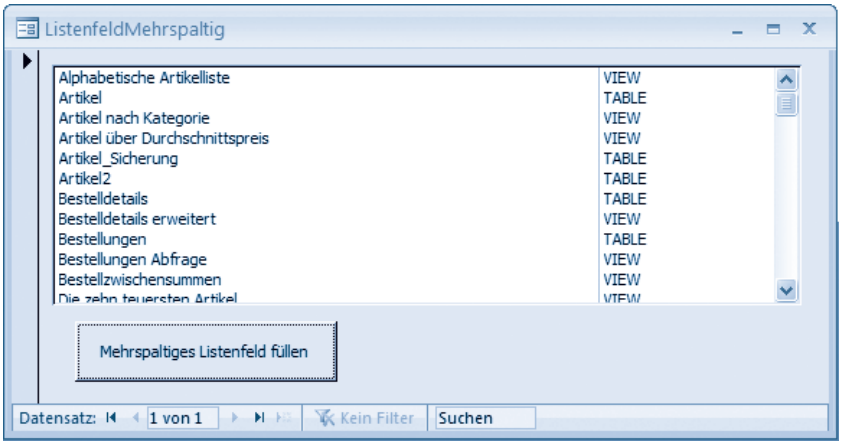


Abbildung 224: Das mehrspaltige Listenfeld wurde gefüllt.

269 Auf einzelne Spalten zugreifen

Wenn Sie das vorherige Formular noch um zwei Textfelder erweitern, dann können Sie das Beispiel ausbauen. Wenn ein Eintrag im Listenfeld angeklickt wird, dann sollen die einzelnen Spalteninhalte des Listenfeldes in die dafür vorgesehenen Textfelder übertragen werden.

Um diese Aufgabe zu lösen, erfassen Sie das `Click`-Ereignis in *Listing 341* für das Listenfeld.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 341: 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 118: Die Argumente der Eigenschaft Column

Über die Eigenschaft `ListIndex` haben Sie Zugriff auf den aktuell ausgewählten Eintrag im Listenfeld.

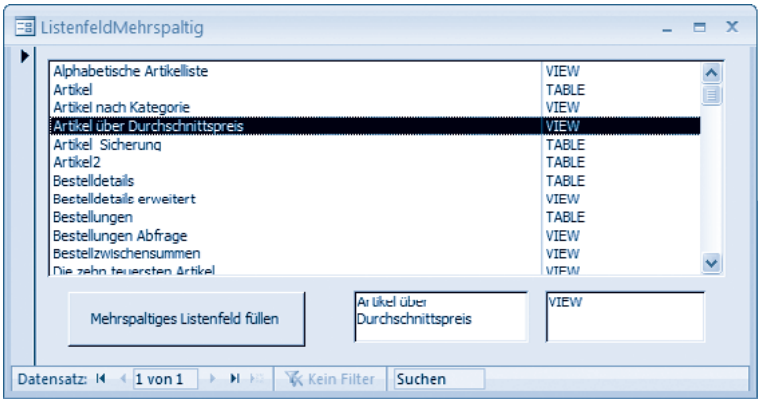


Abbildung 225: Die Spalteninhalte wurden in die Textboxen übertragen.

Kombinationsfelder

Ein Kombinationsfeld ist die platzsparendere Variante des Listenfeldes. Das Kombinationsfeld zeigt erst nach einem Klick auf den Pfeil seinen Listeninhalt an. In den meisten Fällen sind die Methoden und Eigenschaften dieselben wie auch beim normalen Listenfeld.

270 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 226* an.

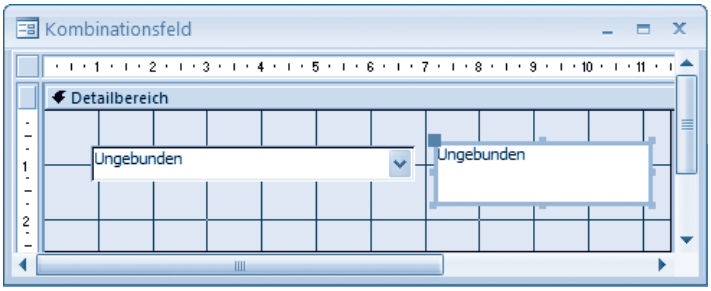


Abbildung 226: Die Monate müssen noch eingelesen werden.

Für das Füllen des Kombinationsfeldes können Sie das Ereignis `Form_Load` aus *Listing 342* nützen, das automatisch ausgeführt wird, wenn das Formular geöffnet wird. Der ideale Zeitpunkt, um die einzelnen Monate ins Kombinationsfeld zu bringen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld
'=======================================================

Private Sub Form_Load()
  With Me.Kombinationsfeld0
    .AddItem "Januar"
    .AddItem "Februar"
    .AddItem "März"
    .AddItem "April"
    .AddItem "Mai"
    .AddItem "Juni"
    .AddItem "Juli"
```

Listing 342: Ein Kombinationsfeld füllen – Variante 1

```
    .AddItem "August"
    .AddItem "September"
    .AddItem "Oktober"
    .AddItem "November"
    .AddItem "Dezember"
    .ListRows = 6
    .Value = MonthName(Month(Date))
  End With
End Sub
```

Listing 342: 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` anzapfen. Da im Kombinationsfeld aber keine Zahlen zwischen 1 und 12, sondern eben der Monatsname in ausgeschriebener Form, also Januar bis Dezember, stehen soll, übergeben Sie diese »Monatsnummer« an die Funktion `MonthName`, die daraus den ausgeschriebenen Monatsnamen macht.

Im zweiten Schritt füllen Sie das Textfeld, das den ersten und letzten Tag im Monat ausgibt. Erfassen Sie dazu die Funktion aus *Listing 343*, 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
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld
'=======================================================

'***In Standardmodul erfassen
Public Function LetzterTagImMonat(dtm As Date) As Date
  LetzterTagImMonat = DateSerial(Year(dtm), Month(dtm) + 1, 0)
End Function


'***Direkt hinter dem Formular erfassen
Private Sub Kombinationsfeld0_Change()
  Dim dtm As Date
  Dim strMonat As String
  Dim strJahr As String
```

Listing 343: Den letzten Tag im Monat ermitteln

```
  strMonat = Me.Kombinationsfeld0.Value
  strJahr = Year(Date)
  dtm = CDate(strMonat & " " & strJahr)
  dtm = LetzterTagImMonat(dtm)
  Text2.Value = "vom 01. " & strMonat & " " & _
    strJahr & " bis zum " & Format(dtm, "DD. MMMM YYYY")
End Sub
```

Listing 343: 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 Kombinationsfeldes. 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.

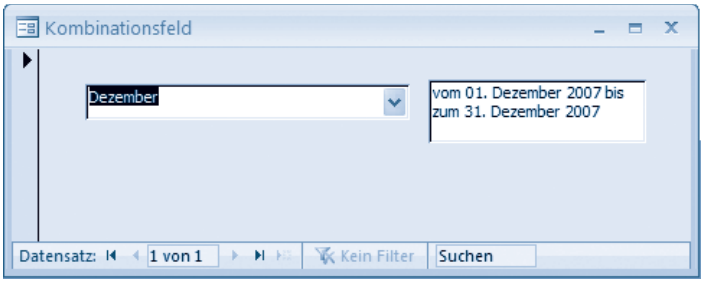


Abbildung 227: Der letzte Tag im Monat wurde ermittelt.

Im folgenden Beispiel aus *Listing 344* 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
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld2
'=====================================================

Private Sub Form_Load()
Dim intTage As Integer

For intTage = 1 To Day(LetzterTagImMonat(Date))
 With Me.Kombinationsfeld0
  .AddItem CDate((intTage) & " " & Month(Date) & " " & Year(Date))
 End With
Next intTage
End Sub
```

Listing 344: Die einzelnen Tage des aktuellen Monats im Kombinationsfeld anbieten

In einer Schleife lesen Sie über die Methode `AddItem` die einzelnen Tage des aktuellen Monats in das Kombinationsfeld ein. Dabei können Sie auch für diese Aufgabe die Funktion `LetzterTagImMonat` nützen, die Sie bereits in der Prozedur aus *Listing 343* aufgerufen haben. Setzen Sie das Datum aus den Einzelteilen (Tages-, Monats- und Jahresanteil) zusammen und wandeln diesen Text in ein gültiges Datum um, indem Sie die Funktion `CDate` einsetzen.

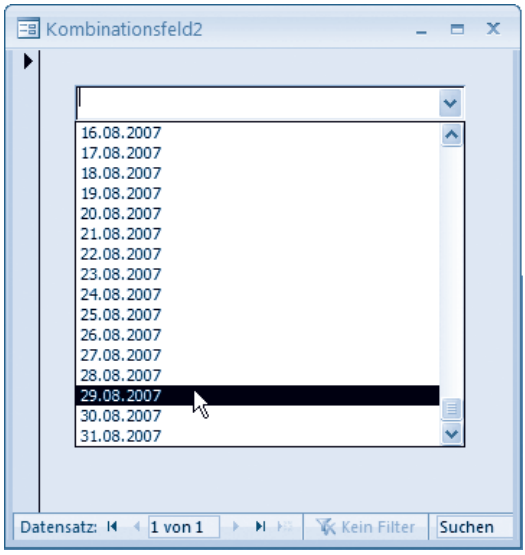


Abbildung 228: Die Tage des aktuellen Monats wurden eingefügt.

271 Kombinationsfeld füllen – Variante 2

Das folgende Beispiel aus *Listing 345* zeigt eine weitere Variante auf, ein Kombinationsfeld zu füllen. Dazu wurde ein neues Formular mit dem Namen KOMBINATIONSFELD3 erstellt, danach ein Kombinationsfeld sowie eine Schaltfläche integriert. Über einen Klick auf die Schaltfläche wird das Kombinationsfeld mit Werten gefüllt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld3
'=======================================================

Private Sub Befehl2_Click()
  Dim cbo As ComboBox
  Set cbo = Me!Kombinationsfeld0
```

Listing 345: Ein Kombinationsfeld füllen – Variante 2

```
  cbo.RowSourceType = "Value List"
  cbo.RowSource = "Sehr gut;Gut;Befriedigend;Ausreichend;" _
    & "Mangelhaft;Ungenügend"
  cbo.LimitToList = True
End Sub
```

Listing 345: Ein Kombinationsfeld füllen – Variante 2 (Forts.)

Setzen Sie die Eigenschaft `RowSourceType` 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 dem Kombinationsfeld Einträge aus einer Liste 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.

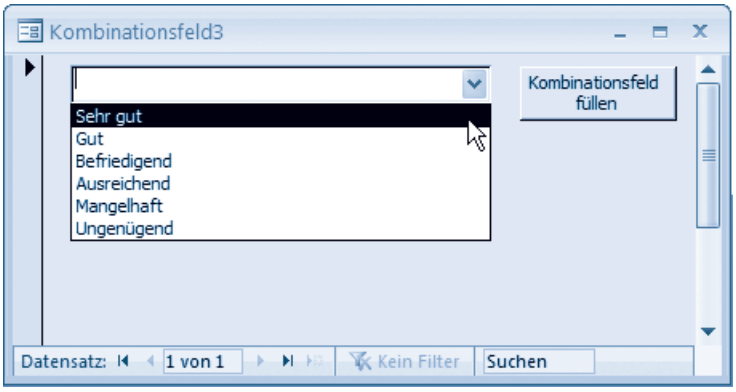


Abbildung 229: Die Alternative, ein Kombinationsfeld zu füllen

272 Editieren zulassen

Standardmäßig ist es bei Kombinationsfeldern so, dass Sie die Einträge, die ausgewählt werden können, definieren. Von Fall zu Fall mag es aber auch einmal sinnvoll sein, dem Kombinationsfeld zusätzliche Einträge, die bisher noch nicht eingestellt sind, hinzuzufügen.

Im nachfolgenden Beispiel aus *Listing 346* wurde ein neues Formular mit dem Namen KOMBINATIONSFELD4 erstellt sowie ein Kombinationsfeld und ein Textfeld eingefügt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=====================================================
```

Listing 346: Das Listenfeld wird beim Öffnen des Formulars gefüllt.

```
Private Sub Form_Load()
  Dim cbo As ComboBox

  Set cbo = Me!Kombinationsfeld5
  With cbo
    .RowSourceType = "Value List"
    .RowSource = "Spanien;Italien;Griechenland;Deutschland"
    .LimitToList = True
  End With
End Sub
```

Listing 346: 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 über die Eigenschaft `Row Source` mit den Einträgen zu füllen. Dabei stellen Sie zusätzlich bei der Eigenschaft `RowSourceType` den Text `Value List` ein. Über die Eigenschaft `LimitToList` legen Sie standardmäßig fest, dass keine zusätzlichen Einträge im Kombinationsfeld vorgenommen werden dürfen.

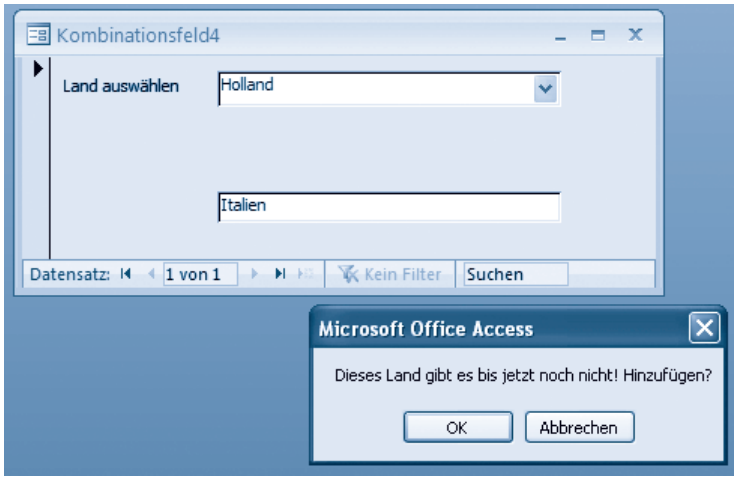


Abbildung 230: Neuen Eintrag zulassen oder verwerfen?

Über das Ereignis `NotInList` aus *Listing 347* 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
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=====================================================
```

Listing 347: Auf die Eingabe neuer Einträge im Kombinationsfeld reagieren

```
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 347: Auf die Eingabe neuer Einträge im Kombinationsfeld reagieren (Forts.)

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 Eintrages im Kombinationsfeld zugelassen. Danach werden über die Eigenschaft `RowSource` die bisherigen Einträge um den neuen erweitert. Dieser neue Eintrag steht automatisch im Argument `NewData` des Ereignisses `NotInList` bereit und kann so direkt übergeben werden.

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 231: Der neue Eintrag ist jetzt auch auswählbar.

Damit der ausgewählte Eintrag in das darunter liegende Textfeld übertragen wird, setzen Sie das Ereignis `AfterUpdate` aus *Listing 348* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Kombinationsfeld4
'=====================================================

Private Sub Kombinationsfeld5_AfterUpdate()
  Text3.Value = Kombinationsfeld5.Value
End Sub
```

Listing 348: Kombinationsfeld und Textfeld verknüpfen

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.

273 Kontrollkästchen auswerten

In der folgenden Prozedur aus *Listing 349* 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
' 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 349: Standardabfragen per Klick starten

```
  If Me!Kontrollkästchen2.Value = -1 Then DoCmd.OpenQuery Abfr01, acViewNormal
  If Me!Kontrollkästchen4.Value = -1 Then DoCmd.OpenQuery Abfr02, acViewNormal
  If Me!Kontrollkästchen6.Value = -1 Then DoCmd.OpenQuery Abfr03, acViewNormal
End Sub
```

Listing 349: Standardabfragen per Klick starten (Forts.)

Definieren Sie zu Beginn der Prozedur 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 unabhängig und damit flexibel zu gestalten.

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 Tab bestätigen.

Weisen die Kontrollkästchen den Wert `-1` auf, so sind diese aktiviert, andernfalls weisen sie den Wert `0` auf. Starten Sie im Aktivierungsfall die entsprechende Abfrage mithilfe der Methode `OpenQuery`.

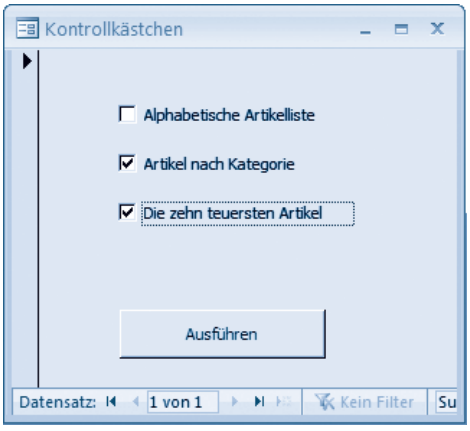


Abbildung 232: Kontrollkästchen auswerten

274 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 350* einstellen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Kontrollkästchen
'=====================================================

Private Sub Form_Load()
  Dim frm As Form
  Dim ctl As Object

  For Each ctl In Me.Controls
    If ctl.ControlType = acCheckBox Then ctl.Value = 0
  Next ctl
End Sub
```

Listing 350: Kontrollkästchen initialisieren

Indem Sie die Eigenschaft `Value` auf den Wert `0` setzen, werden die Kontrollkästchen initialisiert.

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 übliche Funktion einstellen, dass Sie jeweils nur eine Option auswählen können, müssen Sie die einzelnen Optionsschaltflächen in eine Optionsgruppe einfügen.

Beim 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 233*.

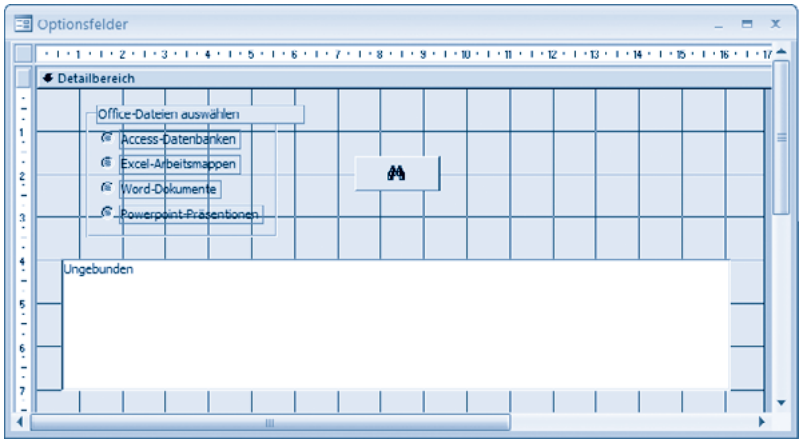


Abbildung 233: Die Optionsfelder werden in einem Gruppenfeld erstellt.

Legen Sie nun die Prozedur aus *Listing 351* hinter die Suchen-Schaltfläche.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Optionsfelder
'=====================================================

Private Sub Befehl13_Click()
  'Office-Dateien suchen
  Dim strDatei As String
  Dim strSuchVerz As String
  Dim strSuchPattern As String
  Dim intZ As Integer

  For intZ = Me!Liste11.ListCount - 1 To 0 Step -1
    Me!Liste11.RemoveItem (intZ)
  Next intZ

  strSuchVerz = "C:\Eigene Dateien\"

  Select Case Me.Rahmen0
    Case 1
      strSuchPattern = "*.mdb"
    Case 2
      strSuchPattern = "*.xls"
    Case 3
      strSuchPattern = "*.doc"
    Case 4
      strSuchPattern = "*.ppt"
    Case Else
      strSuchPattern = "*.*"
  End Select

  strDatei = Dir(strSuchVerz & strSuchPattern)

  intZ = 0
  Do Until (strDatei = "")
    intZ = intZ + 1
    Me!Liste11.AddItem strSuchVerz & strDatei
    strDatei = Dir
  Loop

  Me.Text15.Value = intZ & " Dateien gefunden!"
End Sub
```

Listing 351: Office-Dateien optional suchen

Im ersten Schritt entfernen Sie zunächst alle eventuell schon angezeigten Einträge im Listenfeld `Liste11`. Tun Sie dies nicht, wird bei mehrfacher Betätigung der Schaltfläche die Dateiliste immer länger und enthält mehrfach dieselben Einträge.

Über eine `Select Case`-Anweisung bestimmen Sie, welches Optionsfeld ausgewählt wurde. Beim Formularentwurf haben Sie über den Dialog EIGENSCHAFTEN den einzelnen Optionsfeldern die Werte `1` bis `4` zugewiesen. Dem Rahmen haben Sie den Namen `Rahmen0` gegeben.

In den einzelnen Zweigen der `Select Case`-Anweisung weisen Sie der Variable `strSuchPattern` ein Suchpattern zu, das die jeweilige Dateiendung des Dateityps spezifiziert. Wir beschränken uns in diesem Beispiel auf die Dateinamen-Erweiterungen der Office-Versionen vor Office 2007.

In diesem Beispiel suchen Sie im Verzeichnis, das über die Variable `strSuchVerz` spezifiziert wurde, in einer Schleife über den Einsatz der `Dir`-Funktion und tragen die Suchergebnisse in das Listenfeld ein. Sie setzen hierzu die Methode `AddItem` ein. Über die Variable `intZ` zählen Sie zusätzlich die gefundenen Dateien und geben das Ergebnis in einem Textfeld aus.

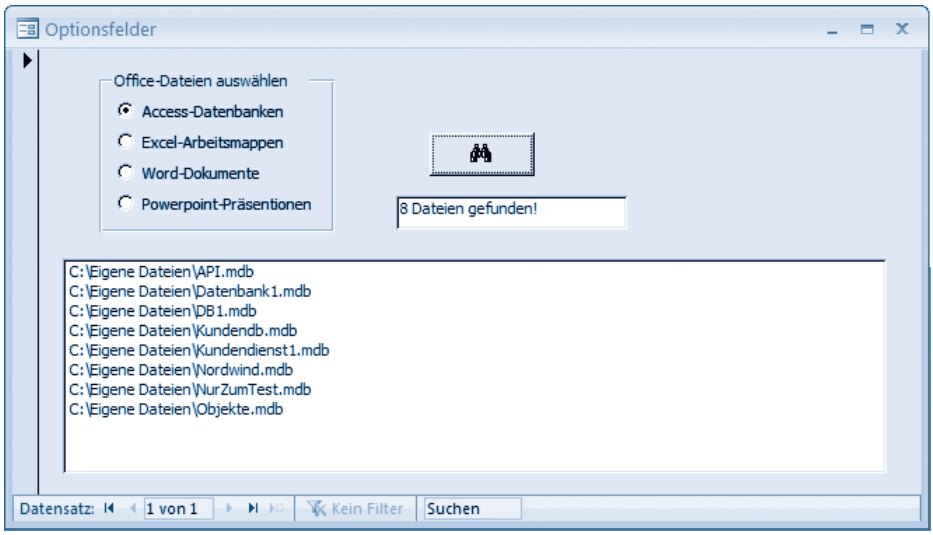


Abbildung 234: Es wurde nach Access-Datenbanken gesucht.

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 eine Prozedur für die Speicherung von Daten in eine Tabelle über ein Formular zu erstellen, verfahren Sie wie folgt:

1. Öffnen Sie das Formular ARTIKEL der Datenbank FORMULAR.MDB in der Entwurfsansicht.
2. Wählen Sie bei Access 2007 aus der Registerkarte ENTWURF aus der Gruppe STEUERELEMENTE das Element SCHALTFLÄCHE bzw. bei früheren Access-Versionen in der TOOLBOX das Symbol BEFEHLSSCHALTFLÄCHE. Ziehen Sie dann auf Ihrem Formular eine Schaltfläche auf.

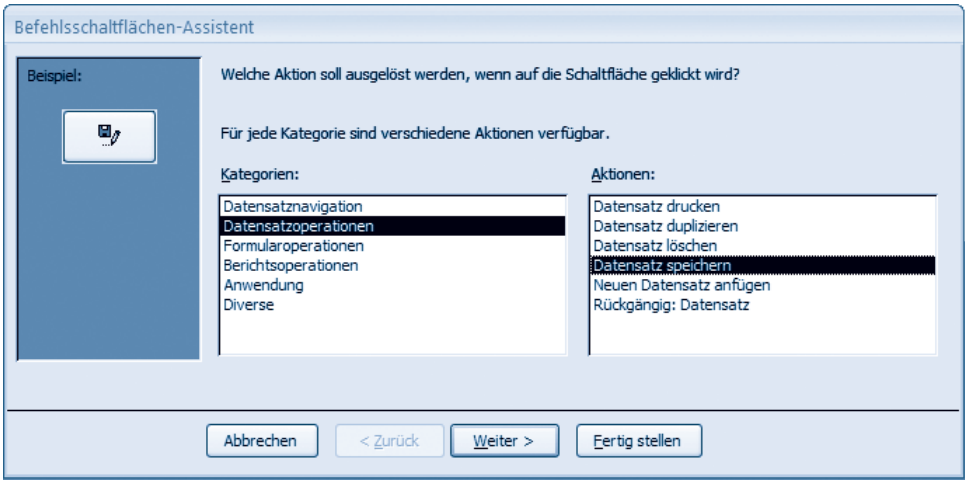


Abbildung 235: Schaltfläche einfügen

3. Der Befehlsschaltflächen-Assistent wird geöffnet. Markieren Sie im Feld KATEGORIEN den Eintrag DATENSATZOPERATIONEN.
4. Wählen Sie im Feld AKTIONEN den Eintrag DATENSATZ SPEICHERN.
5. Klicken Sie auf die Schaltfläche WEITER.

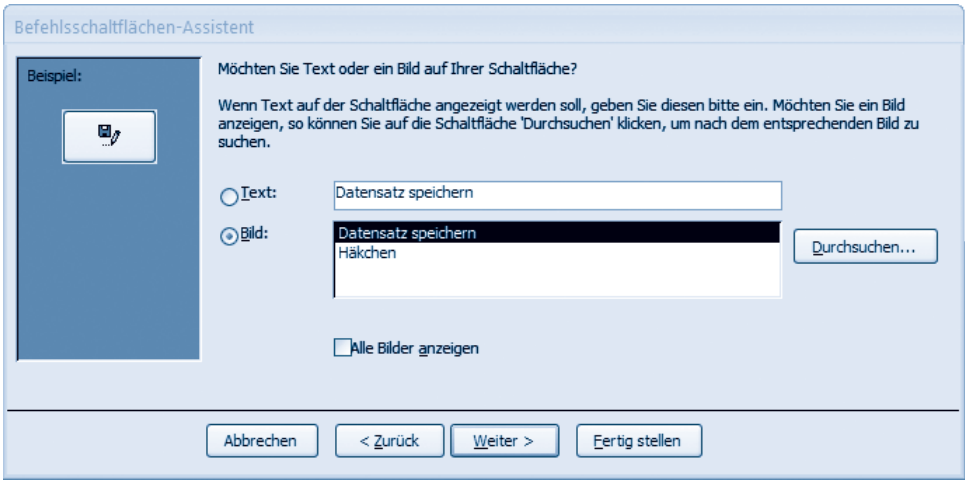


Abbildung 236: Symbol auswählen

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.

Die Schaltfläche wurde nun in Ihr Formular eingefügt. Klicken Sie mit der rechten Maustaste auf die neu eingefügte Schaltfläche und wählen aus dem Kontextmenü den Befehl EREIGNIS. Sie gelangen daraufhin automatisch in die Entwicklungsumgebung von Access und haben dann bei Access-Versionen vor Access 2007 folgende Prozedur aus *Listing 352* vor Augen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 352: Datensatz speichern über DoMenuItem

Dabei wird die Methode `DoMenuItem` eingesetzt. Diese Methode führt den angegebenen Menübefehl aus und 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 119: Die Befehlskonstanten der Methode DoMenuItem

Das Argument Unterbefehl wird nur dann verwendet, wenn Sie den Befehl Objekt aus dem Menü Einfügen einsetzen. Die dazu zur Verfügung stehenden Konstanten lauten acObjectVerb 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 Microsoft Access-Datenbanken der Version 2.0 eingegeben wurde.

Die Methode DoMenuItem wird zwar verwendet, wenn Sie den Befehlsschaltflächen-Assistenten einsetzen, um Schaltflächencode zu generieren. Diese Vorgehensweise können Sie 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
' 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 353: Datensatz speichern über RunCommand

Dies ist übrigens die Art und Weise, wie die Prozedur vom Befehlsschaltflächen-Assistenten bei Access 2007 von vorneherein generiert wird. Die Methode `RunCommand` hat nur noch ein Argument, das Sie in der Online-Hilfe zu Access-VBA komplett bzw. nur die wichtigsten im Anhang nachlesen können. Für das Speichern eines Datensatzes ist das in diesem Beispiel die Konstante `acCmdSaveRecord`.

275 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.

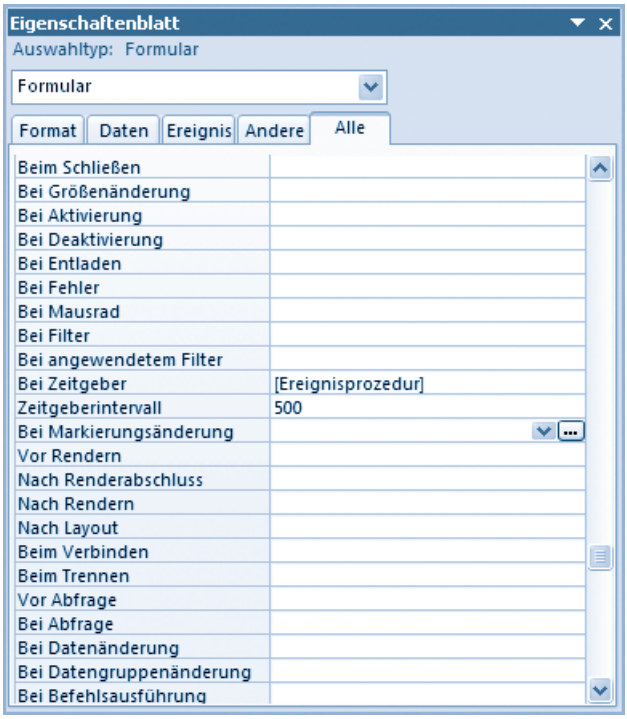


Abbildung 237: Das Zeitgeberintervall einstellen

4. Tragen Sie im Feld ZEITGEBERINTERVALL den Wert 500 ein. Die Einheit ist hier Millisekunden, d. h., die Schaltfläche wird somit zweimal in der Sekunde umgeschaltet.

5. Erstellen Sie jetzt das folgende Ereignis aus *Listing 354*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Form_Timer()
  If Me!Befehl5.Visible = True Then
    Me!Befehl5.Visible = False
  Else
    Me!Befehl5.Visible = True
  End If
End Sub
```

Listing 354: Eine blinkende Schaltfläche erstellen

Da Sie vorher das Intervall eingestellt haben, sorgen Sie in der Ereignisprozedur lediglich dafür, dass Sie die Schaltfläche im Wechsel ein- und ausblenden.

Weitere Steuerelemente

Neben den Standardsteuerelementen können Sie auf eine ganze Reihe weiterer Steuerelemente zugreifen. Klicken Sie hierzu in der Symbolleiste TOOLBOX auf das Symbol WEITERE STEUERELEMENTE.

Auf den nachfolgenden Seiten werden einige dieser zusätzlichen Steuerelemente anhand von praktischen Beispielen vorgestellt. Dabei können Sie anhand des Namens der einzelnen Formulare in der Beispieldatei FORMULARE.MDB schnell die gewünschte Lösung ausprobieren.

276 Kalender-Steuerelement

Standardmäßig ist das Kalender-Steuerelement nicht direkt in der TOOLBOX bzw. der Gruppe STEUERELEMENTE (Access 2007) aufgeführt. Sie können dieses Steuerelement aber trotzdem hinzufügen. Legen Sie dazu ein neues Formular an und klicken in der TOOLBOX auf das Symbol WEITERE STEUERELEMENTE. Daraufhin klappt ein Menü auf, in dem Sie das Steuerelement KALENDER-STEUERELEMENT 10.0 für Access 2002 bzw. das Steuerelement KALENDER-STEUERELEMENT 11.0 für Access 2003 auswählen können. Bei Access 2007 wählen Sie auf der Registerseite ENTWURF in der Gruppe STEUERELEMENTE das Symbol ACTIVEX-STEUERELEMENT EINFÜGEN. Dort wählen Sie das KALENDER-STEUERELEMENT 12.0. Ziehen Sie dann das Kalender-Element auf dem Formular KALENDER in der gewünschten Größe auf.

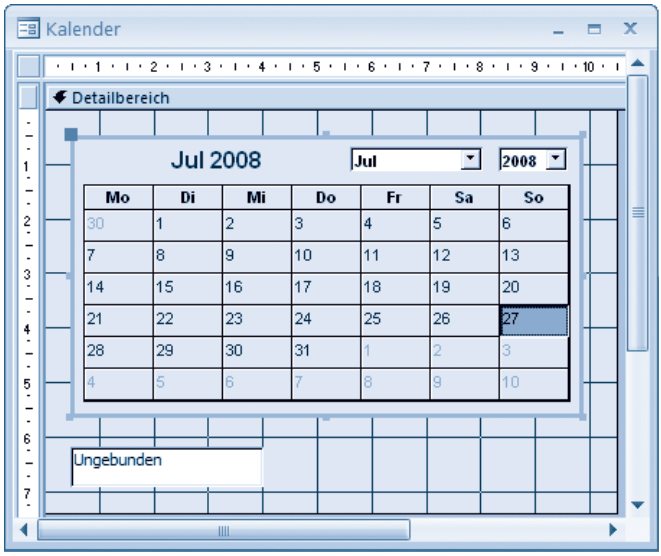


Abbildung 238: Das eingefügte Kalender-Steuerelement

Dieses Steuerelement 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 Kalender-Steuerelement und wählen den Befehl KALENDER-OBJEKT/EIGENSCHAFTEN.

In der *Tabelle 120* werden die wichtigsten Eigenschaften und Methoden des Kalender-Steuerelements 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.
`Day` (E)	Der aktuelle Monatstag im Kalender-Steuerelement kann eingestellt oder abgefragt werden.
`Month` (E)	Über diese Eigenschaft können Sie angeben, welcher Monat im Kalender-Steuerelement angezeigt werden soll.
`Year` (E)	Mit dieser Eigenschaft können Sie bestimmen, welches Jahr im Kalender-Steuerelement angezeigt wird. Dabei können Jahreszahlen zwischen 1900 und 2100 angegeben werden.
`ShowDateSelectors` (E)	Mit dieser Eigenschaft können Sie bestimmen, ob die Dropdown-Listenfelder für Monat und Jahr im Kalender-Steuerelement angezeigt werden.
`DayLength` (E)	Mit dieser Eigenschaft können Sie bestimmen, in welchem Format (kurz, mittel oder lang) die Wochentage im Kalender-Steuerelement 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 Kalender-Steuerelement 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 Kalender-Steuerelements 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 Kalender-Steuerelements um einen Tag erhöhen und das Kalender-Steuerelement aktualisieren.

Tabelle 120: Die wichtigsten Methoden und Eigenschaften für den Kalender

Eigenschaft (E)/ Methode (M)	Beschreibung
NextWeek (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um eine Woche erhöhen und das Kalender-Steuerelement aktualisieren.
NextMonth (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um einen Monat erhöhen und das Kalender-Steuerelement aktualisieren.
NextYear (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um ein Jahr erhöhen und das Kalender-Steuerelement aktualisieren.
PreviousDay (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um einen Tag vermindern und das Kalender-Steuerelement aktualisieren.
PreviousWeek (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um eine Woche vermindern und das Kalender-Steuerelement aktualisieren.
PreviousMonth (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um einen Monat vermindern und das Kalender-Steuerelement aktualisieren.
PreviousYear (M)	Mit dieser Methode können Sie den Wert eines Kalender-Steuerelements um ein Jahr vermindern und das Kalender-Steuerelement aktualisieren.
Today (M)	Mit dieser Methode können Sie den Wert des Kalender-Steuerelements auf das heutige Datum einstellen.

Tabelle 120: 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 355* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 355: Datum per Klick übernehmen

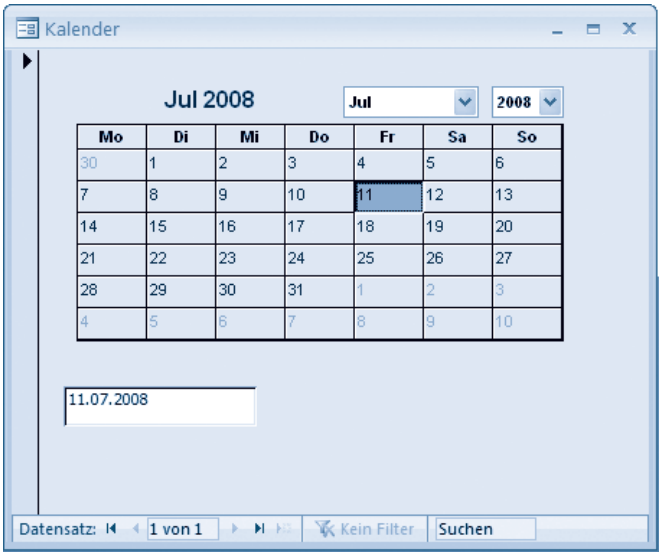


Abbildung 239: Das ausgewählte Datum wird ins Textfeld übernommen.

Im folgenden Beispiel aus *Listing 356* 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.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 356: 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 über die Funktion `Str` in einen String umwandeln. Damit vergleichen Sie dann den Datums-String aus dem Listenfeld mit einem Datums-String aus dem Kalender.

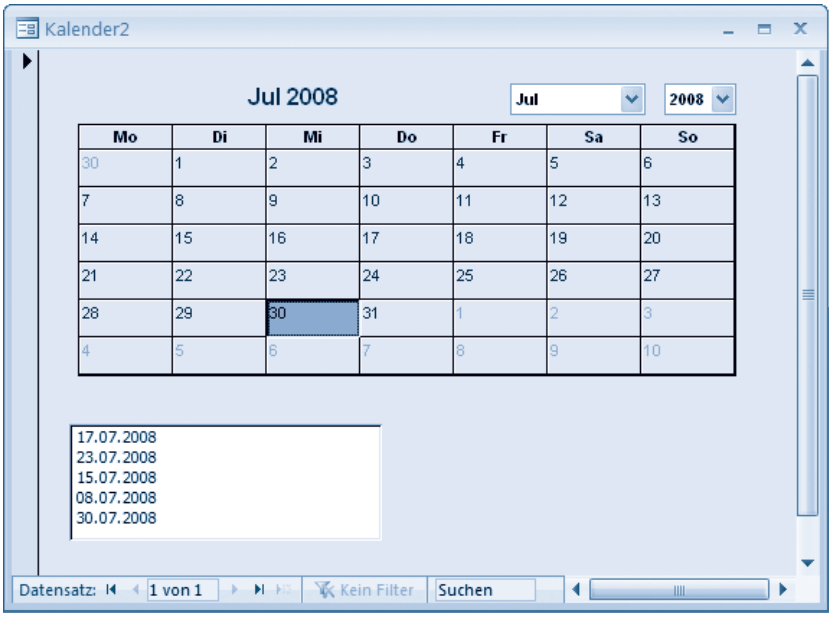


Abbildung 240: Es werden keine doppelten Tage übertragen.

277 Der Fortschrittsbalken

Werden über ein Formular längere Prozeduren abgearbeitet, sollten Sie sich überlegen, ob Sie nicht einen Fortschrittsbalken anbieten möchten. Bei länger laufenden Prozeduren ist es empfehlenswert, den Anwender während des Prozedurablaufs mit gelegentlichen Meldungen in der Statusleiste oder sogar einem Verlaufsbalken über die einzelnen Schritte bzw. den Verlauf der Prozedur in Kenntnis zu setzen.

Das Steuerelement MICROSOFT PROGRESSBAR CONTROL können Sie aktivieren, indem Sie das Symbol WEITERE STEUERELEMENTE in der Toolbox bzw. bei Access 2007 ACTIVEX-STEUERLEMENT EINFÜGEN in der Gruppe STEUERELEMENTE auswählen und anschließend das Steuerelement aus der Liste auswählen. Fügen Sie dieses Steuerelement und eine Schaltfläche in einem neuen Formular (FORTSCHRITTBALKEN) ein. Legen Sie die Prozedur aus *Listing 357* hinter die Schaltfläche.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Fortschrittbalken
'=====================================================

'Deklaration in Modul
Declare Sub Sleep Lib "Kernel32.dll" (ByVal SchlafZeit As Long)

Private Sub Befehl1_Click()
  'Fortschrittsbalken programmieren
  Dim i As Integer

  With Me
    .ProgressBar0.Min = 0
    .ProgressBar0.Max = 100

    For i = ProgressBar0.Min To .ProgressBar0.Max
      .ProgressBar0.Value = i
      Sleep 100
    Next i
    .ProgressBar0.Value = 0
  End With
End Sub
```

Listing 357: Den Fortschrittsbalken animieren

Mithilfe der Eigenschaften `Min` bzw. `Max` können Sie den Start- bzw. Endwert des Steuerelements bestimmen. `0` bis `100` sind dabei typische Werte. 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 einfach die über die `Sleep`-Funktion eingestellte Wartezeit pro Schleifendurchlauf. Bitte beachten Sie, dass Sie zur Verwendung der `Sleep`-Funktion diese in einem Modul deklarieren müssen. Den Fortschrittsbalken bekommen Sie automatisch zum Laufen, wenn Sie die Eigenschaft `Value` bei jedem Schleifendurchlauf mit der inkrementierten Schleifenvariablen `i` initialisieren.

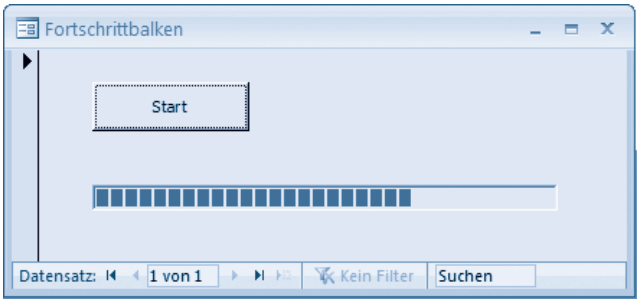


Abbildung 241: Das Steuerelement ProgressBar im Einsatz

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 den Befehl ProgCtrl-Objekt/Properties aus dem Kontextmenü.

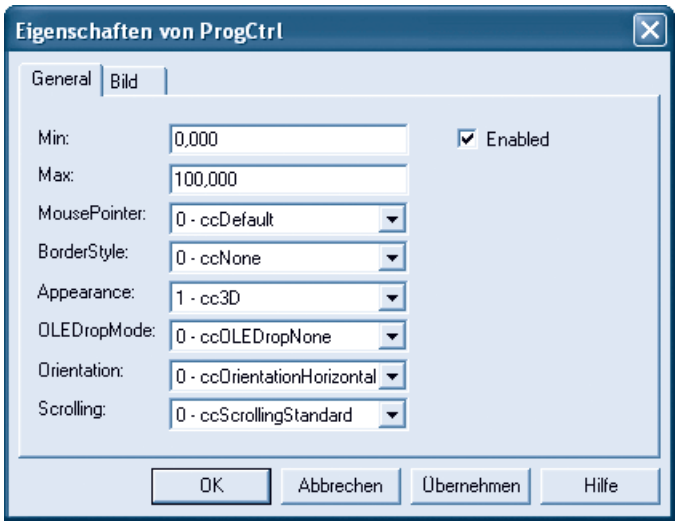


Abbildung 242: Der Fortschrittsbalken lässt sich einstellen.

278 Der Webbrowser

Über den Einsatz des Steuerelements Webbrowser können Sie Internetseiten oder auch Office-Dateien öffnen. Das Steuerelement Microsoft Web Browser können Sie aktivieren, indem Sie das Symbol Weitere Steuerelemente in der Toolbox bzw. bei Access 2007 ActiveX-Steuerelement einfügen in der Gruppe Steuerelemente auswählen. Anschließend wählen Sie das Steuerelement Microsoft Web Browser aus der Liste. Fügen Sie dieses Steuerelement und eine Schaltfläche in einem neuen Formular (Webbrowser) ein. Legen Sie die Prozedur aus *Listing 358* hinter die Schaltfläche.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 358: Webseiten, Office-Dateien oder Bilder über den Webbrowser betrachten

Mithilfe der Methode `Navigate` können Sie die im Argument `URL` angegebene Webseite oder Datei in den Webbrowser laden.

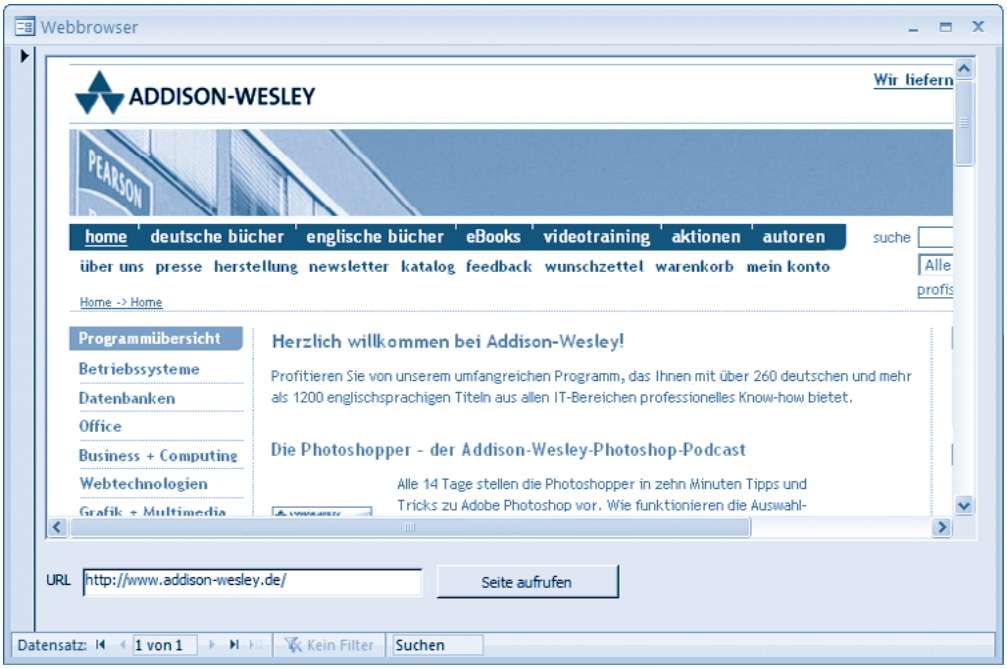


Abbildung 243: Eine Webseite im Webbrowser öffnen

279 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 eine zusätzliche Schaltfläche. Legen Sie danach die Prozedur aus *Listing 359* hinter die Schaltfläche.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_OutlookPost
'=====================================================

Private Sub Befehl1_Click()
  'E-Mail öffnen
  With Me
    .ViewCtl0.Open
  End With
End Sub
```

Listing 359: 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 das markierte Mail im Listenfeld geöffnet.

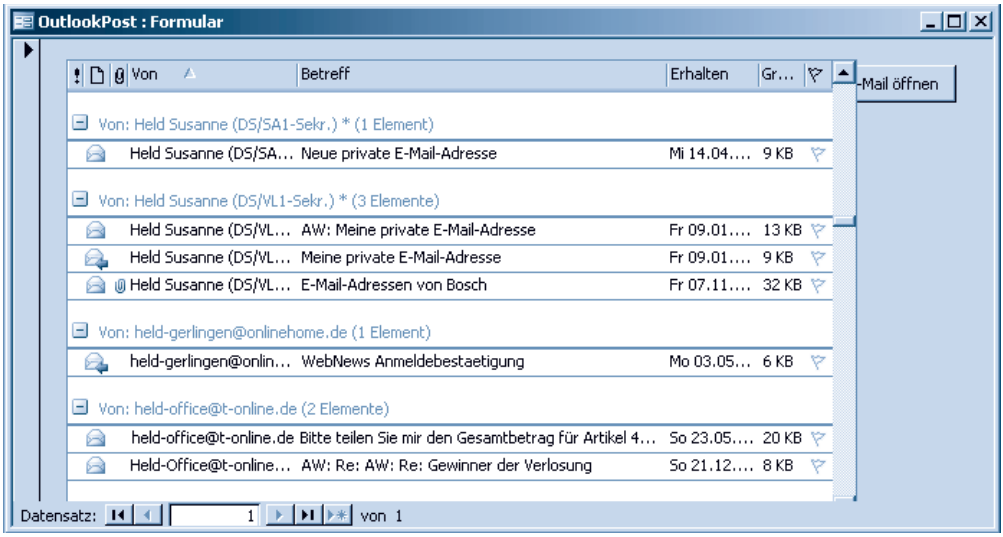


Abbildung 244: Alle Mails werden angezeigt.

280 Das TreeView-Steuerelement

Wie der Name dieses Steuerelementes aussagt, können Sie mithilfe dieses Steuerelements ganze Verzeichnisbäume auf einem Formular abbilden. Wählen Sie das TreeView-Steuerelement aus Ihrer Werkzeugsammlung aus und fügen es in einem neuen Formular (TreeView) ein. Integrieren Sie zusätzlich noch eine weitere Schaltfläche.

Da das TreeView-Steuerelement schon beim Aufruf des Formulars gefüllt sein muss, setzen Sie zu diesem Zweck auch hier das Ereignis `Form_Load` ein, das Sie im folgenden Listing sehen können. Dabei sollen alle Module der aktiven Datenbank im TreeView-Steuerelement angezeigt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_TreeView
'=====================================================

Private Sub Form_Load()
  Dim node  As node
```

Listing 360: Alle Module im TreeView-Steuerelement anzeigen

```
  Dim obj As AccessObject
  Dim dbs As Object

  Set dbs = Application.CurrentProject
  Set node = TreeView0.Nodes.Add(, , , dbs.Name)

  For Each obj In dbs.AllModules
    TreeView0.Nodes.Add node.Index, tvwChild, , obj.Name
  Next obj
  node.Expanded = True
End Sub
```

Listing 360: Alle Module im TreeView-Steuerelement anzeigen (Forts.)

Ermitteln Sie zuerst den Namen der aktiven Datenbank. Fügen Sie diesen Namen über die Methode `Add` als neuen Zweig im Steuerelement `TreeView` ein. Unterhalb dieses Zweiges ermitteln Sie die Namen der Module, die in der Datenbank enthalten sind. Fügen Sie diese Modulnamen wiederum über die Methode `Add` unterhalb des Hauptzweiges ein. Über die Zeile `node.Expanded = True` erreichen Sie, dass der oberste Knoten des Baumes expandiert dargestellt wird und Sie somit die angehängten Knoten auch wirklich gleich sehen.

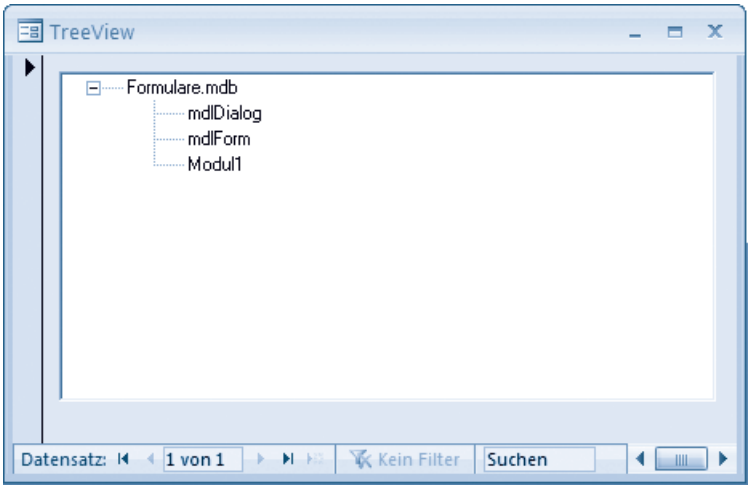


Abbildung 245: Module auflisten

281 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 das MEDIA PLAYER-STEUERELEMENT, ein Listenfeld und eine Schaltfläche ein. Legen Sie die Prozedur aus *Listing 361* hinter die Schaltfläche, um das Listenfeld mit Videodateien zu füllen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_MediaPlayer
'=====================================================

Private Sub Befehl3_Click()
  'Bilddateien einlesen
  Dim strDatei As String
  Dim strSuchVerz As String

  strSuchVerz = "C:\Eigene Dateien\"
  strDatei = Dir(strSuchVerz & "*.mpg")
  Do Until strDatei = ""
    Liste1.AddItem strSuchVerz & strDatei
    strDatei = Dir
  Loop
End Sub
```

Listing 361: Videodateien suchen

Führen Sie eine Suche nach den MPG-Dateien durch. Dazu verwenden Sie die Funktion `Dir`, um die einzelnen Dateien im angegebenen Verzeichnis zu ermitteln. Dabei wird über die Spezifikation `*.mpg` 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.

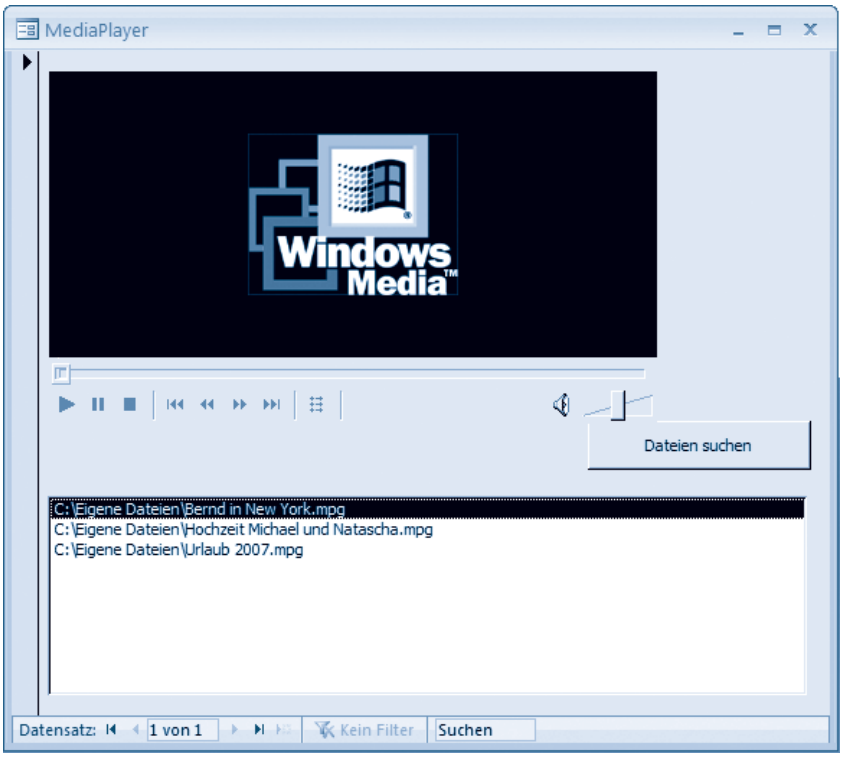


Abbildung 246: Filme im Formular abspielen

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
' Dateiname  Formulare.mdb
' Klasse     Form MediaPalyer
'=======================================================

Private Sub Liste1_DblClick(Cancel As Integer)
  MediaPlayer0.FileName = Liste1.Value
End Sub
```

Listing 362: Videodateien abspielen

Geben Sie über die Eigenschaft `FileName` bekannt, welche Datei der Media Player abspielen soll. Dabei lesen Sie den aktuellen Eintrag im Listenfeld über die Methode `Value` aus.

282 Diagramme in Formulare integrieren

Wenn Sie in Formularen mit Diagrammen arbeiten möchten, dann können Sie das Steuerelement MICROSOFT OFFICE CHART 10.0 (Access 2002) bzw. MICROSOFT OFFICE CHART 11.0 (Access 2003) benutzen.

Bei Access 2007 haben Sie das MICROSOFT OFFICE CHART-Steuerelement grundsätzlich nicht mehr zur Verfügung, sofern Sie nicht auch eine frühere Office-Version auf Ihrem PC installiert haben. Sie haben jedoch prinzipiell die Möglichkeit, sich die sogenannten MICROSOFT WEB COMPONENTS in der Version 11 von der Microsoft-Homepage herunterzuladen und auf Ihrem PC zu installieren. Dadurch haben Sie auch in Ihrer Access 2007-Version grundsätzlich die Möglichkeit, das hier vorgestellte Steuerelement einzusetzen. Ab Access 2007 empfiehlt sich jedoch, das im nächsten *Abschnitt 283* vorgestellte DIAGRAMM-Steuerelement einzusetzen.

Doch nun zunächst zum Steuerelement MICROSOFT OFFICE CHART. Die folgende Aufgabe besteht darin, die Tabelle ARTIKEL der Datenbank FORMULAR.MDB anzuzapfen und die fünf billigsten 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 363*.

```
'======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_OfficeChart
'======================================================

Private Sub Form_Load()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim obj As AccessObject
  Dim avarWert(4) As Variant
  Dim avarBez(4) As Variant
  Dim i As Integer
  Dim ChSp

  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  With rst
    .CursorLocation = adUseClient
    .Open "Artikel", conn, adOpenKeyset, _
      adLockOptimistic
    .Sort = "Einzelpreis Asc"
    For i = 0 To 4
      avarWert(i) = rst("Einzelpreis")
      avarBez(i) = rst("Artikelname")
```

Listing 363: Diagramm in Formular erstellen

```
      .MoveNext
    Next i
  End With

  'Diagrammerstellung
  Set ChSp = Me.ChartSpace0
  With ChSp.Charts(0)
  .Type = chChartTypeColumnClustered
    .SeriesCollection.Add
    .SeriesCollection(0).Caption = _
      "Die biligsten Artikel"
    .SeriesCollection(0).SetData chDimCategories, _
      chDataLiteral, avarBez
    .SeriesCollection(0).SetData chDimValues, _
      chDataLiteral, avarWert
    .HasLegend = True
    .HasTitle = True
  End With

  Set conn = Nothing
  Set rst = Nothing
End Sub
```

Listing 363: Diagramm in Formular erstellen (Forts.)

Beim 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 `avarWert` und `avarBez` geschrieben.

Haben Sie die billigsten 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 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 und auch als Legendentext verwendet werden kann.

Mithilfe der Methode `SetData` werden die Daten für das Diagramm zugewiesen. Diese Daten haben Sie vorher in den Variablen `avarWert` und `avarBez` gespeichert. Über die Eigenschaften `HasLegend` und `HasTitle` bestimmen Sie, dass eine Legende bzw. ein Diagrammtitel angezeigt werden soll.

Vergessen Sie dann nicht, die Tabelle über die Methode `Close` zu schließen und Objektverweise wieder aufzuheben.

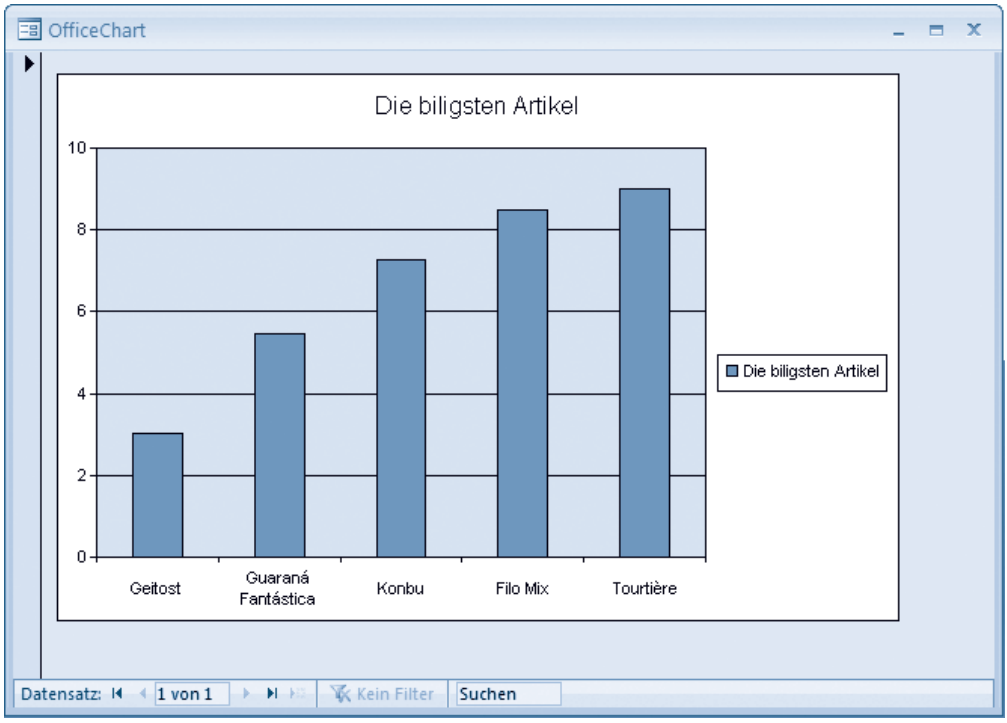


Abbildung 247: Die billigsten Artikel werden in einem Formulardiagramm angezeigt.

283 Diagramme in Formulare integrieren in Access 2007

Microsoft hat in Access-Version 2007 nun erstmals standardmäßig ein Diagramm-Steuerelement in die Hauptgruppe der Formular-Steuerelemente übernommen. Damit ist man nicht mehr gezwungen, Diagramm-Steuerelemente aus der Liste der weiteren Steuerelemente aus externen Bibliotheken zu verwenden. Generell kann ja die Verwendung von Objekten aus externen Bibliotheken, die Sie über Verweise einbinden, zu Problemen führen, wenn Sie Ihre Access-Datenbank weitergeben möchten und auf dem Ziel-PC die entsprechende Bibliothek nicht installiert ist. Dieses Problem umgehen Sie, wenn Sie nur die Steuerelemente einsetzen, die Access standardmäßig zur Verfügung stellt, wie in diesem Fall das neue Steuerelement DIAGRAMM.

Wie im vorigen Abschnitt zeigen wir in dieser Übung die fünf billigsten Artikel an, verknüpfen hier das Diagramm aber direkt mit einer Abfrage, sodass wir ausnahmsweise keine VBA-Prozedur erstellen.

Platzieren Sie in einem leeren Formular zunächst ein Steuerelement DIAGRAMM. Der Diagramm-Assistent wird gestartet. Wählen Sie auf der ersten Seite des Diagramm-Assistenten die Tabelle ARTIKEL und klicken Sie auf WEITER.

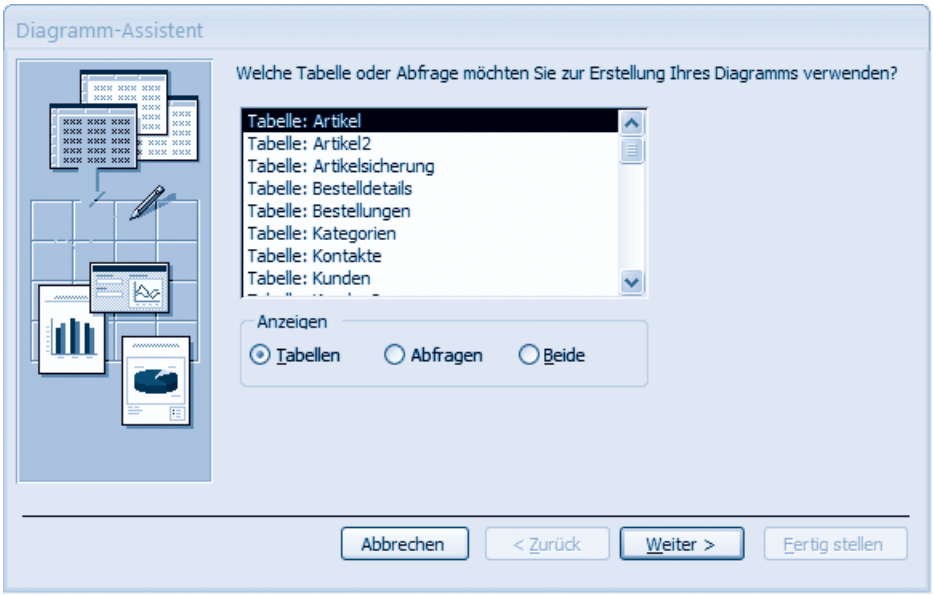


Abbildung 248: Tabelle oder Abfrage im Diagramm-Assistenten auswählen

Wählen Sie auf der zweiten Seite des Diagramm-Assistenten die Felder ARTIKELNAME und EINZELPREIS aus. Dies sind die Daten, die wir im Diagramm anzeigen möchten.

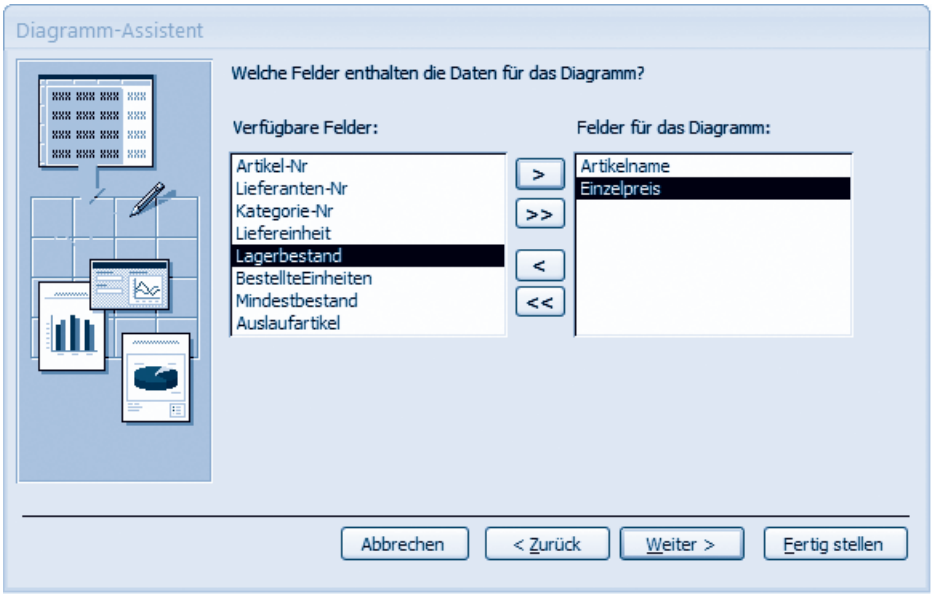


Abbildung 249: Darzustellende Felder im Diagramm-Assistenten auswählen

Auf der dritten Seite des Diagramm-Assistenten haben Sie die Möglichkeit, zwischen verschiedenen Diagrammtypen auszuwählen. In unserem Fall wählen wir den Diagrammtyp SÄULENDIAGRAMM.

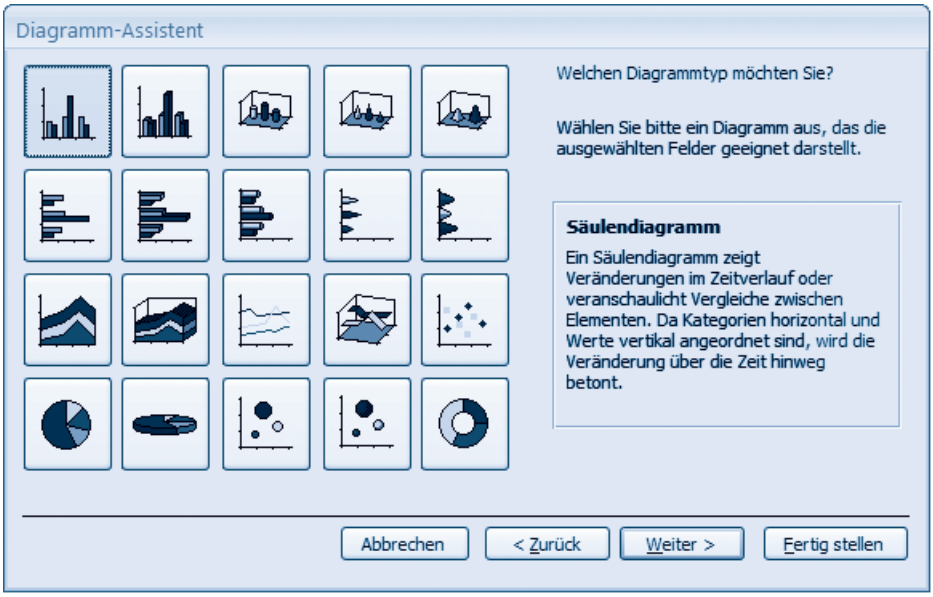


Abbildung 250: Der Diagramm-Assistent bietet eine Vielzahl von Diagrammtypen.

Auf der vierten Seite können Sie festlegen, welche der zuvor ausgewählten Felder wie im Diagramm dargestellt werden.

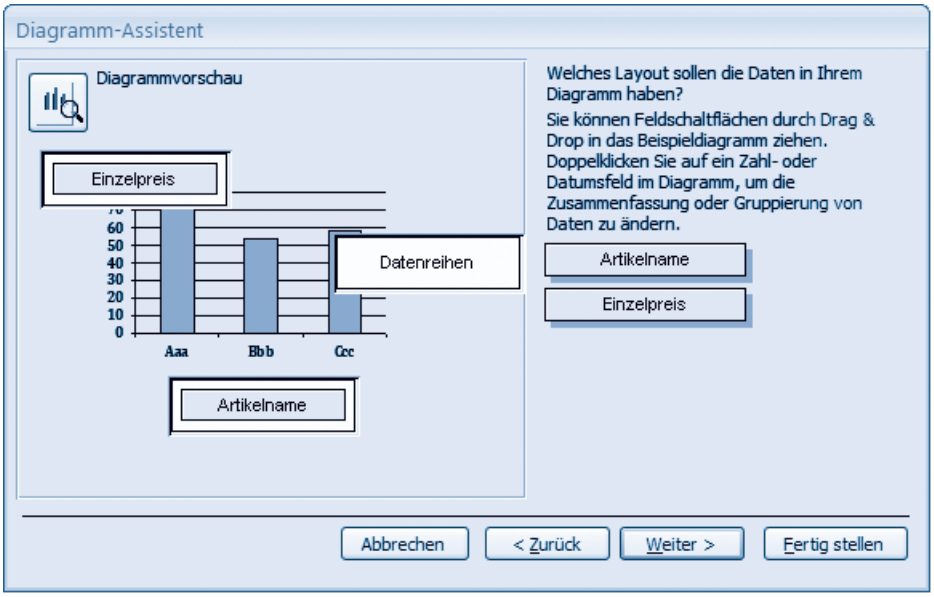


Abbildung 251: Zuordnung der Felder zu den Diagramm-Achsen

Auf der letzten Seite können Sie Ihrem Diagramm noch einen Titel geben und können festlegen, ob in Ihrem Diagramm eine Legende angezeigt werden soll. Geben Sie als Name »Die billigsten Artikel« ein und klicken Sie auf FERTIG STELLEN.

Wenn Sie sich nun Ihr Formular in der Formularansicht ansehen, werden Sie feststellen, dass das Diagramm bisher noch nicht das gewünschte Ergebnis zeigt. Es wird mehr oder weniger versucht, alle Artikel im Diagramm darzustellen. Wir wollen jedoch nur die fünf billigsten Artikel darstellen. Um dies zu erreichen, müssen wir noch eine kleine Modifikation vornehmen.

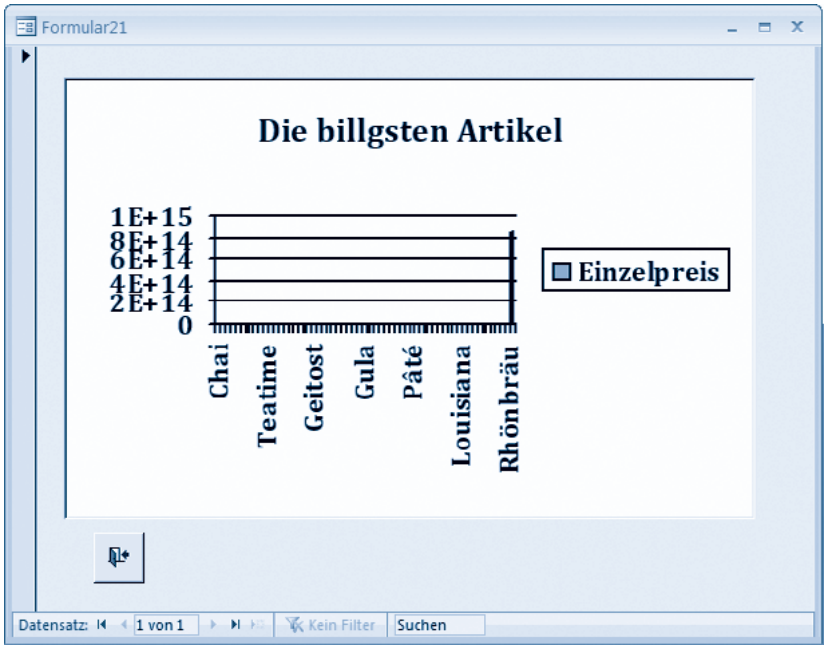


Abbildung 252: Das Formular zeigt noch nicht das gewünschte Ergebnis.

Lassen Sie sich Ihr Formular in der Entwurfsansicht anzeigen, klicken Sie mit der rechten Maustaste auf das Diagramm und wählen Sie im Kontextmenü den Menüpunkt EIGENSCHAFTEN aus.

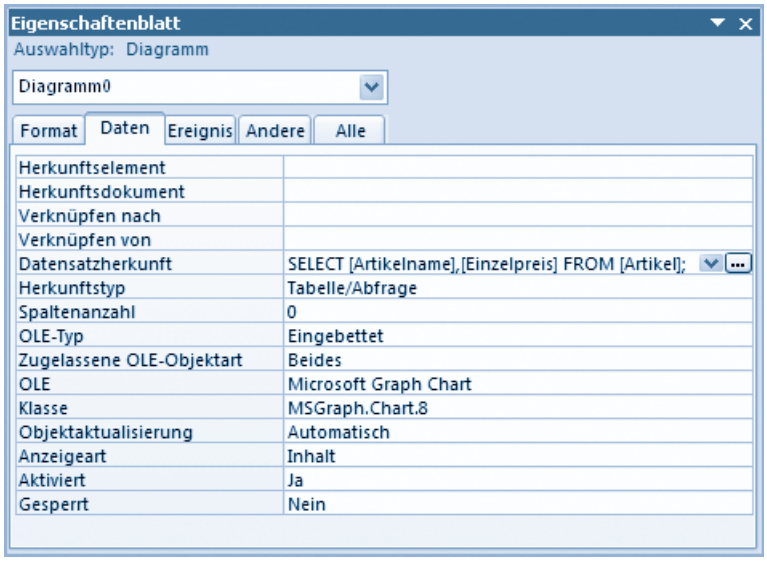


Abbildung 253: Das Feld Datensatzherkunft zeigt die SQL-Abfrage auf der das Diagramm basiert.

Sehen Sie sich auf dem dann angezeigten EIGENSCHAFTENBLATT auf der Registerkarte DATEN das Feld DATENSATZHERKUNFT an. Dort sehen Sie, auf welcher SQL-Abfrage das Diagramm basiert. Entsprechend der im Diagramm-Assistenten von uns zuvor festgelegten Auswahl lautet die SQL-Abfrage wie folgt:

```
SELECT [Artikelname],[Einzelpreis] FROM [Artikel]
```

Wir möchten jedoch nur die billigsten Artikel anzeigen. Hierzu müssen wir die SQL-Abfrage so modifizieren, dass das Suchergebnis der Abfrage nach Einzelpreis sortiert wird und die Abfrage nur die Top 5 des Gesamtsuchergebnisses enthält. Die modifizierte SQL-Abfrage müsste dann folgendermaßen aussehen.

```
SELECT TOP 5 [Artikelname],[Einzelpreis] FROM [Artikel] ORDER BY [Einzelpreis]
```

Um die Änderung einfacher vornehmen zu können, können Sie sich den Inhalt des Feldes DATENSATZHERKUNFT in der ZOOM-Darstellung anzeigen lassen. Positionieren Sie hierzu den Cursor im Feld DATENSATZHERKUNFT und drücken Sie die Tastenkombination UMSCHALT + F2. Anschließend öffnet sich der Dialog ZOOM und zeigt Ihnen den Inhalt des zuvor ausgewählten Datenfeldes, in unserem Fall den Inhalt des Feldes DATENSATZHERKUNFT.

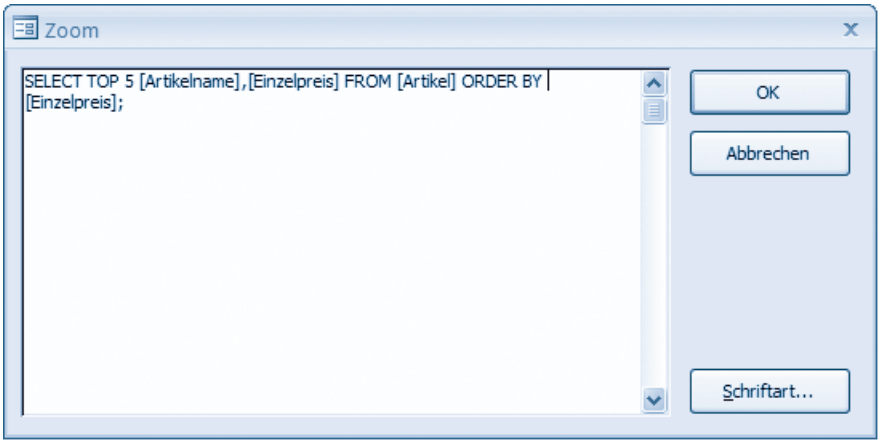


Abbildung 254: Die modifizierte SQL-Abfrage im Dialog Zoom

Nehmen Sie die Modifikation der SQL-Abfrage vor und schließen Sie den Dialog mit OK. Anschließend bekommen Sie im Diagramm, wie gewünscht, nur die fünf billigsten Artikel angezeigt.

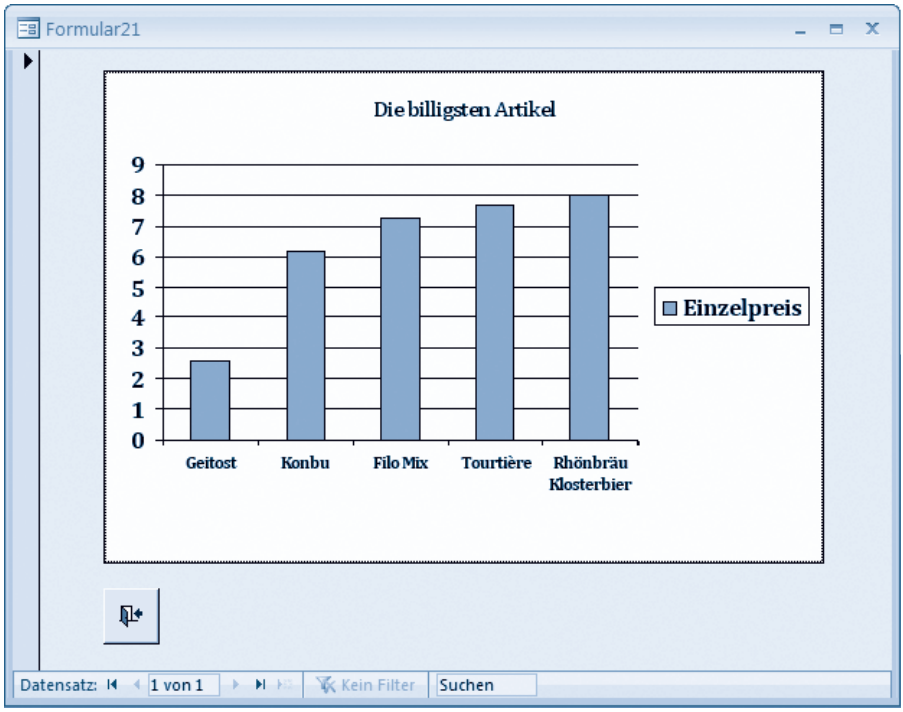


Abbildung 255: Die billigsten Artikel über das Access 2007-Diagramm-Steuerelement

284 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 wurde ein neues Formular mit dem Namen SPREADSHEET angelegt, das Steuerelement MICROSOFT OFFICE SPREADSHEET 11.0 (Access 2003), sowie ein Listenfeld und eine Schaltfläche eingefügt.

Bei Access 2007 haben Sie das MICROSOFT OFFICE SPREADSHEET-Steuerelement grundsätzlich nicht mehr zur Verfügung, sofern Sie nicht auch eine frühere Office-Version auf Ihrem PC installiert haben. Sie haben jedoch die Möglichkeit, sich die sogenannten MICROSOFT WEB COMPONENTS in der Version 11 von der Microsoft-Homepage herunterzuladen und auf Ihrem PC zu installieren. Dadurch haben Sie auch in Ihrer Access 2007-Version die Möglichkeit, das hier vorgestellte Steuerelement einzusetzen. Leider bietet Access zurzeit kein alternatives Steuerelement, das ähnlich tolle Features bereitstellt wie das Steuerelement MICROSOFT OFFICE SPREADSHEET. Daher möchte ich es Ihnen jetzt vorstellen.

Nun zum Beispiel: 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 die Prozedur aus *Listing 364* direkt hinter die Schaltfläche.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_SpreadSheet
'=====================================================

Private Sub Befehl5_Click()
  'Bilddateien einlesen
  Dim strDatei As String
  Dim strSuchVerz As String

  strSuchVerz = "C:\Eigene Dateien\"
  strDatei = Dir(strSuchVerz & "*.xls")
  Do Until strDatei = ""
    Liste1.AddItem strSuchVerz & strDatei
    strDatei = Dir
  Loop
End Sub
```

Listing 364: Excel-Arbeitsmappen einlesen

Im Beispiel führen Sie in einem vorgegebenen Verzeichnis eine Suche nach den Excel-Arbeitsmappen mit der Endung .XLS durch. Sie verwenden dazu die Funktion `Dir` in einer Schleife, wie schon vielfach in diesem Buch vorgestellt. Über die Methode `AddItem` fügen Sie innerhalb der Schleife die Suchergebnisse ins Listenfeld ein.

Der Doppelklick auf das Listenfeld aus *Listing 365* öffnet die ausgewählte Arbeitsmappe im Hintergrund und überträgt den Inhalt der ersten Tabelle ins SPREADSHEET-Steuerelement.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_SpreadSheet
'=====================================================

Private Sub Liste1_DblClick(Cancel As Integer)
  'Tabelle in Spreadsheet-Element übertragen
  Dim intZeilen As Long
  Dim intSpalten As Integer
  Dim objXlApp As Object
  Dim objXlMap As Object
  Dim intZ As Integer
```

Listing 365: Tabelle Zelle für Zelle übertragen

```
  Set objXlApp = CreateObject("Excel.Application")
  Set objXlMap = objXlApp.Workbooks.Open(Me.Liste1.Value)

  Me.Spreadsheet0.Cells.ClearContents
  For intZeilen = 1 To objXlMap.Sheets(1).UsedRange.Rows.Count
    For intSpalten = 1 To objXlMap.Sheets(1).UsedRange.Columns.Count
      Me.Spreadsheet0.Cells(intZeilen, intSpalten) = _
        objXlMap.Sheets(1).Cells(intZeilen, intSpalten)
    Next intSpalten
  Next intZeilen
  objXlMap.Close savechanges:=False
  objXlApp.Quit
  Set objXlApp = Nothing
  Set objXlMap = Nothing
End Sub
```

Listing 365: Tabelle Zelle für Zelle übertragen (Forts.)

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 der Prozedur 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 den entsprechenden Objektvariablen den Wert `Nothing` zuweisen.

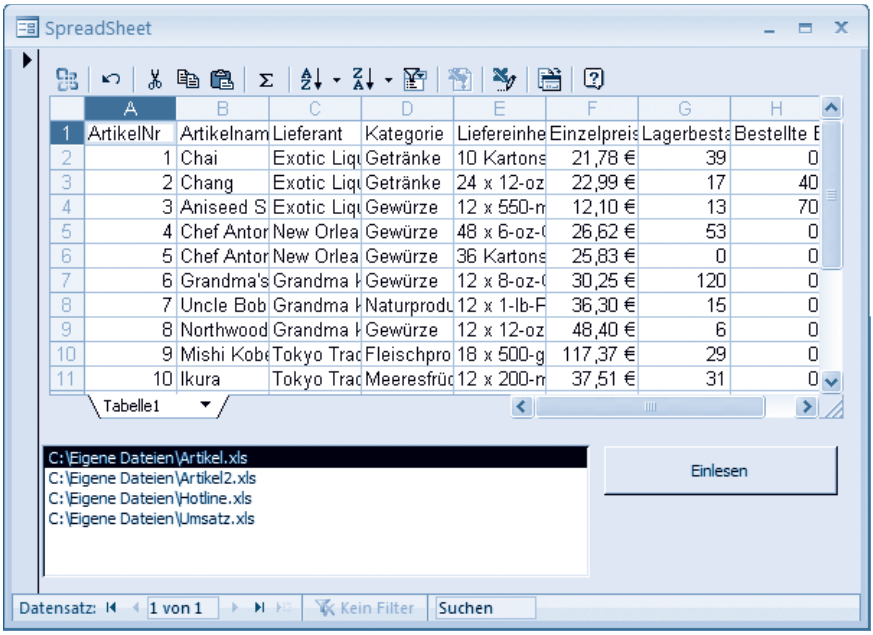


Abbildung 256: Die Excel-Tabelle wird im Access-Formular angezeigt.

285 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, in dem man den eingestellten Wert des SLIDERS anzeigt.

Das Steuerelement MICROSOFT SLIDER CONTROL finden Sie bei Access-Versionen vor Access 2007 unter WEITERE STEUERELEMENTE in der Toolbox bzw. bei Access 2007 in der Gruppe STEUERELEMENTE unter ACTICEX-STEUERELEMENT EINFÜGEN. Wählen Sie dort das entsprechende Slider-Steuerelement aus der Liste aus. 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.

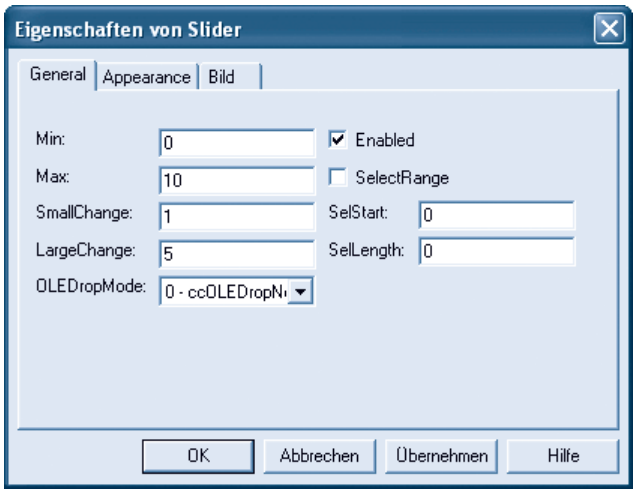


Abbildung 257: Das Slider-Steuerelement konfigurieren

Im Dialog aus *Abbildung 257* lässt sich das Slider-Steuerelement einstellen. Um nun das Textfeld mit dem SLIDER zu synchronisieren, erfassen Sie das Ereignis `Scroll` des SLIDERS.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Slider
'=======================================================

Private Sub Slider0_Scroll()
  'Textfeld mit Slider synchronisieren
  Text1.Value = Slider0.Value
End Sub
```

Listing 366: Übertragen des Slider-Status ins Textfeld

Über die Methode `Value` fragen Sie den aktuellen Status des SLIDERS ab und übertragen ihn in das Textfeld.

Abbildung 258: Slider & Text

286 Das Spinbutton-Steuerelement

Über das SPINBUTTON-Steuerelement können Sie elegant über zwei Drehknöpfe Werte in einem bestimmten Wertebereich in einem verknüpften Textfeld einstellen. 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 367* den kleinsten sowie größten Wert, den Sie über den SPINBUTTON einstellen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Formulare.mdb
' Klasse     Form_Spinbutton
'=====================================================

Private Sub Form_Load()
  With Me
   .SpinButton3.Max = 100
   .SpinButton3.Min = 1
  End With
End Sub
```

Listing 367: Die Grenzwerte des Spinbuttons festlegen

Über die Eigenschaft `Max` legen Sie den größtmöglichen Wert des Spinbuttons fest. Die Eigenschaft `Min` liefert den kleinsten Wert.

Um den aktuellen Wert des Spinbuttons ins Textfeld zu übertragen, erfassen Sie die Prozedur aus *Listing 368*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 368: Übertragen des Spinbutton-Wertes ins Textfeld

Über die Methode `Value` fragen Sie den aktuellen Status des Spinbuttons ab und übertragen ihn in das Textfeld.

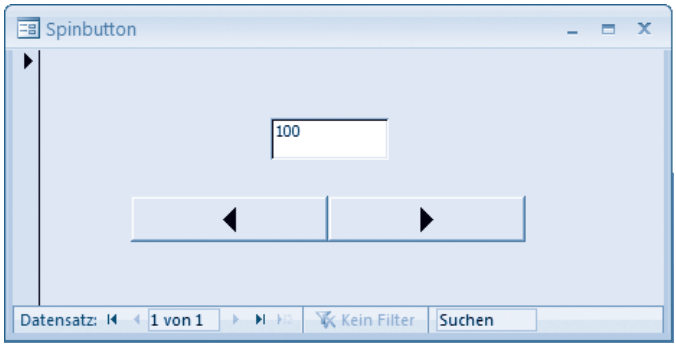


Abbildung 259: Dieser Spinbutton kann Werte von 1 bis 100 einstellen.

Berichte & 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.

Selbstverständlich können Sie Datenbanken nicht nur lokal betreiben, sondern auch im Internet sogenannte Datenzugriffsseiten ablegen, und diese dann einsetzen, um Ihren lokalen Datenbestand abzufragen bzw. zu aktualisieren.

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 durchzuführen.

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.

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.

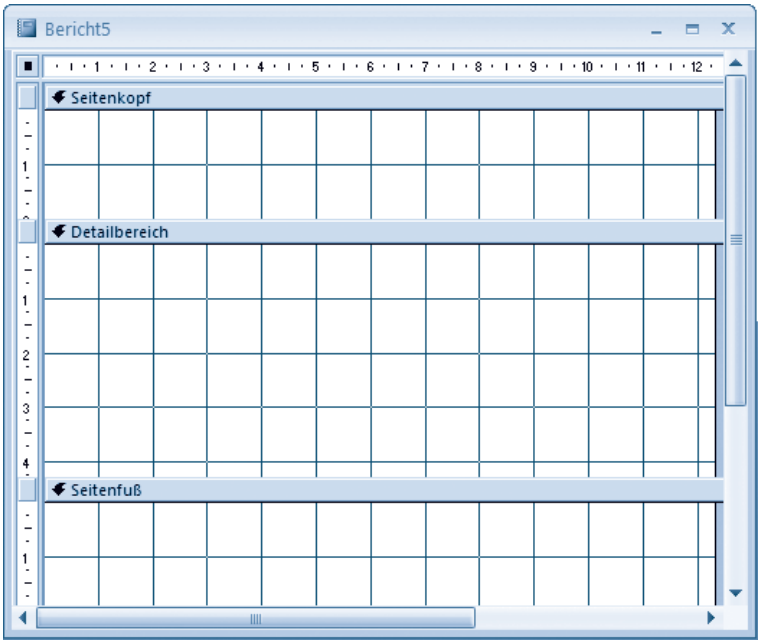


Abbildung 260: Die Grundbestandteile eines Berichts

Es gibt noch weitere Bestandteile eines Berichts, die bei Bedarf verwendet werden können. So können Sie zu Beginn einen Berichts ein Titelblatt definieren, das als Berichtskopf bezeichnet wird. Dieser Berichtskopf wird dann nur einmal 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.

Berichte mit DoCmd steuern

Bei einfacheren Aktionen wie das Öffnen, Suchen, Drucken und Schließen von Berichten können Sie mit dem Objekt `DoCmd` arbeiten. Sie können die Methoden des `DoCmd`-Objekts verwenden, um Microsoft Access-Aktionen aus Visual Basic heraus auszuführen.

287 Berichte öffnen

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 weiterarbeiten, wenn Sie den Bericht wieder geschlossen haben.
`OpenArgs`	Optional. Beim Öffnen des Berichts können bestimmte Werte bzw. Einstellungen mitgegeben werden.

Tabelle 121: Die Argumente der Methode OpenReport

Beim folgenden Beispiel aus *Listing 369* wird der Bericht UMSÄTZE NACH KATEGORIE in der Vorschau angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtÖffnen()
  On Error GoTo Fehler
  DoCmd.OpenReport "Umsätze nach Kategorie", acViewPreview
  DoCmd.Maximize
  Exit Sub
```

Listing 369: Bericht in der Vorschau öffnen

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

Listing 369: 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.

288 Berichtsfilter setzen

Im nächsten Beispiel aus *Listing 370* 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
' 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 370: Bericht aufrufen und Filter setzen

Stellen Sie als Filterkriterium eine Lieferantenauswahl ein. Da der Feldname 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.

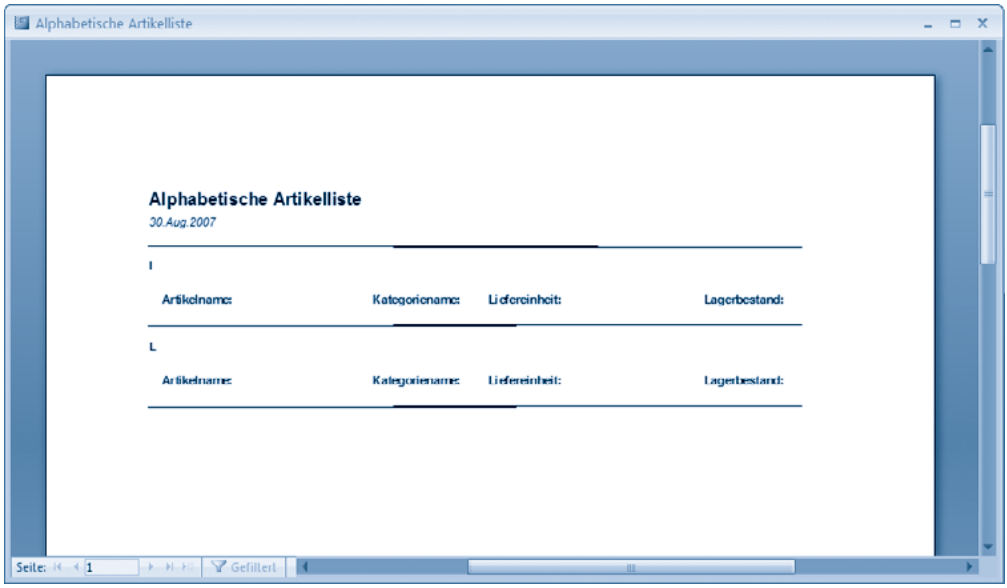


Abbildung 261: Nur bestimmte Datensätze anzeigen

289 Bericht schließen

Über die Methode `Close` können Sie einen Bericht schließen. Die Syntax dieser Methode lautet:

```
Close(Objekttyp, Objektname, Speichern)
```

Argument	Beschreibung
`Objekttype`	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 `Objekttyp` 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 122: Die Argumente der Methode Close

290 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, dass 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 123: Die Argumente der Methode PrintOut

Im folgenden Beispiel aus *Listing 371* 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
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtDrucken()
  DoCmd.OpenReport "Umsätze nach Kategorie", acViewPreview
  DoCmd.PrintOut acPages, 1, 2, acMedium, 5, True
End Sub
```

Listing 371: 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 372* werden die ersten beiden Seiten des Berichts KATALOG in der schnelleren Laser-Variante ausgedruckt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 372: Den noch schnelleren Laserdruck einstellen

291 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 die 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 124: Die Argumente der Methode CopyObject

Im folgenden Beispiel aus *Listing 373* wird der Bericht KATALOG kopiert und unter dem Namen KATALOGNEU gespeichert

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 373: Einen Bericht in der aktuellen Datenbank kopieren

Soll der Bericht in eine andere Datenbank kopiert werden, dann sehen Sie sich die Prozedur aus *Listing 374* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtKopierenErw()
  On Error GoTo Fehler
```

Listing 374: Bericht in eine andere Datenbank kopieren

```
  DoCmd.CopyObject "C:\Eigene Dateien\Nordwind.mdb", "Artikelkatalog", _
    acReport, "Katalog"
  MsgBox "Kopieraktion durchgeführt!"
  Exit Sub

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

Listing 374: Bericht in eine andere Datenbank kopieren (Forts.)

292 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 bei der Methode `CopyObject` beschrieben.
`OldName`	Optional. Beinhaltet den alten Namen des Objekts.

Tabelle 125: Die Argumente der Methode Rename

Im folgenden Beispiel aus *Listing 375* wird der Bericht KATALOGNEU umbenannt. Der Name des neuen Berichts lautet dann ARTIKELANGEBOT.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 375: Bericht umbenennen

293 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 inkl. aller Funktionen und Prozeduren ▶ `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.
`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 die die Daten transferiert werden sollen. Dabei muss die Datei noch nicht vorhanden sein. Access legt diese bei Bedarf selbst 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 126: Die Argumente der Methode OutputTo

Im folgenden Beispiel aus *Listing 376* wird der Bericht RECHNUNG in eine RTF-Tabelle transferiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDoCmd
'=====================================================

Sub BerichtAlsRtfExportieren()
  DoCmd.OutputTo acOutputReport, "Rechnung", acFormatRTF, _
    CurrentProject.Path & "C:\Eigene Dateien\Rechnung.rtf"
End Sub
```

Listing 376: Bericht exportieren

Sonstige Lösungen für Berichte

Auf den nächsten Seiten sehen Sie weitere Lösungen für Berichte, die über den Einsatz des Objektes `DoCmd` hinausgehen.

294 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 377* wird dem Bericht UMSÄTZE NACH JAHR der zusätzliche Text KOPIE hinzugefügt.

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

Listing 377: Ein zusätzliches Label einfügen – Variante 1

```
    .FontBold = True
    .ForeColor = RGB(256, 0, 0)
    .CurrentX = rpt.ScaleWidth / 2
  End With
  rpt.Print strText
End Sub
```

Listing 377: Ein zusätzliches Label einfügen – Variante 1 (Forts.)

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 bestimmen Sie, dass das Label etwa in der Mitte des Berichts eingefügt werden soll.

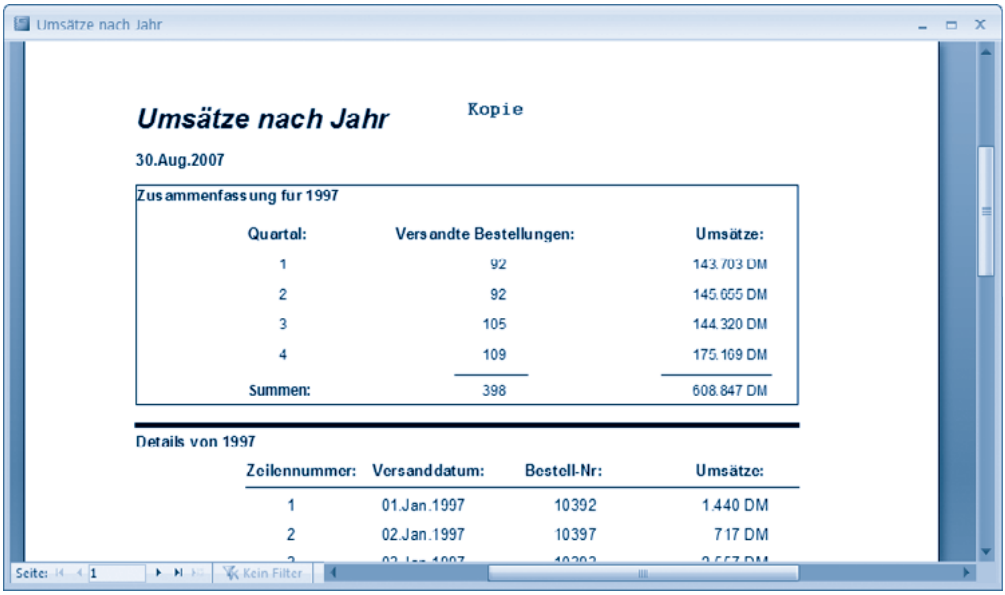


Abbildung 262: Das zusätzliche Label wurde eingefügt.

Weitere Eigenschaften für Formatierungen können Sie der *Tabelle 127* entnehmen.

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 127: Die wichtigsten Eigenschaften rund um die Formatierung von Berichten

In der Prozedur aus *Listing 377* 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 128* sehen können.

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 128: Die Standardfarben über die Funktion BQColor

Im folgenden Beispiel wird das Beispiel aus *Listing 378* über den Einsatz der Funktion `QBColor` gelöst.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 378: 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 Hellblau und die folgende Zeile mit der Farbe Grün formatiert werden. Dazu setzen Sie das Ereignis `Format` des Detailbereichs wie in *Listing 379* ein.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Klasse     Report_Katalog Unterformular
'=====================================================

Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)
  Static i As Integer
  i = i + 1
  If i Mod 2 = 0 Then
    Me.Detailbereich.BackColor = QBColor(2)
  Else
    Me.Detailbereich.BackColor = QBColor(9)
  End If
End Sub
```

Listing 379: Bericht im Wechsel einfärben

Die statische Variable zu Beginn des Ereignisses wird jeweils um den Wert `1` erhöht wenn das Ereignis ausgeführt wird, d. h., wenn also eine Formatierung vorgenommen wurde. Innerhalb der Schleife wird über die Funktion `Mod` geprüft, ob die Division ein gerades oder ein ungerades Ergebnis liefert. In beiden Fällen wird mithilfe der Eigenschaft `BackColor` der Hintergrund des Berichts angesprochen. Über den Einsatz der Funktion `RGB` wird dann jeweils die gewünschte Farbe festgelegt.

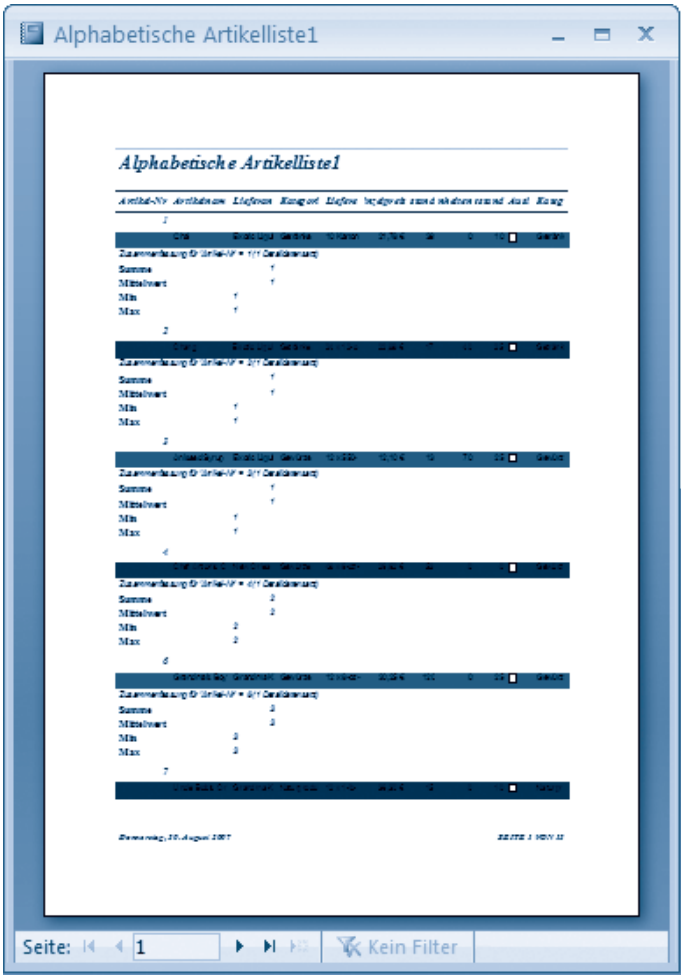


Abbildung 263: Der etwas andere »Look« im Bericht

295 Berichte listen

Mit der `AllReports`-Eigenschaft können Sie auf die `AllReports`-Auflistung und deren dazugehörige Eigenschaften zugreifen.

Im folgenden Beispiel aus *Listing 380* werden die Namen aller Berichte der Datenbank im Direktfenster der Entwicklungsumgebung ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 380: 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.

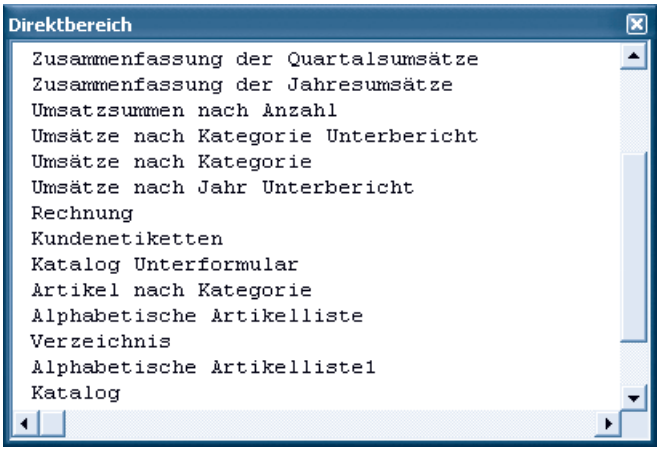


Abbildung 264: Die Berichte der aktuellen Datenbank wurden dokumentiert.

296 Berichtselemente identifizieren

Gehen Sie nun noch einen Schritt weiter und identifizieren Sie alle Elemente eines bestimmten Berichts. In der Prozedur aus *Listing 381* öffnen Sie den Bericht UMSÄTZE NACH KATEGORIE in der Entwurfsansicht, lesen dann alle verwendeten Steuerelemente aus und geben diese im Direktbereich aus. Schließen Sie danach den Bericht mithilfe der Methode `Close`.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=======================================================

Sub AlleElementeVonBerichtAuflisten()
  Dim obj As Object
  Dim rpt As Report

  DoCmd.OpenReport "Umsätze nach Kategorie", acViewDesign
  Set rpt = Reports("Umsätze nach Kategorie")
  For Each obj In rpt
    Debug.Print obj.Name
  Next obj
  DoCmd.Close
End Sub
```

Listing 381: 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.

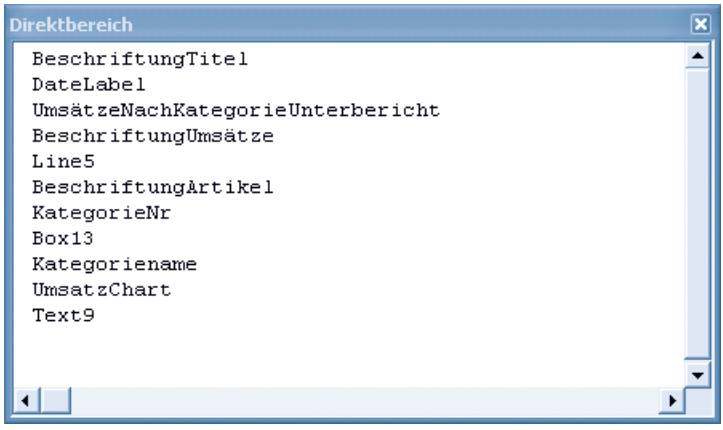


Abbildung 265: Die Berichtselemente wurden aufgelistet.

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 129: Die Steuerelementkonstanten der Eigenschaft ControlType

Im Beispiel aus *Listing 382* werden die Steuerelemente des Berichts UMSÄTZE NACH KATEGORIE identifiziert.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub FormularSteuerelementNamen()
  Dim obj As Object
  Dim strE As String
  Dim rpt As Report

  DoCmd.OpenReport "Umsätze nach Kategorie", acViewDesign
  Set rpt = Reports("Umsätze nach Kategorie")
  For Each obj In rpt.Controls
```

Listing 382: Berichtselemente identifizieren

```
    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 382: 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.

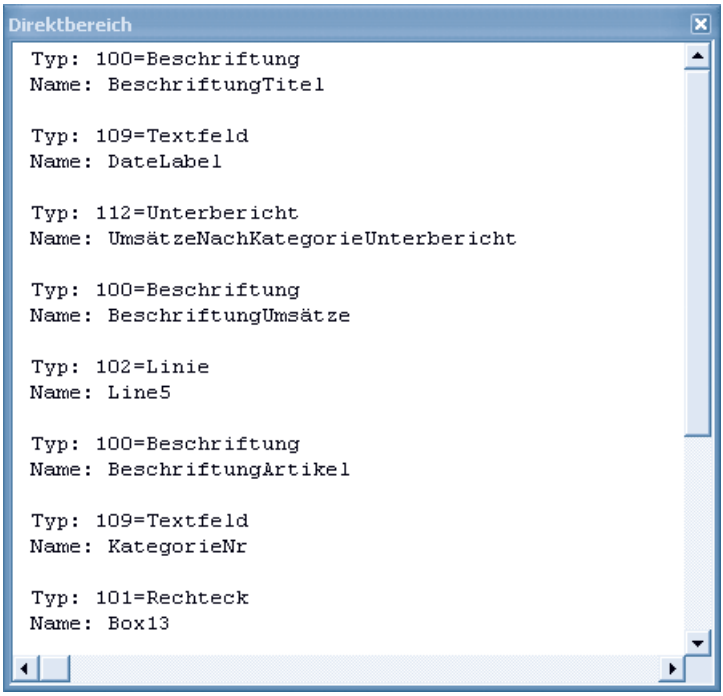


Abbildung 266: Die einzelnen Berichtselemente wurden identifiziert.

297 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 die dabei möglichen Werte aus der folgenden Tabelle.

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 130: Die Berichtssegmente

Das folgende Beispiel aus *Listing 383* blendet beispielsweise den Detailbereich beim Öffnen des Formulars aus. Dabei soll nur die Struktur eines Berichts, ohne Daten, angezeigt werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Klassen    Report_Alphabetische Artikelliste
'=======================================================

Private Sub Report_Open(Cancel As Integer)
  Me.Section(acDetail).Visible = False
End Sub
```

Listing 383: 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.

298 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 131: Die Argumente der Methode CreateReport

Im folgenden Beispiel aus *Listing 384* wird ein leerer Bericht auf Basis der Tabelle KATEGORIEN erzeugt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub NeuenBerichtEinfügen()
  Dim rpt As Report
  Dim strName As String

  Set rpt = CreateReport
  rpt.RecordSource = "Kategorien"
  strName = rpt.Name
  DoCmd.Close acReport, strName, acSaveYes
End Sub
```

Listing 384: 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.

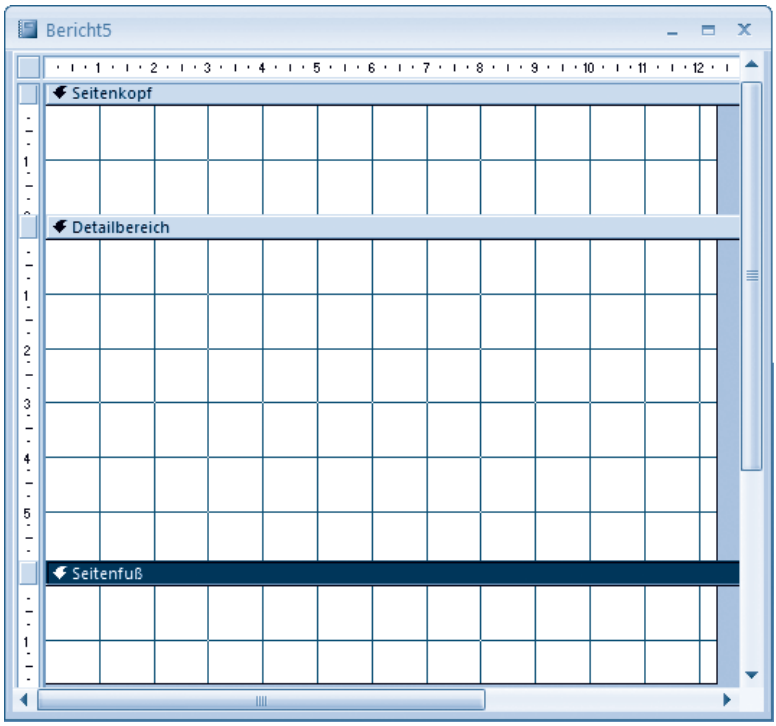


Abbildung 267: Einen neuen, noch leeren Bericht erzeugen

299 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

Tabelle 132: Die Argumente der Methode CreateReportControl

Argument	Beschreibung
Steuerelementtyp	▶ 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
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 Feldes 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 deren Abmessung einstellen. Die Einheit für diese Positionierung wird in TWIPS ausgegeben. Dabei entsprechen 567 Twips einem Zentimeter.

Tabelle 132: Die Argumente der Methode CreateReportControl

Im folgenden Beispiel aus *Listing 385* wird ein neuer Bericht auf Basis der Tabelle KATEGORIEN erstellt und mit einem Text sowie einem Bezeichnungsfeld angelegt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub BerichtselementeEinfügen()
  Dim rpt As Report
  Dim ctlBez As Control
  Dim ctlText As Control

  Set rpt = CreateReport
  rpt.RecordSource = "Kategorien"

  Set ctlText = CreateReportControl(rpt.Name, acTextBox, _
    acDetail, "", "Kategoriename", 1500, 50, 1200, 200)
  Set ctlBez = CreateReportControl _
    (rpt.Name, acLabel, , ctlText.Name, "Kategoriename", 150, 50, 800, 200)
  DoCmd.Close acReport, rpt.Name, acSaveYes
End Sub
```

Listing 385: Einen Bericht mit Feldern füllen

Möchten Sie einen Bericht noch schneller und komfortabler erstellen, dann können Sie dies auch über den Einsatz von ADO machen. Im folgenden Beispiel wird ein Bericht auf Basis der Tabelle KUNDEN erstellt und alle dazu benötigten Steuerelemente werden automatisch in den Bericht einfügt und verknüpft.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele\
' Dateiname  Berichte.mdb
' Modul      mdlReport
'=====================================================

Sub BerichtErstellen()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim fld As ADODB.Field
  Dim ctlText As Control
  Dim ctlBez As Control
  Dim lngPosOben As Long
  Dim lngPosLinks As Long
  Dim rpt As Report

  Set rpt = CreateReport
  lngPosLinks = 0
  lngPosOben = 0
```

Listing 386: Die vollautomatische Berichterstellung

```
  rpt.Report.RecordSource = "Kunden"
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  rst.Open _
    "SELECT * FROM Kunden WHERE (Land = 'Deutschland')", _
    conn, adOpenKeyset, adLockOptimistic
  For Each fld In rst.Fields
    Set ctlText = CreateReportControl(rpt.Report.Name, acTextBox, _
      acDetail, , fld.Name, lngPosLinks + 1500, lngPosOben)
    ctlText.SizeToFit
    Set ctlBez = CreateReportControl(rpt.Report.Name, acLabel, _
      acDetail, ctlText.Name, fld.Name, _
      lngPosLinks, lngPosOben, 1400, ctlText.Height)
    ctlBez.SizeToFit
    lngPosOben = lngPosOben + ctlText.Height + 50
  Next fld
  rst.Close
  Set rst = Nothing
  Set conn = Nothing
  Set rst = Nothing
End Sub
```

Listing 386: Die vollautomatische Berichterstellung (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 schon am Beginn der Prozedur über Variablen durch, damit Sie später diese nur hochaddieren 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 ein neues `Recordset`-Objekt.

Führen Sie dann nach dem Öffnen der Tabelle KUNDEN eine SQL-Abfrage durch, die alle Kunden aus Deutschland im `Recordset`-Objekt `rst` 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 Datenfeldes.

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 auf die Variable `lngPosOben` den Wert `1500`. Beim nächsten Schleifendurchlauf wird dadurch die neue Einfügeposition festgelegt.

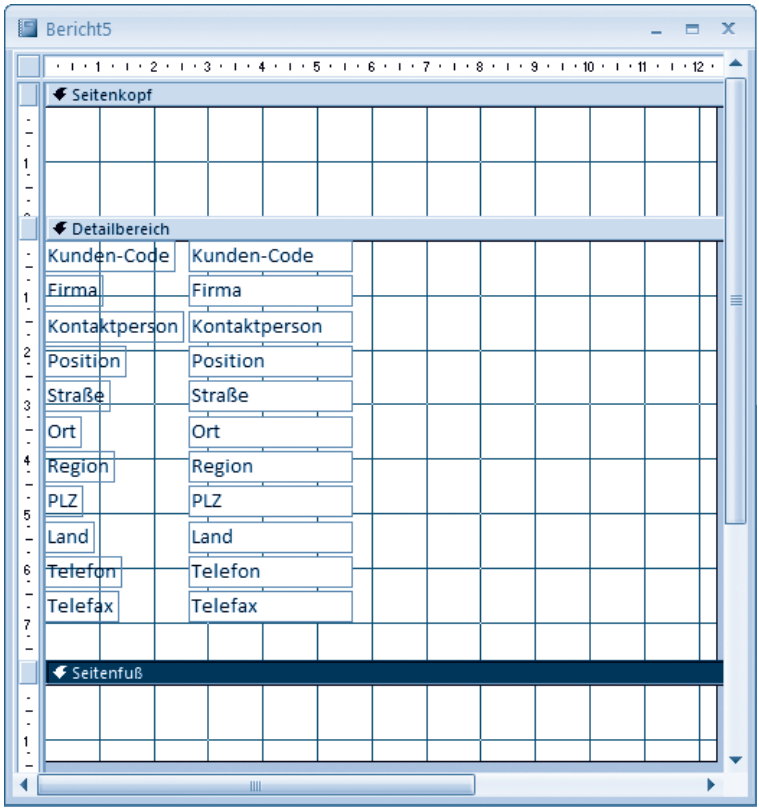


Abbildung 268: Der Bericht wurde angelegt.

Datenzugriffsseiten

Die Datenzugriffsseiten erstellen Sie wie ganz normale Formulare, die Sie über den Assistenten erzeugen.

Hinweis

Die folgenden Beispiele im Zusammenhang mit den Datenzugriffsseiten haben in Access 2007 keine Gültigkeit mehr. Datenzugriffsseiten wurden in Access 2007 durch die sogenannten SharePoint Services ersetzt. Dieses Thema ist jedoch nicht Bestandteil dieses Buches. Daher sind die folgenden Seiten nur für Anwender der Access-Versionen vor Access 2007.

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 Dropdown-Feld TABELLEN/ABFRAGEN die Quelle für Ihr Online-Formular aus.
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.

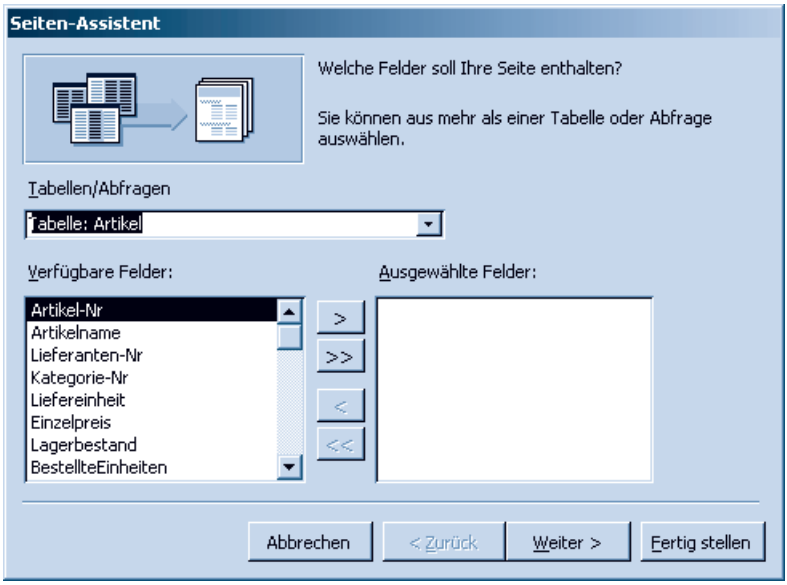


Abbildung 269: Datenzugriffsseite erstellen

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.

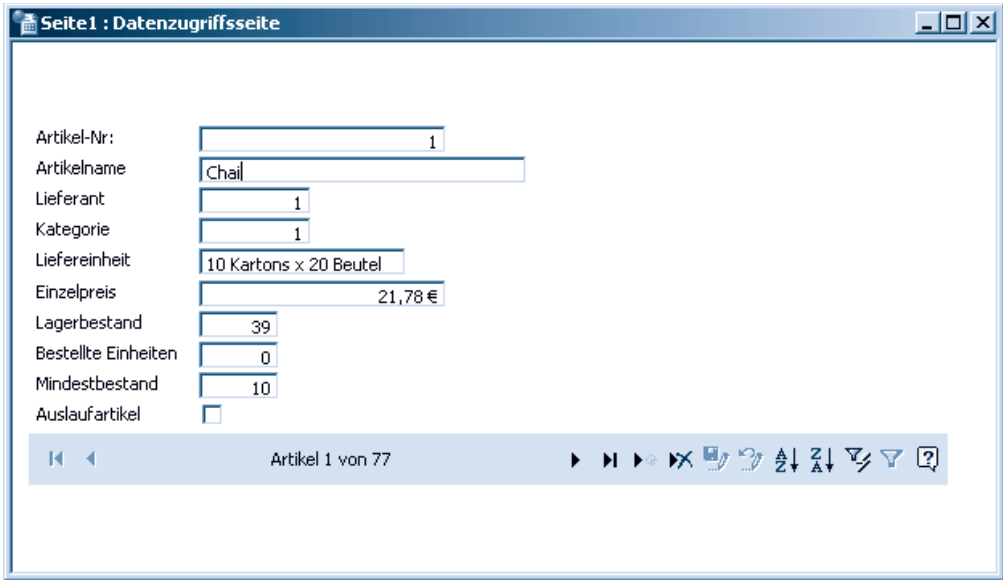


Abbildung 270: Die fertige Datenzugriffsseite

Die Datenzugriffsseite wird standardmäßig als HTML-Datei im selben Verzeichnis gespeichert wie die aktuelle Datenbank.

Das Erstellen einer Datenzugriffsseite kann auch über eine Prozedur, wie im Beispiel aus *Listing 387*, erledigt werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 387: 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 natürlich die standardmäßig angebotenen Designs 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 auf weißem Hintergrund und formschönen grünlichen Elementen aus.
Citrus	Bei diesem Design wird eine frühlingshafte Musterung 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 einem olivgrünen Hintergrund zu.
rmnsque	Bei diesem Design wird der Datenzugriffsseite ein gepunktetes Muster auf einem hellgrauen 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 133: Mögliche Designkonstanten für Datenzugriffsseiten

300 Datenzugriffsseiten auslesen

Um zu ermitteln, welche Datenzugriffsseiten in Ihrer aktuellen Datenbank vorhanden sind, können Sie die Prozedur aus *Listing 388* einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseitenAusgeben()
  Dim dap As AccessObject
  Dim dbs As CurrentProject

  Set dbs = Application.CurrentProject
  For Each dap In dbs.AllDataAccessPages
    Debug.Print dap.Name
  Next dap
End Sub
```

Listing 388: Alle Datenzugriffseiten 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.

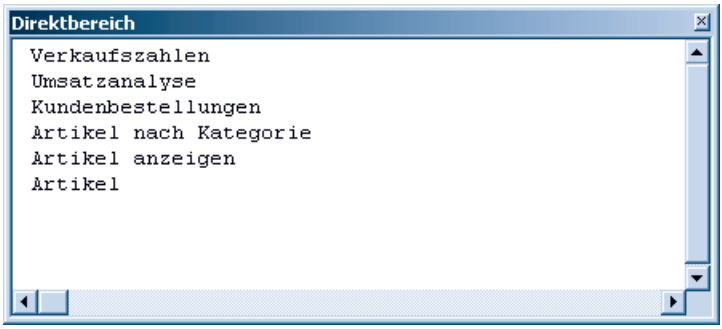


Abbildung 271: Alle Datenzugriffsseiten wurden ermittelt.

301 Speicherort der Datenzugriffsseiten ermitteln

Um herauszufinden, wo genau sich die Datenzugriffsseiten auf Ihrer Festplatte befinden, können Sie die Prozedur aus *Listing 389* einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseitenQuellenAusgeben()
  Dim dap As AccessObject
  Dim dbs As CurrentProject

  Set dbs = Application.CurrentProject

  For Each dap In dbs.AllDataAccessPages
    Debug.Print "Die Datenzugriffsseite:'" & _
      dap.Name & vbLf & "' ist gespeichert unter: " & _
      vbLf & dap.FullName & vbLf
  Next dap
End Sub
```

Listing 389: Die Speicherorte von Datenzugriffsseiten finden

Mit der Eigenschaft `FullName` können Sie den vollständigen Pfad (einschließlich des Dateinamens) der Datenzugriffsseite ermitteln.

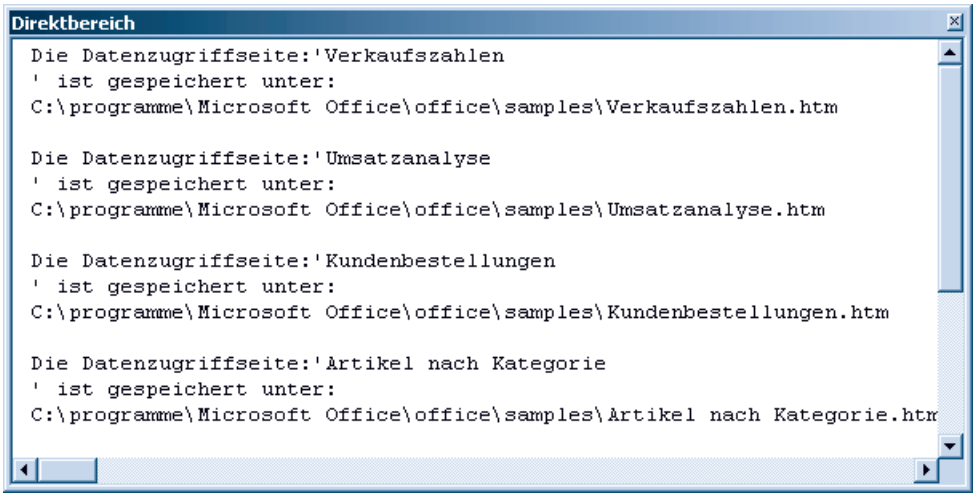


Abbildung 272: Die Speicherpfade der Datenzugriffsseiten werden ausgelesen.

302 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 390* wird die Datenzugriffsseite ARTIKEL geöffnet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Berichte.mdb
' Modul      mdlDap
'=====================================================

Sub DatenzugriffsseiteÖffnen()
  DoCmd.OpenDataAccessPage "Artikel", acDataAccessPageBrowse
End Sub
```

Listing 390: Datenzugriffsseite öffnen

303 Datenzugriffsseite anpassen

Über VBA-Prozeduren können Sie jederzeit bestehende Datenzugriffsseiten anpassen und wieder speichern. In der folgenden Prozedur aus *Listing 391* wird eine Datenzugriffsseite geöffnet, ein bestimmtes Steuerelement angesprungen und verändert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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
    .Echo True
  End With
End Sub
```

Listing 391: Datenzugriffsseite anpassen

Beim Ausführen einer Prozedur in Microsoft Access werden durch Bildschirmaktualisierungen oft Informationen angezeigt, die für die Funktionalität der Prozedur ohne Bedeutung sind. Wenn Sie das Argument `Echo` auf den Wert `False` setzen, wird die Prozedur ohne Bildschirmaktualisierung ausgeführt. Beim Beenden der Prozedur schaltet Microsoft Access `Echo` automatisch wieder ein und aktualisiert das Fenster.

Öffnen Sie die Datenzugriffsseite mithilfe der Methode `Open DataAccessPage` im Entwurfsmodus, 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 Feldes 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 gleich im Anschluss die Datenzugriffsseite in der Seitenansicht.

Die Ereignisprogrammierung

In Access gibt es jede Menge an Ereignissen, die Sie für die Programmierung einsetzen können. Diesen Ereignissen können dann zusätzliche Befehle zugewiesen werden. 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.

Die Ereignisse Form_Open bzw. Report_Open

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.

304 Zugangsverwaltung in Access (Formular)

Im folgenden Beispiel können Sie über das Ereignis `Open` eine Art Zugangsschutz für ein Formular einrichten. 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 eine neue Prozedur auf Modulebene an, wie in *Listing 392* dargestellt.

```
'======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Modul      mdlAPI
'======================================================

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

Listing 392: 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 Dropdown den Befehl FORMULAR.

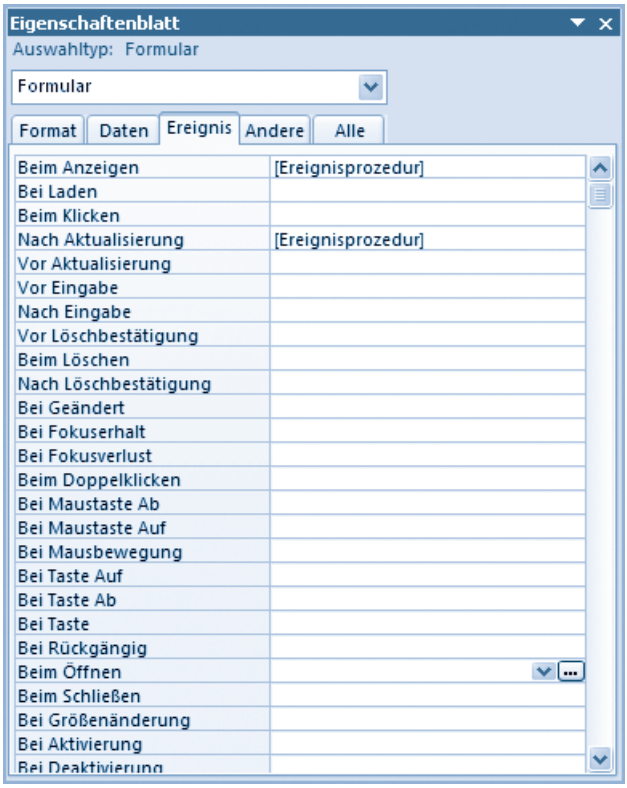


Abbildung 273: Eine Ereignis-Prozedur anlegen

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 die folgende Ereignisprozedur aus *Listing 393*.

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

Listing 393: Die Benutzerverwaltung über eine Ereignisprozedur einstellen

```
  lngSize = 8
  strID = "        "
  lngAns = GetUserName(strID, lngSize)
  strID = Trim(strID)
  If strID = "Held" Or strID = "admin" Then
    Me.TabCtl0.Pages(1).Visible = True
  Else
    Me.TabCtl0.Pages(1).Visible = False
  End If
End Sub
```

Listing 393: Die Benutzerverwaltung über eine Ereignisprozedur einstellen (Forts.)

Über die API-Funktion `GetUserName` 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 durch 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.

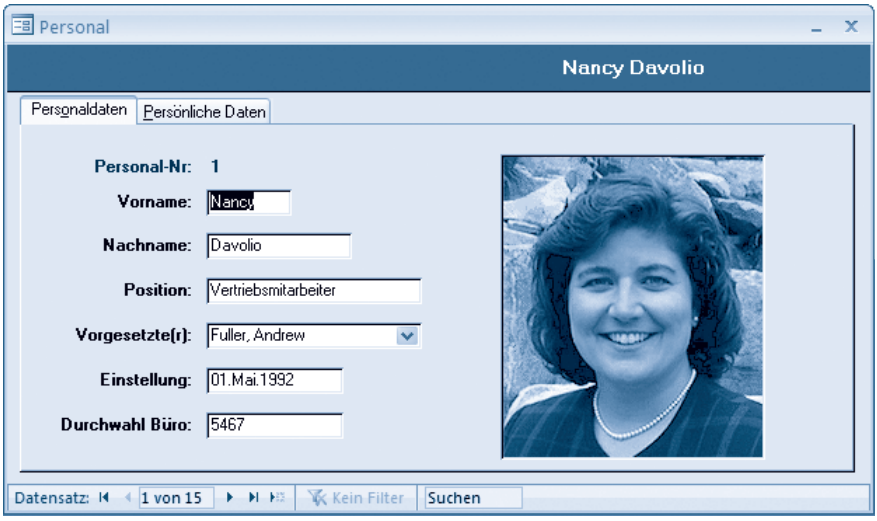


Abbildung 274: Als zugelassener Anwender dürfen Sie beide Registerblätter einsehen.

305 Zugangsverwaltung in Access (Bericht)

Auch für einen Bericht können Sie das Ereignis `Open` einsetzen. Dabei wird im Beispiel aus *Listing 394* das Ereignis `Report_Open` im BERICHT5 eingestellt und daneben geprüft, ob der Anwender den Bericht 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
' Dateiname  Ereignisse.mdb
' Klasse     Report_Bericht5
'=====================================================

Private Sub Report_Open(Cancel As Integer)
  Dim strUser As String

  strUser = Environ("Username")
  Select Case strUser
    Case "Held", "Mueller", "Schmitt"
      'Zugriff erlaubt
    Case Else
      'Zugriff verweigert
      MsgBox "Keine Berechtigung diesen Bericht auszuführen!"
      Cancel = True
  End Select
End Sub
```

Listing 394: Zugangsberechtigung bei Berichten prüfen

Über eine `Select Case`-Anweisung überprüfen Sie mit 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.

306 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 395* den Mauszeiger im Formular LIEFERANTEN nach dem Öffnen in das Feld TELEFON.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Form_Open(Cancel As Integer)
  Me.Telefon.SetFocus
End Sub
```

Listing 395: Den Fokus in ein bestimmtes Feld setzen

Mithilfe des Schlüsselwortes `Me` haben Sie Zugriff auf alle Steuerelemente des aktiven Formulars. So können Sie auch das Steuerelement TELEFON ansprechen und den Mauszeiger über die Methode `SetFocus` in das Feld hineinsetzen.

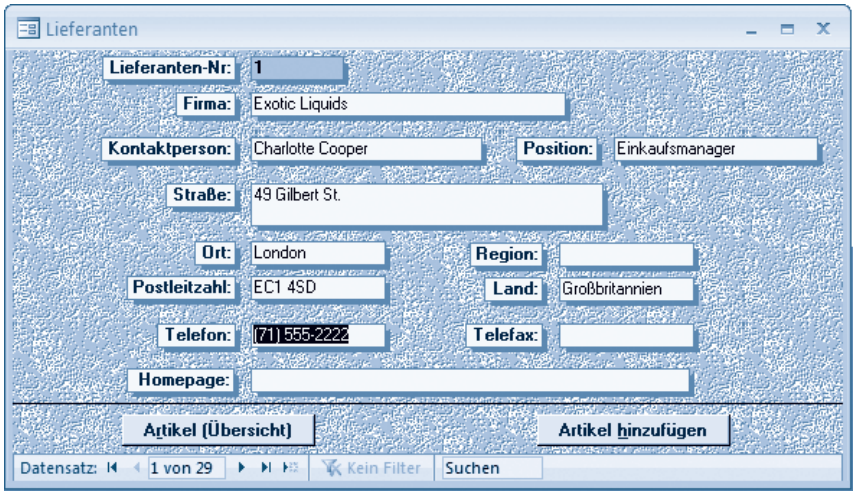


Abbildung 275: Der Fokus befindet sich nach dem Öffnen des Formulars im Feld Telefon.

307 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 die Prozedur aus *Listing 396* hinter dem Formular ARTIKEL3 ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel3
'=======================================================

Private Sub Form_Open(Cancel As Integer)
  Dim conn As Database
  Dim rst As Recordset

  Set conn = CurrentDb
  Set rst = conn.OpenRecordset("Artikel3", dbOpenDynaset)

  If rst.RecordCount = 0 Then
    MsgBox "Es sind keine Daten in der Tabelle " & rst.Name
    Cancel = True
  End If
```

Listing 396: Vor dem Öffnen prüfen, ob Daten bereitstehen

```
  rst.Close
  Set conn = Nothing
End Sub
```

Listing 396: Vor dem Öffnen prüfen, ob Daten bereitstehen (Forts.)

Mithilfe der Methode `CurrentDB` referenzieren Sie die momentan geöffnete Datenbank. Öffnen Sie danach die verknüpfte Tabelle und überprüfen Sie, ob in der Tabelle 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.

Die Ereignisse Form_Close und Report_Close

Das Ereignis `Close` tritt ein, wenn ein Formular oder ein Bericht geschlossen wurde und nicht mehr angezeigt wird. Das Schließen eines Formulars/Berichts können Sie nicht über VBA verhindern.

308 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 397* schließt nach dem Beenden des Formulars LIEFERANTEN die Formulare ARTIKEL und ARTIKELLISTE, sofern diese noch geöffnet sind.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Form_Close()
  'Die Formulare "Artikelliste" und "Artikel" schließen, wenn geöffnet
  If IsLoaded("Artikelliste") Then DoCmd.Close acForm, "Artikelliste"
  If IsLoaded("Artikel") Then DoCmd.Close acForm, "Artikel"
End Sub
```

Listing 397: 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.

309 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 nach ca. fünf Sekunden automatisch das dazugehörige Formular ARTIKEL geschlossen wird.

Diesen Countdown stellen Sie ein, indem Sie beim Schließen des Formulars LIEFERANTEN »den Zeitzünder scharf machen«. Dabei geben Sie folgendes Ereignis aus *Listing 398* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Sub Form_Close()
  Form_Artikel.TimerInterval = 5000
End Sub
```

Listing 398: Den Schließ-Verzögerer setzen

Mit der Eigenschaft `TimerInterval` können Sie die Länge des Intervalls in Millisekunden angeben, das zwischen zwei Timer-Ereignissen eines Formulars liegen soll.

Stellen Sie nun auf der anderen Seite im Formular ARTIKEL die Ereignisse aus *Listing 399* ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================
Private Sub Form_Load()
  Me.TimerInterval = 0
End Sub

Sub Form_Timer()
  Me.TimerInterval = 0
  MsgBox "Zeit abgelaufen!"
  DoCmd.Close acForm, "Artikel"
End Sub
```

Listing 399: Das Formular Artikel wird nach fünf Sekunden geschlossen.

Setzen Sie die Eigenschaft `TimerInterval` auf den Wert `0`. Rufen Sie danach testweise beide Formulare auf und beenden dann das Formular LIEFERANTEN. Gedulden Sie sich danach fünf Sekunden und sehen Sie dabei zu, wie auch das Formular ARTIKEL automatisch geschlossen wird.

310 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 nach der *Abbildung 276* an.

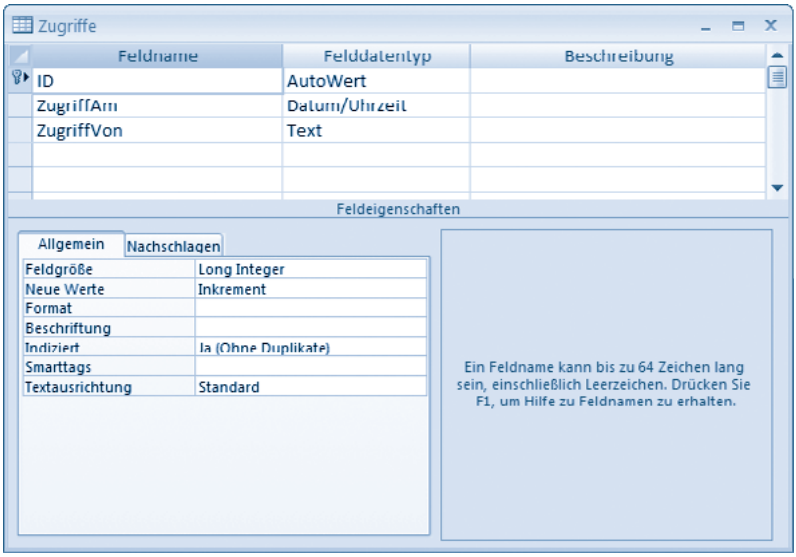


Abbildung 276: Die Tabelle für die Zugriffsüberwachung

Stellen Sie jetzt das Ereignis Report_Close ein und ergänzen den Ereignisrahmen wie in *Listing 400* gezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Report_Bericht5
'=====================================================

Private Sub Report_Close()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset

  With rst
    .Open "Zugriffe", conn, adOpenKeyset, adLockOptimistic
    .AddNew
    .Fields("ZugriffAm") = Now
    .Fields("ZugriffVon") = Environ("Username")
    .Update
    .Close
  End With

  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 400: Jeder Zugriff auf den Bericht wird festgehalten.

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. Mit dem Einsatz der Methode Update wird der Datensatz gesichert. Die Methode Close schließt die Tabelle wieder.

Abbildung 277: Alle Zugriffe werden im Hintergrund dokumentiert.

Das Ereignis Form_Load

Das Ereignis `Load` tritt ein, wenn ein Formular geöffnet wird und dessen Datensätze angezeigt werden. Dabei wird dieses Ereignis in der zeitlichen Reihenfolge vor dem Ereignis `Open` abgewickelt. Wenn Sie daher auf Werte bestimmter Datenfelder reagieren möchten, die nach dem Ereignis zur Verfügung stehen, dann ist dieses Ereignis das richtige.

311 Listenfeld beim Laden des Formulars füllen

Im folgenden Beispiel aus *Listing 401* wird im Formular LISTBOXFÜLLEN ein Listenfeld mit den in der Datenbank enthaltenen Tabellen gefüllt und der letzte Eintrag im Listenfeld markiert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_ListboxFüllen
'=======================================================

Private Sub Form_Load()
  Dim cat As ADOX.Catalog
  Dim TabInfo As ADOX.Table
  Dim intz As Integer

  Set cat = New ADOX.Catalog
  cat.ActiveConnection = _
  CurrentProject.Connection

  For Each TabInfo In cat.Tables
    With TabInfo
      If TabInfo.Type = "TABLE" Then
        Me!Liste0.AddItem .Name
      End If
```

Listing 401: Listenfeld füllen und letzten Eintrag markieren

```
    End With
  Next

  Set cat = Nothing
  intz = Me.Liste0.ListCount
  Me.Liste0.Selected(intz - 1) = True
End Sub
```

Listing 401: Listenfeld füllen und letzten Eintrag markieren (Forts.)

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 Listenfeldes den Wert `1` subtrahieren. Über die Eigenschaft `Selected` markieren Sie einen Eintrag im Listenfeld.

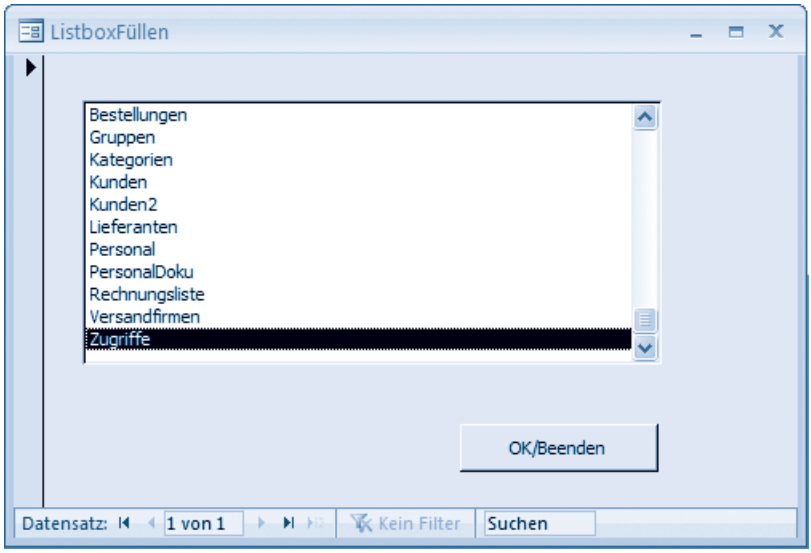


Abbildung 278: Der letzte Eintrag im Listenfeld wurde markiert.

Das Ereignis Form_Unload

Über das Ereignis `Form_Unload` können Sie auf das Schließen eines Formulars reagieren.

312 Schließen verhindern

In der folgenden Lösung soll verhindert werden, dass ein Formular versehentlich geschlossen wird. Dazu soll eine Rückfrage angezeigt werden, ob das Formular geschlossen werden soll oder nicht.

Im Beispiel aus *Listing 402* wird beim Schließen des Formulars KUNDEN2 nachgefragt, ob der Vorgang wirklich durchgeführt werden soll.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden2
'=====================================================

Private Sub Form_Unload(Cancel As Integer)
  Dim intRück As Integer

  intRück = MsgBox("Möchten Sie beenden?", vbYesNo + vbQuestion)
  If intRück = 6 Then
    Cancel = False
  Else
    Cancel = True
  End If
End Sub
```

Listing 402: 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. Im 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.

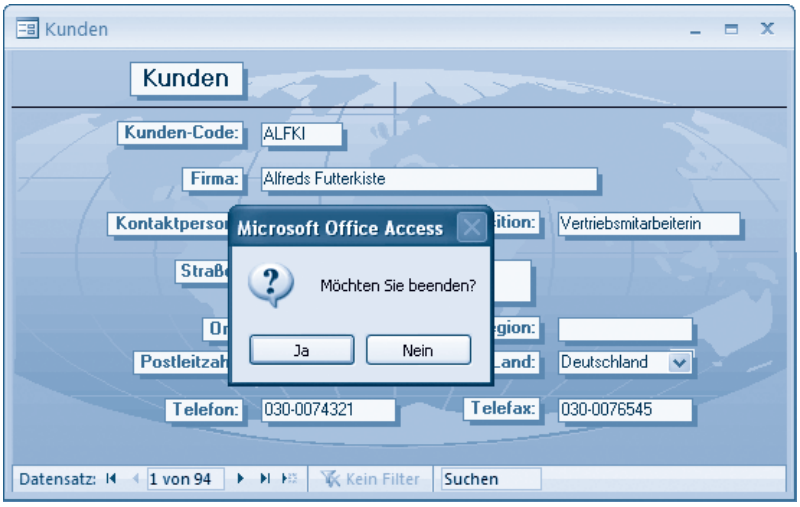


Abbildung 279: Beim Beenden der Form erscheint eine Rückfrage.

313 Rechnungsfälligkeiten überwachen

Gegeben ist die Tabelle mit dem Namen RECHNUNGSLISTE aus der *Abbildung 280* und ein Formular RECHNUNGSLISTE, über das Sie Ihre Rechnungen erfassen und überwachen. Wenn Sie durch das Formular blättern, dann sollen Sie auf Rechnungen, deren Fälligkeitstermin bereits überschritten ist, sofort aufmerksam gemacht werden.

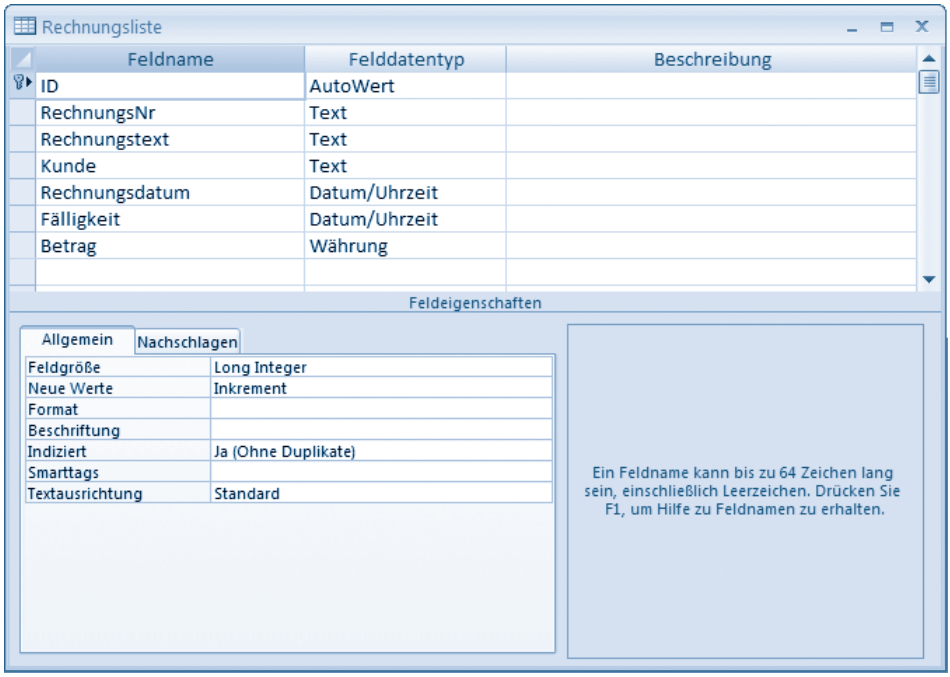


Abbildung 280: Der Aufbau der Rechnungsliste

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 die Ereignisprozedur `Form_Current` aus *Listing 403*, die automatisch beim Blättern durch das Formular aufgerufen wird.

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

Listing 403: Rechnungsfälligkeiten überwachen

```
  Else
    Me.Fälligkeit.ForeColor = RGB(0, 0, 0)
    Me.Fälligkeit.FontBold = True
  End If
End Sub
```

Listing 403: Rechnungsfälligkeiten überwachen (Forts.)

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 Feldes zu formatieren. Die gewünschte Farbe mischen Sie über die Funktion `RGB`. Über die Eigenschaft `FontBold` geben Sie der Schriftart die Formatierung FETT, indem Sie diese mit dem Wert `True` ausstatten.

Abbildung 281: Die Rechnungsfälligkeit wurde überschritten.

Das Ereignis Form_Current

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.

314 Information beim letzten Satz

Im nächsten Beispiel aus *Listing 404* 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
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================
```

Listing 404: Beim Erreichen des letzten Satzes eine Meldung anzeigen

```
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 404: Beim Erreichen des letzten Satzes eine Meldung anzeigen (Forts.)

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 die Eigenschaft `RecordCount` an, um die Anzahl der hinterlegten Datensätze zu erfragen. Diese Anzahl vergleichen Sie dann mit dem momentan eingestellten Datensatz im Formular. Im 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` wieder den ersten Datensatz im Formular ein.

Abbildung 282: Der letzte Datensatz im Formular wurde erreicht.

315 Dynamisches Ein- und Ausblenden von Feldern

In Abhängigkeit von bestimmten Datenfeldern 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 können Sie in *Listing 405* sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 405: 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 beinhalten, darauf, dass Sie den Feldnamen in eckige Klammern setzen.

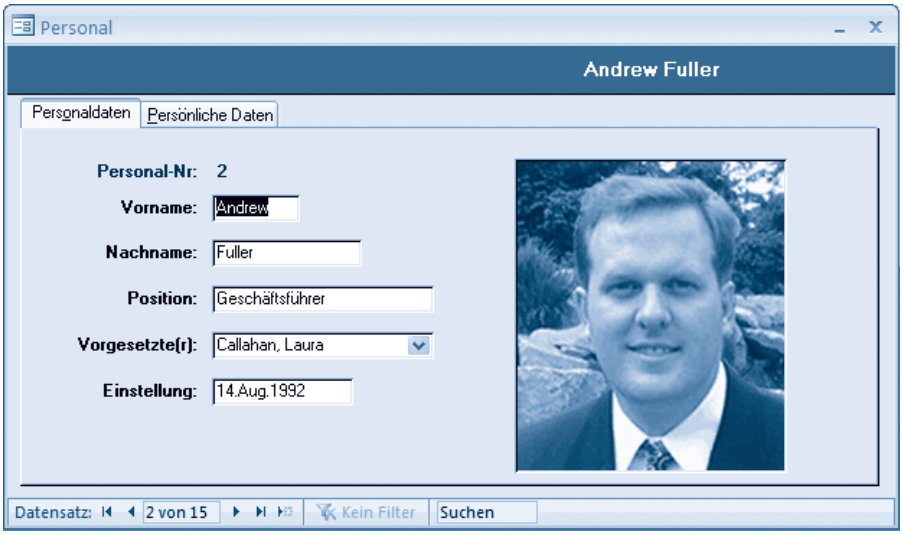


Abbildung 283: Beim Geschäftsführer wurde die Telefon-Nr. ausgeblendet.

316 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 im *Listing 406* der Name der jeweiligen Kategorie in Großbuchstaben angezeigt. Wenn eine neue Kategorie hinzugefügt wird, wird das aktuelle Datum in der Titelleiste ausgegeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 406: 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.

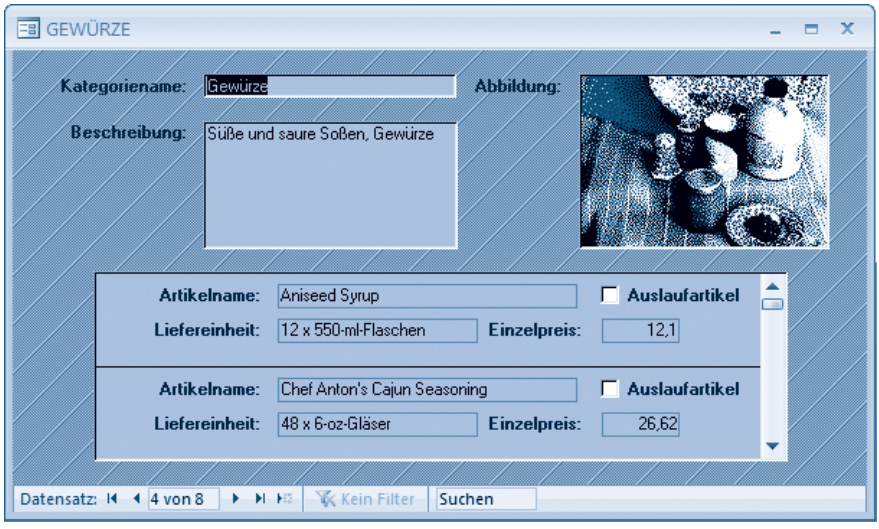


Abbildung 284: Der Kategoriename wird als Fensterüberschrift definiert.

317 QuickInfos definieren

Im nächsten Beispiel aus *Listing 407* 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 Textfeldes.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 407: Den Feldwert zusätzlich als QuickInfo angeben

In einer `For Each Next`-Schleife durchlaufen Sie alle Steuerelemente auf 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 Textfeldes.

Abbildung 285: Die QuickInfo ist beim Feld Liefereinheit zu sehen.

318 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 maximal 100 Sätze eingegeben werden können.

Im Beispiel aus *Listing 408* wird im Formular KUNDEN2 die Eingabe von höchstens 100 Sätzen erlaubt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden2
'=====================================================

Private Sub Form_Current()
  Dim rst As Recordset
  Dim intz As Integer

  Set rst = RecordsetClone
  intz = rst.RecordCount

  If Me.NewRecord Then
    Me.AllowAdditions = intz < 100
    If Not Me.AllowAdditions Then
      MsgBox "Die max. Anzahl der Sätze ist erreicht!"
    End If
  Else
    Me.AllowEdits = True
  End If
End Sub
```

Listing 408: Die Eingabe von Datensätzen einschränken

Mit der Eigenschaft `RecordsetClone` stellen Sie eine Kopie der Tabelle her, die dem Formular zugrunde liegt. Über die Methode `RecordCount` ermitteln Sie die Anzahl der 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 die Eigenschaft `AllowEdits` im `Else`-Zweig der `If`-Anweisung auf den Wert `True`. Im `Then`-Zweig der `If`-Anweisung definieren Sie eine Bedingung, die `intz < 100` lautet.

Das Ereignis Form_AfterInsert

Das Ereignis `AfterInsert` tritt nach dem Hinzufügen eines neuen Datensatzes ein.

319 Neue Mitarbeiter dokumentieren

Beim folgenden Beispiel werden über das Formular PERSONAL neue Mitarbeiter erfasst. Um nun eine lückenlose Dokumentation der Neueinstellungen zu haben, erstellen Sie zunächst eine Tabelle wie sie in *Abbildung 286* gezeigt ist.

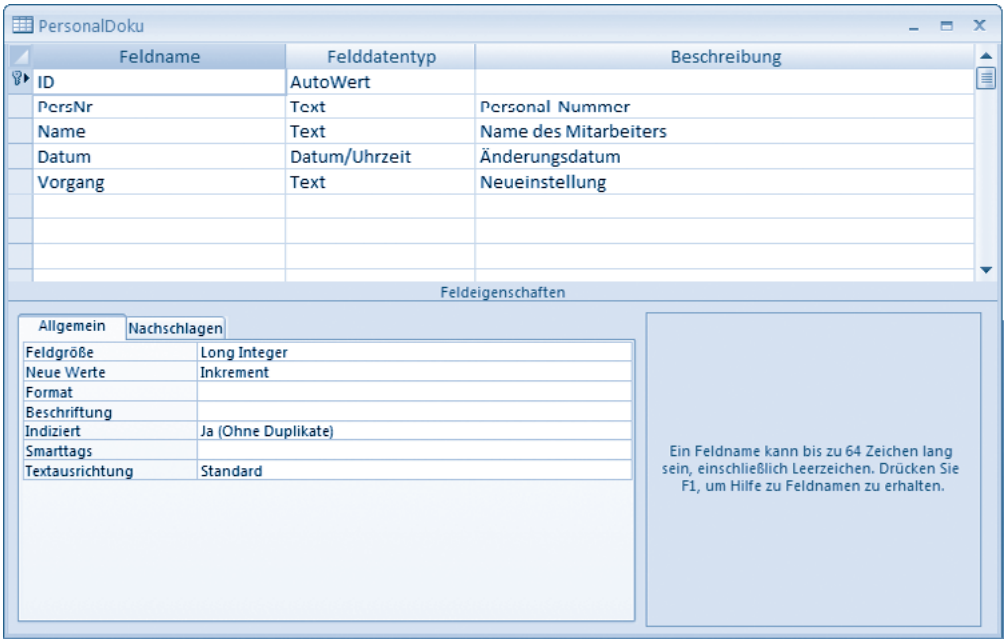


Abbildung 286: Die Tabelle zur Dokumentierung des Personals

Erfassen Sie jetzt das Ereignis `AfterInsert` und ergänzen Sie es wie in *Listing 409* gezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Form_AfterInsert()
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset
```

Listing 409: Neue Einstellungen werden in einer Tabelle dokumentiert.

```
  If Me.NewRecord = False Then
    MsgBox "Neuanlage"
    Set conn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "PersonalDoku", conn, adOpenKeyset, adLockOptimistic
    rst.AddNew
    rst.Fields("PersNR") = Me.[Personal-Nr]
    rst.Fields("Name") = Me.Nachname & ", " & Me.Vorname
    rst.Fields("Datum") = Me.Einstellung
    rst.Fields("Vorgang") = "Neueinstellung"
    rst.Update
    rst.Close
    Set rst = Nothing
    Set conn = Nothing
  End If
End Sub
```

Listing 409: 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 bemerken, 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 die Felder aus dem Formular PERSONAL in die Tabelle. Vergessen Sie dabei nicht, am Ende die Methode `Update` aufzurufen, um sämtliche neuen Feldinhalte in die Tabelle zu übernehmen. Schließen Sie danach die Tabelle über die Methode `Close` und heben die Objektverknüpfung auf.

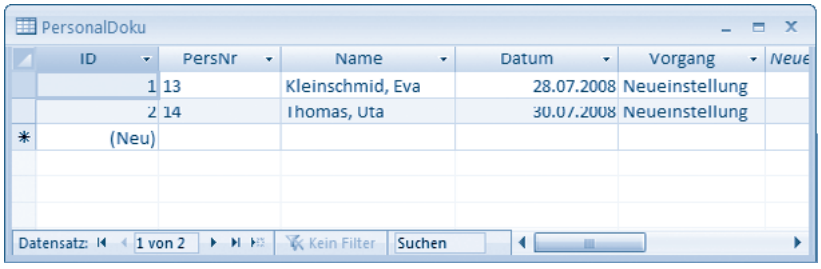


Abbildung 287: Alle Neueinstellungen werden lückenlos festgehalten.

Das Ereignis Form_BeforeInsert

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.

320 Felder automatisch vorbelegen

Im nächsten Beispiel aus *Listing 410* 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
' 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 410: 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 288: Die zuletzt vorgenommenen Einträge wurden automatisch eingestellt.

Hinweis

Wenn Sie lieber Festwerte als Vorbelegungstexte haben möchten, dann geben Sie diese, wie in *Listing 411* gezeigt, in doppelten Anführungszeichen an.

Hinweis

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden (auskommentiert)
'=======================================================

Sub Form_BeforeInsert(Cancel As Integer)
  With Me
    .Ort = "Stuttgart"
    .Region = "BW"
    .Land = "Deutschland"
  End With
End Sub
```

Listing 411:Festtexte als Vorgabe verwenden

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. Beim folgenden Beispiel aus *Listing 412* wird im ARTIKEL2 jeweils der erste Satz der Tabelle ARTIKEL eingesetzt, wenn ein neuer Satz angegangen wird.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=======================================================

Private Sub Form_BeforeInsert(Cancel As Integer)
  Dim rst As Recordset

  Set rst = Me.RecordsetClone
  With Me
    If rst.RecordCount > 0 Then
      rst.MoveFirst
      .Artikelname = rst!Artikelname
      .[Lieferanten-Nr] = rst![Lieferanten-Nr]
      .[Kategorie-Nr] = rst![Kategorie-Nr]
      .Liefereinheit = rst!Liefereinheit
      .Einzelpreis = rst!Einzelpreis
      .Lagerbestand = rst!Lagerbestand
      .BestellteEinheiten = rst!BestellteEinheiten
      .Mindestbestand = rst!Mindestbestand
      .Auslaufartikel = rst!Auslaufartikel
    End If
  End With
End Sub
```

Listing 412: Den ersten Satz im Formular einstellen

Mit der Eigenschaft RecordsetClone stellen Sie ein Duplikat der Abfrage her, die dem Formular zugrunde liegt. Diese wird über die Eigenschaft RecordSource in den Eigenschaften des Formulars festgelegt. Aktivieren Sie den ersten Datensatz der Tabelle über die Methode MoveFirst. Jetzt übertragen Sie die Inhalte dieses Satzes Feld für Feld in die Formularfelder.

Abbildung 289: Der erste Datensatz wurde als Vorlage verwendet.

Das Ereignis Form_BeforeUpdate

Das Ereignis BeforeUpdate tritt auf, bevor geänderte Daten in einem Datensatz aktualisiert werden.

321 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. In der Prozedur aus *Listing 413* wurde beim Formular LIEFERANTEN diese Rückfrage eingebaut.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
  If MsgBox("Möchten Sie wirklich speichern?", _
    vbYesNo + vbQuestion, "Speichern bestätigen") <> vbYes Then
    Cancel = True
```

Listing 413: Eine Rückfrage vor dem Speichern einholen

```
    Me.Undo
  End If
End Sub
```

Listing 413: Eine Rückfrage vor dem Speichern einholen (Forts.)

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 das Argument `Cancel` auf den Wert `True`, um das Ereignis abzubrechen.

322 Eingabe-Prüfung vornehmen

Das Ereignis `Form_BeforeUpdate` können Sie unter anderem gut einsetzen, wenn Sie Eingaben in Formularfeldern kontrollieren möchten.

Im folgenden Beispiel aus *Listing 414* wird im Formular PERSONAL geprüft, ob die Eingabe einer Postleitzahl richtig durchgeführt wurde. Für die Länder Frankreich, Italien, Spanien und Deutschland gilt die einheitliche Länge von fünf Zeichen. In Australien und Singapur wird mit einer 4-stelligen Postleitzahl gearbeitet. In Kanada enthält die Postleitzahl sowohl numerische als auch alphanumerische Zeichen. Dabei wird die kanadische Postleitzahl in zwei Dreier-Blöcken z. B. H1J 1C3 für Quebec dargestellt.

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

Listing 414: Die PLZ-Prüfung vor dem Speichern des Datensatzes durchführen

```
    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 414: Die PLZ-Prüfung vor dem Speichern des Datensatzes durchführen (Forts.)

Über eine `Select Case`-Anweisung wird das Feld LAND ausgewertet. Ist kein Land erfasst, kann auch eine PLZ-Prüfung unterbleiben. Daher wird in diesem Zweig die Prozedur 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, dann 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, damit eine Korrektur schnell erfolgen kann.

Bei den Postleitzahlen von Kanada muss ein Abgleich eines Musters, bestehend aus zwei Dreier-Blöcken, durchgeführt werden. 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 das Schlüsselwort `Like` durch.

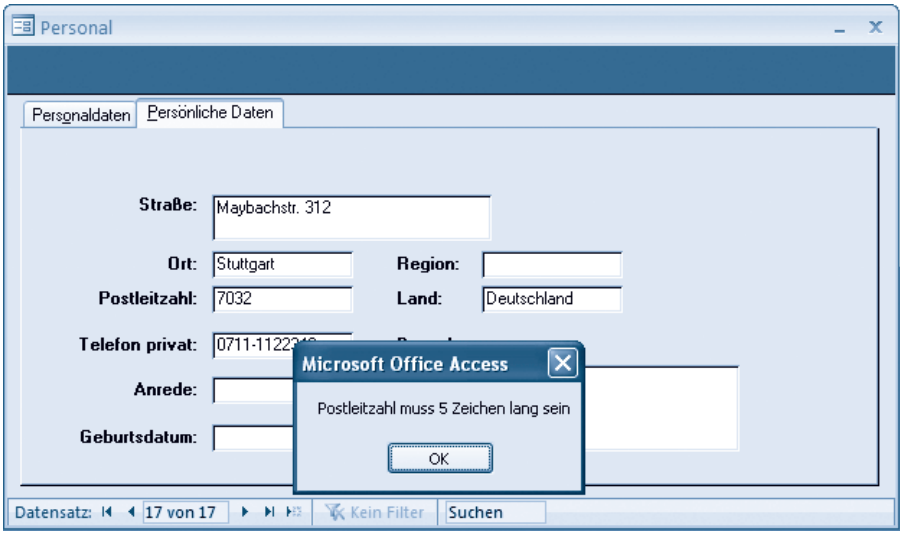


Abbildung 290: Eine ungültige PLZ für Deutschland wurde erfasst.

323 Muss-Felder definieren

Auf eine ganz ähnliche Weise können Sie Muss-Felder definieren, d. h. in diesen Feldern muss ein Eintrag vorgenommen werden, damit der Datensatz gespeichert werden kann.

Im folgenden Beispiel aus *Listing 415* werden die beiden Felder LAGERBESTAND sowie MINDESTBESTAND im Formular ARTIKEL als Muss-Felder definiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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
  End If
  If IsNull(Me.Mindestbestand) Then
    MsgBox "Geben Sie einen Mindestbestand ein!"
    Cancel = True
  End If
End Sub
```

Listing 415: Muss-Felder definieren

Mithilfe der Funktion `IsNull` können Sie überprüfen, ob in einem Feld eine Eingabe vorgenommen wurde. Wenn dies nicht der Fall ist, dann machen Sie den Anwender mit einer Bildschirmmeldung darauf aufmerksam.

324 Änderungen in Textdatei festhalten

Beim nächsten Beispiel aus *Listing 416* werden alle Änderungen im Formular KUNDEN bzw. in der dahinter liegenden Tabelle KUNDEN in eine Textdatei geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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")
```

Listing 416: Änderungen am Kundenstamm zusätzlich festhalten

```
  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 416: Änderungen am Kundenstamm zusätzlich festhalten (Forts.)

Sobald Sie eine Änderung in der Tabelle KUNDEN über das Formular KUNDEN vornehmen, wird im Hintergrund eine Textdatei geöffnet. Dabei übergeben Sie bei der Methode `Open` neben dem Dateipfad das Argument `For Append`. Damit werden alle Änderungen in der Textdatei jeweils am Ende der Textdatei angehängt. Schreiben Sie zu Beginn der jeweiligen Änderung mithilfe der Anweisung `Print` das Änderungsdatum sowie den Namen des Anwenders. Danach durchlaufen Sie mit 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`.

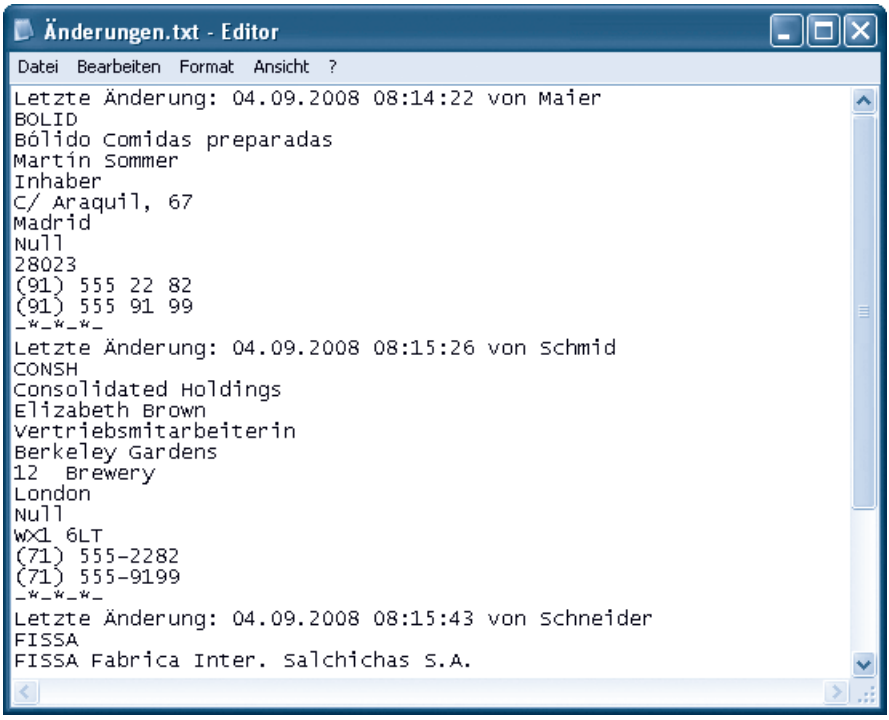


Abbildung 291: Alle Änderungen werden in einer separaten Textdatei festgehalten.

325 Neuanlage von Datensätzen verhindern

Das Ereignis `Form_BeforeUpdate` können Sie ebenso einsetzen, wenn Sie eine Neuanlage eines Datensatzes über ein Formular verhindern möchten.

Im Beispiel aus *Listing 417* wird im Formular KATEGORIEN die Neuanlage von Datensätzen für bestimmte Anwender verweigert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kategorien
'=====================================================

Private Sub Form_BeforeUpdate(Cancel As Integer)
  If Me.NewRecord And Environ("username") <> "Held" Then
    MsgBox "Es ist nicht möglich, neue Sätze anzuhängen!"
    Me.Undo
    Cancel = True
  End If
End Sub
```

Listing 417: 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 wiederherzustellen. Indem Sie das Argument `Cancel` auf den Wert `True` setzen, wird der Speichervorgang abgebrochen.

326 Überhaupt keine Änderungen zulassen

Gehen Sie einen Schritt weiter und lassen Sie im nächsten Beispiel aus *Listing 418* überhaupt keine Änderung zu. Auch hier können Sie das Ereignis `Form_BeforeUpdate` einsetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 418: Das Ereignis BeforeUpdate einsetzen, um Änderungen an Datensätzen zu verhindern

Mit der Eigenschaft Dirty können Sie ermitteln, ob der aktuelle Datensatz seit dem letzten Speichern verändert wurde. Wenn ja, wenden Sie die Methode Undo an, um die getätigte Änderung rückgängig zu machen. Setzen Sie das Argument Cancel auf den Wert True, um den Speichervorgang abzubrechen.

Das Ereignis Form_AfterUpdate

Das Ereignis AfterUpdate tritt in einem Formular dann ein, wenn ein Datensatz aktualisiert wurde.

327 Letzte Änderung festhalten

Bei der Prozedur aus *Listing 419* wurde in der Tabelle ARTIKEL ein zusätzliches Feld mit dem Namen LETZTESUPDATE angelegt. Nachdem eine Änderung an einem Datensatz im Formular ARTIKEL ausgeführt wurde, wird in der Tabelle beim betreffenden Satz automatisch das Änderungsdatum geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Form_AfterUpdate()
  Dim rst As ADODB.Recordset
  Dim var As Variant

  var = DLookup("[Artikel-Nr]", "Artikel", "[Artikel-Nr] = " _
    & Forms!Artikel!ArtikelNr)
  var = "[Artikel-Nr]='" & var & "'"

  On Error GoTo Fehler
  Set rst = New ADODB.Recordset

  With rst
    .Open Source:="Artikel", ActiveConnection:=CurrentProject.Connection, _
     CursorType:=adOpenKeyset, LockType:=adLockOptimistic

    .Find Criteria:=var, SearchDirection:=adSearchForward

    If Not .EOF Then
      .Fields("LetztesUpdate") = Now
    End If
  End With
```

Listing 419: Die letzte Änderung an einem Datensatz in der Tabelle zurückschreiben

```
  rst.Update
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 419: 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 diesen Suchbegriff nach dem Öffnen der Tabelle der Methode `Find`. Wurde der Datensatz der Tabelle gefunden, dann übertragen Sie die aktuelle Zeit, inkl. des Datums, über die Funktion `Now` in das Feld LETZTESUPDATE. 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 292: Die letzten Änderungen sind jetzt sofort ersichtlich.

Das Ereignis Form_Delete

Das Ereignis `Form_Delete` tritt dann ein, wenn ein Anwender eine Lösch-Aktion ausführt, z. B. das Drücken der Taste Entf zum Löschen eines Datensatzes. Das Ereignis tritt ein, bevor der Datensatz tatsächlich gelöscht wird.

328 Löschen von Datensätzen verhindern

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 420* wird im Formular ARTIKEL jegliche Löschung von Artikeln verhindert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 420: 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 aber mit einer Bildschirmmeldung auf diese Funktion aufmerksam machen.

In der Praxis wird man hin und wieder zusätzliche Kriterien definieren, die ein Löschen von Datensätzen generell erlauben, jedoch nicht in speziellen Fällen.

Im Beispiel aus *Listing 421* dürfen Kunden aus Deutschland im Formular KUNDEN nicht gelöscht werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 421: Das Löschen ist nicht für alle Datensätze erlaubt.

Das Ereignis BeforeDelConfirm

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.

329 Eine eigene Löschabfrage einsetzen

Wenn Sie einen Datensatz aus einem Formular löschen 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 422* gezeigt, ersetzen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 422: Eine eigene Lösch-Rückfrage einsetzen

Im Dialogfeld zeigen Sie die Schaltflächen OK und ABBRECHEN an. Wenn die Schaltfläche ABBRECHEN angeklickt wurde, wird der Wert `2` zurückgemeldet, was der Konstante `vbCancel` entspricht. In diesem Fall setzen Sie das Argument `Cancel` auf den Wert `True`, um die Standardlöschabfrage von Access abzubrechen.

Das Ereignis Form_Activate

Das Ereignis `Activate` tritt ein, wenn ein Formular den Fokus erhält und zum aktiven Fenster wird.

330 Den letzten Datensatz einstellen

Beim Beispiel aus *Listing 423* wird im Formular ARTIKEL2 standardmäßig der letzte Datensatz der damit verknüpften Tabelle ARTIKEL angezeigt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Form_Activate()
  DoCmd.GoToRecord , , acLast
End Sub
```

Listing 423: Den letzten Datensatz der Tabelle im Formular einstellen

Setzen Sie die Methode `GoToRecord` ein und übergeben dieser Methode die Konstante `acLast`, um das Formular mit dem letzten Datensatz der verknüpften Tabelle einzustellen.

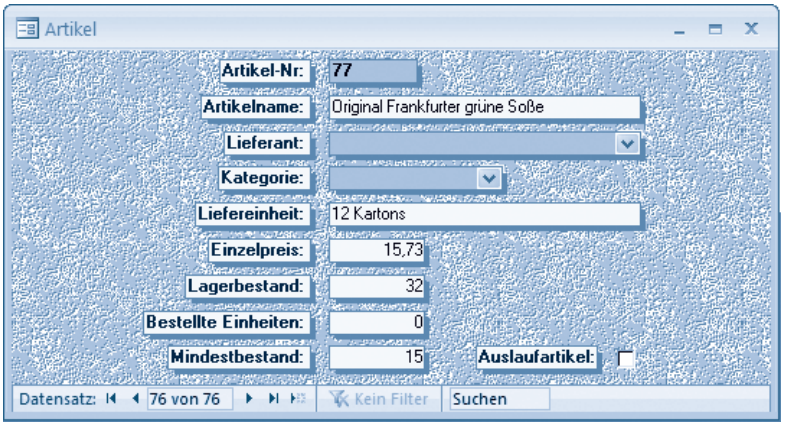


Abbildung 293: Der letzte Satz wird angezeigt.

331 Den zuletzt geänderten Datensatz aufrufen

Erinnern Sie sich noch, wie Sie vorher die letzte Änderung eines Datensatzes in der Tabelle ARTIKEL im Feld LETZTESUPDATE über das Ereignis `Form_AfterUpdate` vorgenommen haben?

Jetzt können Sie genau auf diese letzte Änderung zugreifen und beim Aktivieren des Formulars genau diesen Datensatz aufrufen, indem Sie das Ereignis aus *Listing 424* einsetzen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=======================================================

Private Sub Form_Activate()
  Dim rst As ADODB.Recordset
  Dim var As Variant
  Dim lngPos As Integer

  var = DMax("LetztesUpdate", "Artikel")
  var = "LetztesUpdate='" & var & "'"

  On Error GoTo Fehler
  Set rst = New ADODB.Recordset
```

Listing 424: Den zuletzt geänderten Datensatz im Formular voreinstellen

```
  With rst
    .Open Source:="Artikel", ActiveConnection:=CurrentProject.Connection, _
      CursorType:=adOpenKeyset, LockType:=adLockOptimistic
    .Find Criteria:=var, SearchDirection:=adSearchForward
      lngPos = .AbsolutePosition
    If Not .EOF Then
      DoCmd.GoToRecord acDataForm, "Artikel", acGoTo, lngPos
    End If
  End With
  rst.Close
  Set rst = Nothing
  Exit Sub

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

Listing 424: Den zuletzt geänderten Datensatz im Formular voreinstellen (Forts.)

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 LETZTESUPDATE. Bilden Sie danach den Suchbegriff, der sich aus dem Feldnamen in der Tabelle sowie dem ermittelten höchsten Datum zusammensetzt, und übergeben diesen Suchbegriff nach dem Öffnen der Tabelle der Methode `Find`. Wurde der Datensatz der Tabelle gefunden, dann speichern Sie die aktuelle Position des `Recordset`-Objektes 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.

Das Ereignis Form_Resize

Das Ereignis `Form_Resize` tritt ein, wenn ein Formular geöffnet wird und wenn sich die Größe eines Formulars ändert.

332 Variable Feldanpassung vornehmen

Beim Eintreten des `Resize`-Ereignisses können Sie ein Steuerelement verschieben oder dessen Größe ändern, wenn sich die Größe des Formulars ändert, auf dem sich dieses Steuerelement befindet.

Im nächsten Beispiel aus *Listing 425* werden Sie die Länge der beiden Textfelder FIRMA und STRAßE im Formular KUNDEN variabel gestalten.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 425: Textfelder werden dynamisch angepasst.

Über die Eigenschaft `WindowWidth` können Sie die aktuelle Breite eines Formulars ermitteln. 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.

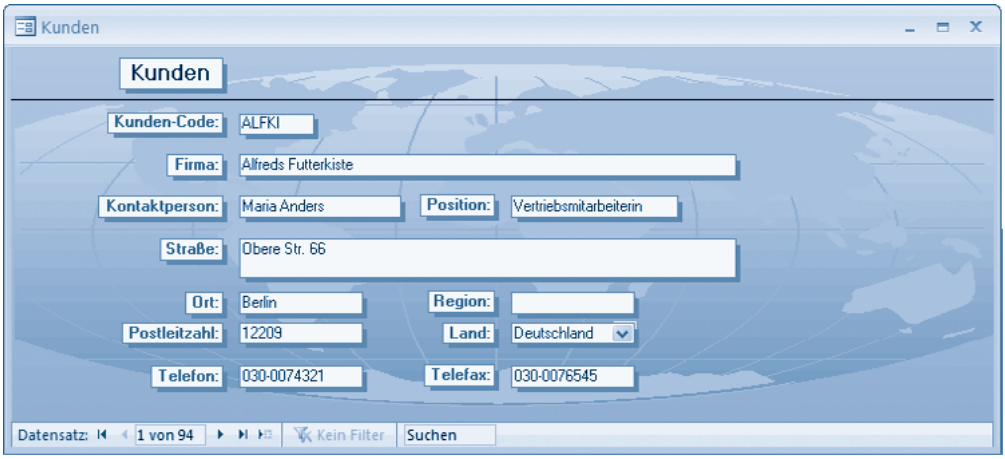


Abbildung 294: Beim Verbreitern des Formulars werden die Textfelder Firma und Straße angepasst.

Das Ereignis Form_DblClick

Bei Formularen tritt das Ereignis `DblClick` ein, wenn der Benutzer doppelt auf eine leere Fläche oder auf einen Datensatzmarkierer des Formulars klickt.

333 Mit Doppelklick Felder ein- und ausblenden

Beim folgenden Beispiel aus *Listing 426* werden mit einem Doppelklick auf das Formular ARTIKEL bestimmte Felder, je nach augenblicklichem Status, entweder ein- bzw. ausgeblendet.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 426: 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.

Abbildung 295: Alle Felder sind sichtbar.

Klicken Sie doppelt auf eine freie Fläche im Formular, um die Felder auszublenden.

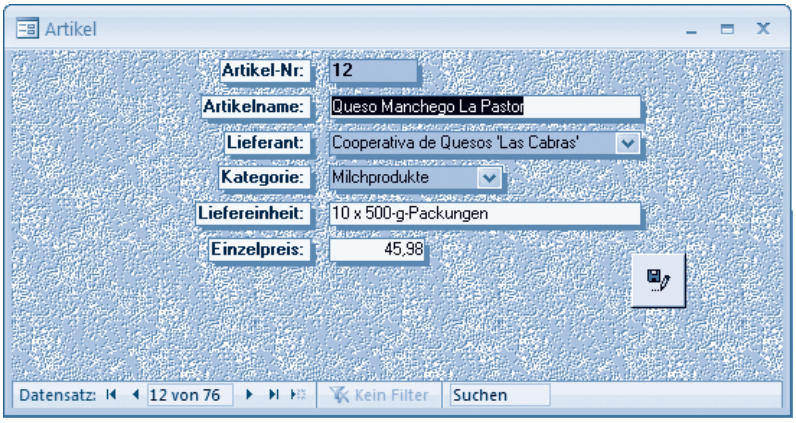


Abbildung 296: Einige Felder sind nun ausgeblendet.

Das Ereignis Form_Click

Das Ereignis `Click` tritt auf, wenn der Benutzer auf die linke Maustaste klickt, während sich der Mauszeiger auf einem freien Bereich oder auf dem Datensatzmarkierer in der Datenblattansicht des Formulars befindet. Dieses Ereignis kann man selbstverständlich auf jedes andere Steuerelement im Formular anwenden.

334 Lagerbestand per Mausklick erhöhen

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 427* gezeigt ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel
'=====================================================

Private Sub Lagerbestand_Click()
  Me.Lagerbestand.Value = Me.Lagerbestand.Value + 1
End Sub
```

Listing 427: Lagerzugänge per Mausklick einpflegen

Die Ereignisse Form_MouseDown und Form_MouseUp

Mit diesen beiden Ereignissen, die eigentlich dem `Click`-Ereignis ähnlich sind, können Sie unter anderem überprüfen, welche Taste der Anwender gedrückt hat. Das Ereignis `Form_MouseDown` tritt dann ein, wenn der Anwender eine Maustaste drückt. Das Ereignis `Form_MouseUp` tritt auf, wenn der Anwender die gedrückte Maustaste wieder loslässt.

Das Ereignis `Form_MouseDown` tritt vor dem Ereignis `Click` auf. Das Ereignis `Form_MouseUp` liegt in der zeitlichen Reihenfolge hinter dem `Click`-Ereignis. Das `Click`-Ereignis erkennt zudem, wenn eine Maustaste gedrückt und losgelassen wurde. Dabei ist dieser Vorgang eine einzige Aktion. Ein weiterer Unterschied zwischen den Mausereignissen und dem `Click`-Ereignis besteht darin, dass diese Mausereignisse es ermöglichen, zwischen der linken, rechten und mittleren Maustaste sowie den Tasten `Umschalt`, `Strg` und `Alt` zu unterscheiden.

335 Was wurde angeklickt?

Zunächst einmal werden Sie in *Listing 428* sehen, wie man feststellen kann, welche Maustaste im Detailbereich des Formulars KUNDEN geklickt wurde. Dabei kann es vorkommen, dass vorher, zusätzlich zu den gedrückt gehaltenen Maustasten, eine der Tasten `Alt`, `Umschalt` oder `Strg` gedrückt wurde.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Kunden
'=====================================================

Private Sub Detail_MouseDown(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)
```

Listing 428: Feststellen, welche Maustaste gedrückt wurde

```
  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 428: Feststellen, welche Maustaste gedrückt wurde (Forts.)

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.

336 Schaltfläche beim Klicken verändern

Beim folgenden Beispiel aus *Listing 429* wird im Formular LISTBOXFÜLLEN die Schaltfläche beim Klicken verändert. Dabei wird der Schriftschnitt FETT sowie die Schriftfarbe ROT eingestellt. Beim Freigeben der linken Maustaste wird die Standardformatierung wiederhergestellt.


```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_ListboxFüllen
'=====================================================

Private Sub Befehl2_MouseDown(Button As Integer, Shift As Integer, _
  X As Single, Y As Single)
```

Listing 429: Schaltflächen beim Klick verändern

```
  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 429: Schaltflächen beim Klick verändern (Forts.)

Mit der Funktion `QBColor` können Sie die Farbe bestimmen, 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.

337 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 dabei fest, in welchem Bereich des Formulars dieses Kontextmenü ausgeschaltet werden soll.

Im folgenden Beispiel aus *Listing 430* wird im Detailbereich des Formulars LIEFERANTEN das Kontextmenü deaktiviert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Lieferanten
'=====================================================

Private Sub Detail_MouseDown(Button As Integer, Shift As Integer, _
```

Listing 430: Das Kontextmenü deaktivieren

```
  X As Single, Y As Single)
  If Button = acRightButton Then
    Me.ShortcutMenu = False
  End If
End Sub
```

Listing 430: Das Kontextmenü deaktivieren (Forts.)

Über die Abfrage der gedrückten Taste auf Ihrer Maus können Sie mit dem 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.

Das Ereignis MouseMove

Das Ereignis `MouseMove` tritt ein, wenn der Benutzer die Maus bewegt. Das Ereignis wird dabei ständig wiederholt ausgelöst, während sich der Mauszeiger über einem Objekt bewegt. Solange kein anderes Objekt ein Mausereignis auslöst, erkennt ein Objekt immer dann das Ereignis `MouseMove`, wenn der Mauszeiger innerhalb der Umrandung des Objekts positioniert wird.

338 Spezialeffekt 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 297* gezeigt aus.

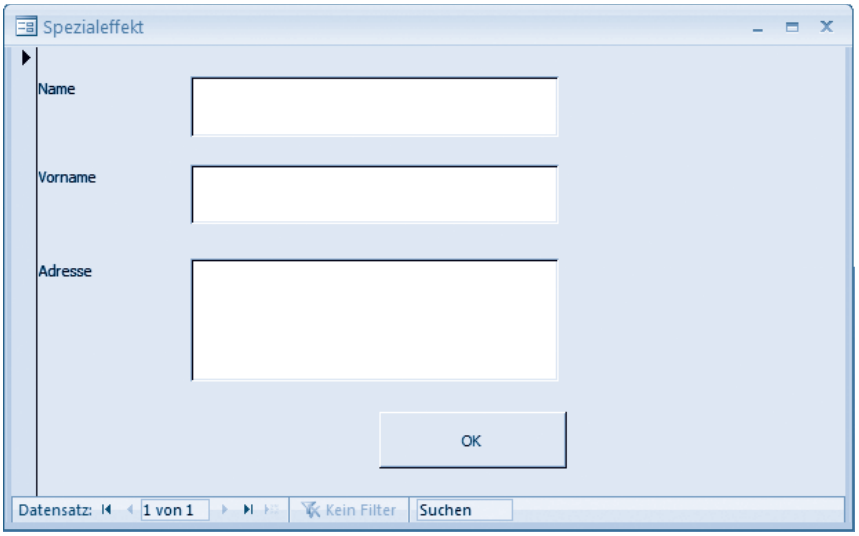


Abbildung 297: Die Textfelder sind alle identisch eingestellt.

Sie können jetzt das Ereignis `MouseOver` einsetzen, um einen anderen Spezialeffekt für die Felder, über die Sie streichen, einzustellen.

Im folgenden Beispiel aus *Listing 431* wird beim Überstreichen der Textfelder geprüft, welcher Spezialeffekt eingestellt ist, und dementsprechend ein anderer Spezialeffekt eingestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Spezialeffekt
'=====================================================

Private 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


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 431: Spezialeffekte für Textfelder einsetzen

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 aus 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 134: Die Spezialeffekte in Access

339 Ins Internet verzweigen

Auf dem Formular SPEZIALEFFEKT wurde bei der folgenden Aufgabe eine weitere Schaltfläche integriert, deren Beschriftung eine gültige Internetadresse beinhaltet. Wenn man mit der Maus über diese Schaltfläche 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 432* gestartet werden.

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

Listing 432: Eine Internetseite besuchen

```
  Dim intA As Integer
  intA = MsgBox("Möchten Sie eine Verbindung ins Internet herstellen?", _
    vbYesNo + vbQuestion)

  If intA = vbYes Then
    ShellExecute Me.hwnd, "open", Befehl10.Caption, 0, 0, 5
  End If
End Sub
```

Listing 432: Eine Internetseite besuchen (Forts.)

Die Sprungadresse für die Internetseite holen Sie sich mithilfe der Eigenschaft `Caption` direkt aus der Beschriftung der Schaltfläche. Die notwendige Aktion heißt `open`, was nichts anderes bedeutet, als dass die Internetseite geöffnet werden soll.

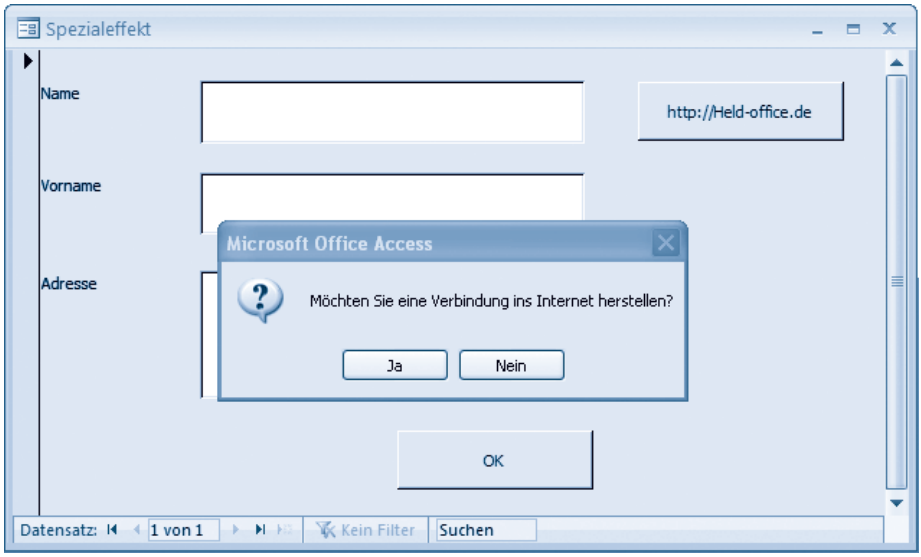


Abbildung 298: Rückfrage vor dem Sprung ins Internet einholen

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.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Spezialeffekt
'=======================================================
```

Listing 433: Den Internet Explorer starten

```
Private Sub Befehl11_Click()
  Dim objIE As Object

  Set objIE = CreateObject("InternetExplorer.Application")
  objIE.Navigate Befehl11.Caption
  objIE.Visible = True
End Sub
```

Listing 433: Den Internet Explorer starten (Forts.)

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.

Die Ereignisse GotFocus und LostFocus

Das Ereignis `GotFocus` tritt auf, wenn ein Formular oder ein Steuerelement den Fokus erhält. Das Ereignis `LostFocus` tritt ein, wenn ein Formular oder ein Steuerelement den Fokus abgibt. Diese beiden Ereignisse werden in der Praxis häufig eingesetzt, um Steuerelemente zu formatieren.

340 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 434* wird im Formular TEXTFELDER diese Methode angewendet. Dabei wird bei Fokuserhalt des Textfeldes die Hintergrundfarbe des Textfeldes mit GELB formatiert. Beim Fokusverlust wird die Standardfarbe des jeweiligen Textfeldes wiederhergestellt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Textfelder
'=====================================================

Private Sub Text0_GotFocus()
  Me.Text0.BackColor = QBColor(6)
End Sub
```

Listing 434: Textfelder bei Fokuserhalt färben

```
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 434: Textfelder bei Fokuserhalt färben (Forts.)

Über die Eigenschaft `BackColor` können Sie den Hintergrund eines Textfeldes mit einer Farbe belegen. Die Farbe wird über die Funktion `QBColor` erzeugt. Dabei wird die Farbe Gelb durch die Nummer `6`, und die blendend weiße Farbe durch die Nummer `15` repräsentiert. Eine Tabelle mit den Farbzuordnungen können Sie im Anhang dieses Buches nachschlagen.

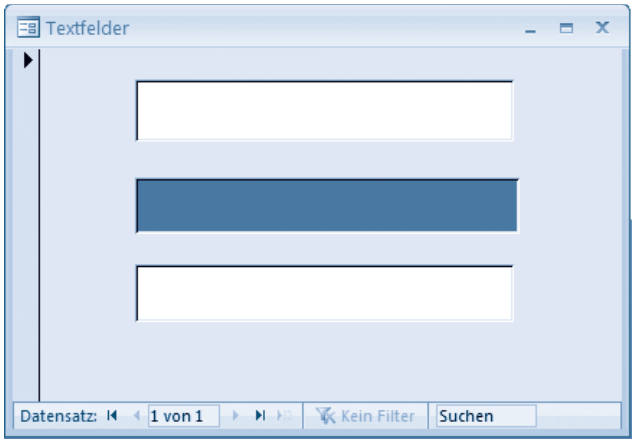


Abbildung 299: Das zweite Feld wurde bei Fokuserhalt mit Farbe hinterlegt.

Die Key-Ereignisse

In Access können Sie auf jeden einzelnen Tastenklick reagieren. Dazu stehen Ihnen die Ereignisse KeyDown und KeyUp zur Verfügung. Eine Taste wird gedrückt (KeyDown) und eine Taste wird losgelassen (KeyUp).

341 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, Umschalt und Strg drücken. All diese Tasten werden in der Prozedur aus *Listing 435* erkannt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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!"
  End Select
End Sub
```

Listing 435: Tasten auswerten über den KeyCode

Jede Taste kann über eine sogenannte KeyCode-Konstante angesprochen werden.

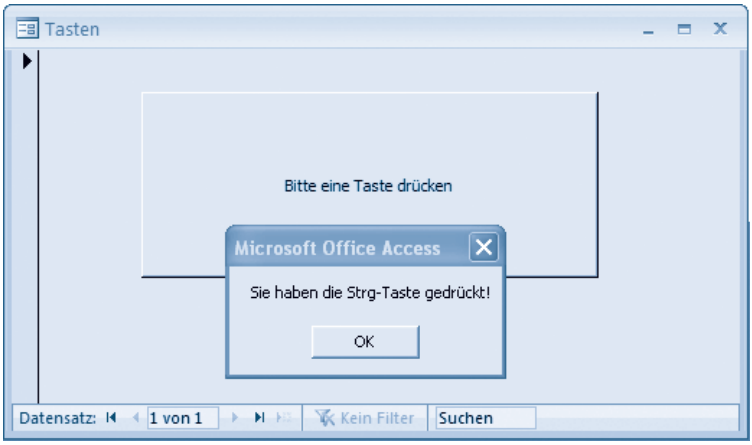


Abbildung 300: Die Abfrage von bestimmten Tasten in einem Formular

Hinweis

Weitere Informationen über die Belegung der Tasten finden Sie im Antwort-Assistenten der Online-Hilfe unter dem Stichwort TASTEN-CODE-KONSTANTEN.

342 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 oder eleganter und schneller über Tastenkombinationen erledigt werden. Für das Füllen der Felder wurden dabei 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 436* gezeigt, eingestellt, d. h., die Tastenkombinationen sind dann nur gültig, wenn der Mauszeiger im jeweiligen Textfeld ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Tastenkombinationen
'=====================================================
```

Listing 436: Die Tastenkombinationen abfragen und Aktionen durchführen

```
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 436: Die Tastenkombinationen abfragen und Aktionen durchführen (Forts.)

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.

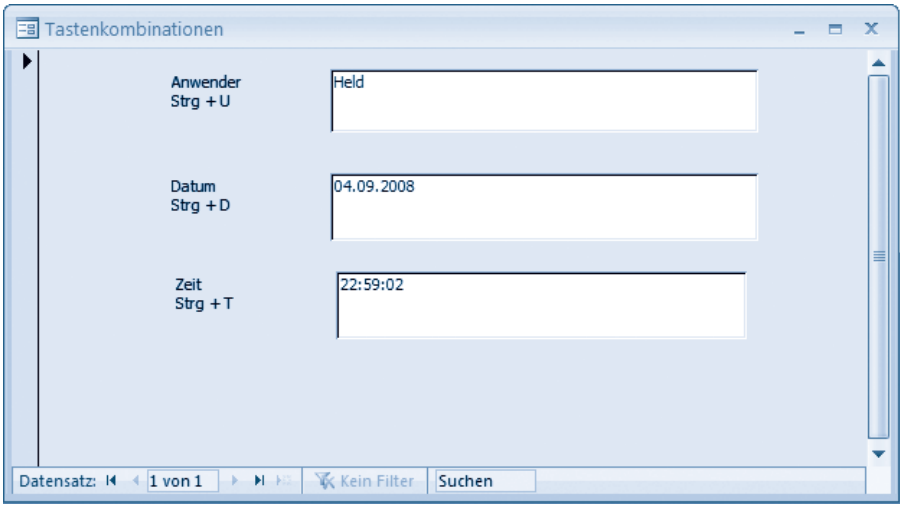


Abbildung 301: Schnellere Eingabe durch Tastenkombinationen erreichen

343 Aktuellen Datensatz ans Ende kopieren

Beim folgenden Beispiel wird die Tastenkombination Strg + Y im Feld ARTIKELNAME des Formulars ARTIKEL2 genutzt, um den aktuell aktiven Datensatz 1:1 ans Ende zu kopieren. Die Artikelnummer wird dabei jedoch noch wie gewünscht von Access selbst vergeben. Legen Sie die Prozedur aus *Listing 437* hinter das Feld ARTIKELNAME.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Artikel2
'=====================================================

Private Sub Artikelname_KeyDown(KeyCode As Integer, Shift As Integer)
  Dim rst As Recordset

  'Aktuellen Satz kopieren
  If (Shift = acCtrlMask) And KeyCode = Asc("Y") Then
    Set rst = Me.RecordsetClone

    If rst.RecordCount > 0 Then
      rst.AddNew
      rst!Artikelname = Me.Artikelname
      rst![Lieferanten-Nr] = Me.[Lieferanten-Nr]
      rst![Kategorie-Nr] = Me.[Kategorie-Nr]
      rst!Liefereinheit = Me.Liefereinheit
      rst!Einzelpreis = Me.Einzelpreis
      rst!Lagerbestand = Me.Lagerbestand
      rst!BestellteEinheiten = Me.BestellteEinheiten
      rst!Mindestbestand = Me.Mindestbestand
      rst!Auslaufartikel = Me.Auslaufartikel
      rst.Update
      rst.Requery
    End If
  End If
End Sub
```

Listing 437: Aktuellen Datensatz ans Ende kopieren

Erstellen Sie zunächst mithilfe der Methode `RecordsetClone` einen »Klon« von der aktuellen Abfrage. Danach fügen Sie über die Methode `AddNew` einen neuen Datensatz im Formular ein und füllen diesen mit den Werten aus dem aktuellen Datensatz. Über die Methode `Update` speichern Sie den neuen Datensatz.

Hinweis

Die Tastenkombination Strg + Y funktioniert nur dann, wenn der Mauszeiger im Feld ARTIKELNAME des Formulars ARTKEL2 steht.

Das Ereignis Steuerelement_BeforeUpdate

Das Ereignis BeforeUpdate tritt auf, bevor geänderte Daten in einem Steuerelement oder Datensatz aktualisiert werden. Dies ist ein guter Augenblick, um Gültigkeitsprüfungen durchzuführen, denn das Ereignis BeforeUpdate kann auch gecancelt werden, wenn die Gültigkeitsprüfung ein ungültiges Ergebnis detektiert.

344 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 Feldes FIRMA ein, das Sie in *Listing 438* sehen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 438: Einträge bei der Eingabe auf Eindeutigkeit überprüfen

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, das Sie auf den Wert True setzen, um die Speicherung des Kunden abzubrechen.

345 Eingaben vorwegnehmen

Das Ereignis `BeforeUpdate` können Sie auch einsetzen, um einige Formularfelder automatisch füllen zu lassen. So können Sie beispielsweise bei der Eingabe der Postleitzahl automatisch den dazugehörigen Ort, die Region und das Land in die dafür zur Verfügung stehenden Formularfelder einfügen.

Beim folgenden Beispiel aus *Listing 439* wird im Formular KUNDEN2 das Feld PLZ überwacht. Sobald dort eine Eingabe vorgenommen wird, wird im bisherigen Kundenstamm nachgesucht, ob diese Postleitzahl bereits verwendet wurde. Wenn ja, dann wird der dazugehörige Ort, die Region und das Land an den neuen Datensatz vererbt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 439: 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.

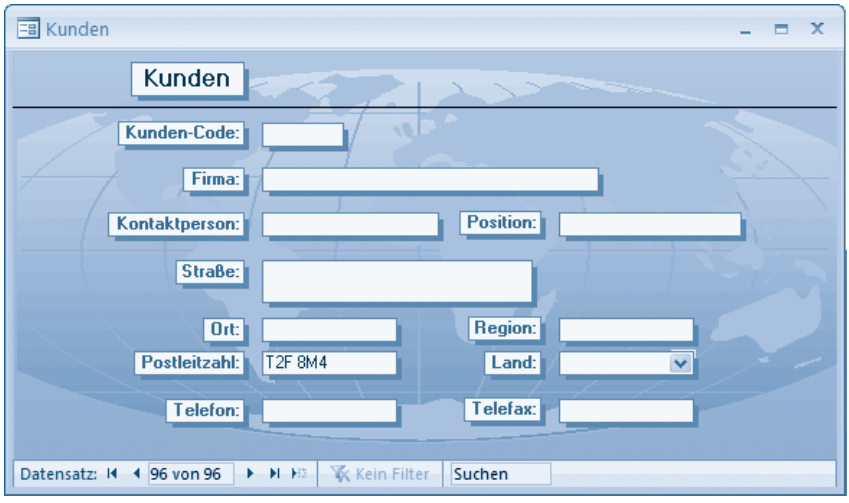


Abbildung 302: Die PLZ wurde erfasst.

Die PLZ wurde erfasst und mit der Taste Tab wurde ins nächste Feld gewechselt. Dieser Vorgang löst das Ereignis BeforeUpdate aus.

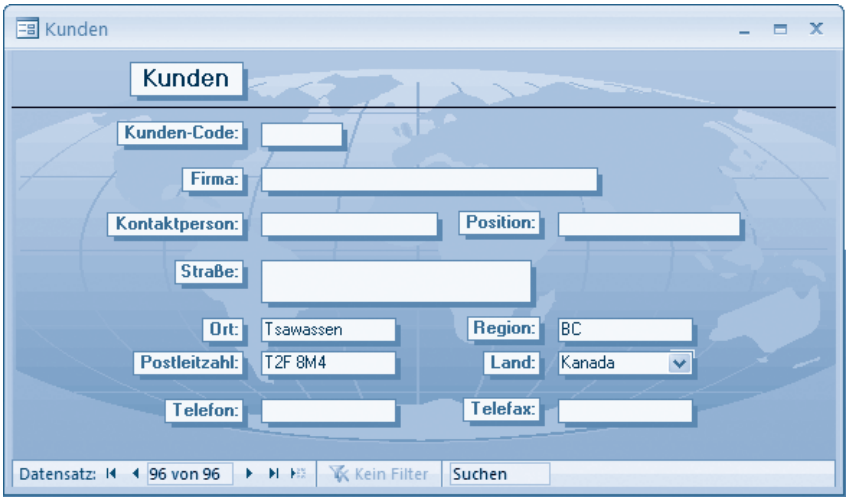


Abbildung 303: Die Felder Ort, Region und Land wurden automatisch gefüllt.

346 Feldlöschung verhindern

Mithilfe des Ereignisses BeforeUpdate können Sie Ihre Anwender auch davon abhalten, bestimmte Felder im Formular LIEFERANTEN zu löschen.

In der folgenden Ereignisprozedur aus *Listing 440* wird verhindert, dass bereits erfasste Werte im Textfeld KONTAKTPERSON gelöscht werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 440: Die Löschung von Daten verhindern

Mithilfe der Funktion IsNull bekommen 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.

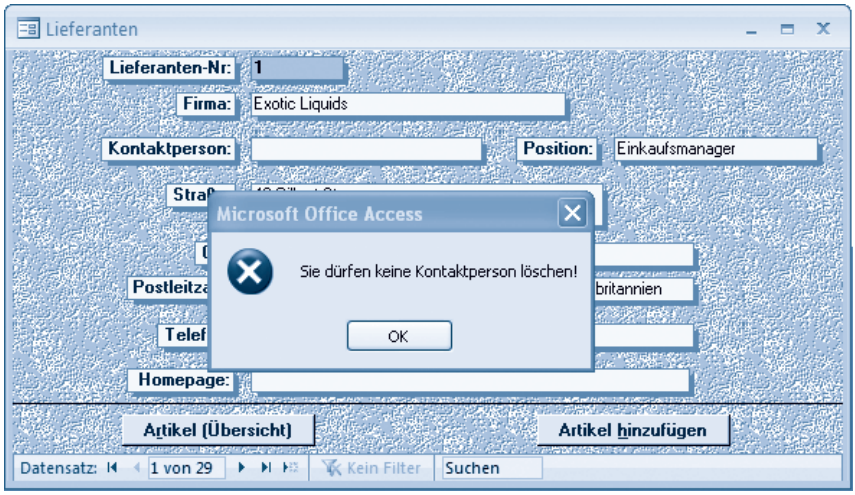


Abbildung 304: Beim Löschen einer Kontaktperson wird eine Meldung angezeigt.

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, das im Anschluss beschrieben wird.

Das Ereignis Steuerelement_Enter

Das Ereignis `Enter` tritt ein, bevor ein Steuerelement den Fokus von einem anderen Steuerelement desselben Formulars erhält. Dieses Ereignis wirkt sich nicht auf Kontrollkästchen, Optionsfelder oder Umschaltflächen in einer Optionsgruppe aus.

347 QuickInfos anzeigen

Im folgenden Beispiel aus *Listing 441* 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. Ob das Feld leer ist, machen Sie daran aus, ob es entweder den Wert null oder einen leeren String enthält.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Ereignisse.mdb
' Klasse     Form_Personal
'=====================================================

Private Sub Bemerkungen_Enter()
  Dim strText As String

  strText = "Bitte eigene Informationen angeben!"

  If IsNull(Me.Bemerkungen) Or Me.Bemerkungen = "" Then
    Me.Bemerkungen.ControlTipText = strText
  Else
    Me.Bemerkungen.ControlTipText = ""
  End If
End Sub
```

Listing 441: Eine QuickInfo anzeigen

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.

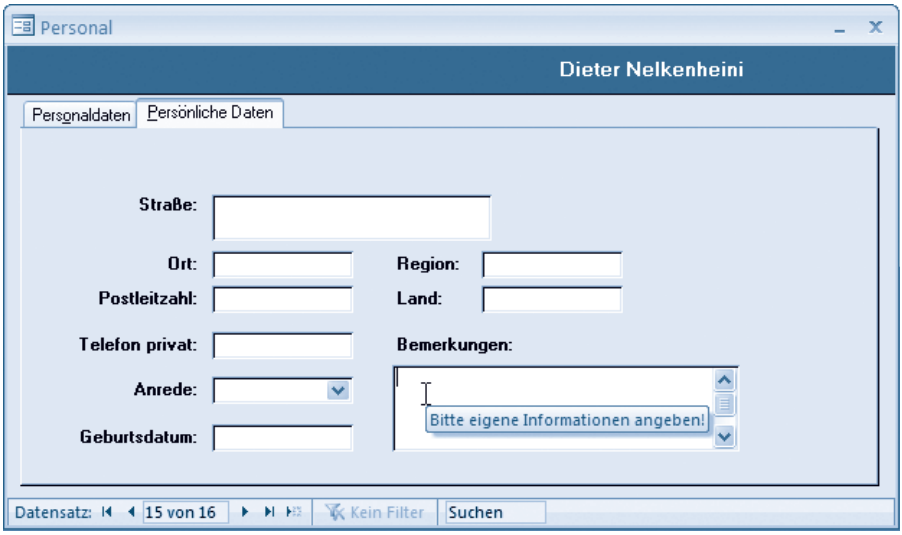


Abbildung 305: QuickInfo bei Feldeintritt anzeigen

Das Ereignis Steuerelement_Exit

Das Ereignis `Exit` tritt unmittelbar bevor ein Steuerelement den Fokus an ein anderes Steuerelement desselben Formulars abgibt auf. Dieses Ereignis wirkt sich nicht auf Kontrollkästchen, Optionsfelder oder Umschaltflächen in einer Optionsgruppe aus.

Die Reihenfolge der Ereignisse

Wie Sie im bisherigen Verlauf gesehen haben, reagieren viele Ereignisse ähnlich. Der einzige Unterschied zwischen einzelnen Ereignissen besteht darin, dass die Ereignisse in einer bestimmten Reihenfolge nacheinander ablaufen. Bei manchen Ereignissen haben Sie die Möglichkeit, Aktionen abzubrechen. Diese Ereignisse haben dann in ihrer Kopfzeile immer das Argument `Cancel`, das Sie auf den Wert `False` setzen können, um das verbundene Ereignis abzubrechen.

348 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, Bei Laden, Bei Größenänderung, Bei Aktivierung, Beim Anzeigen

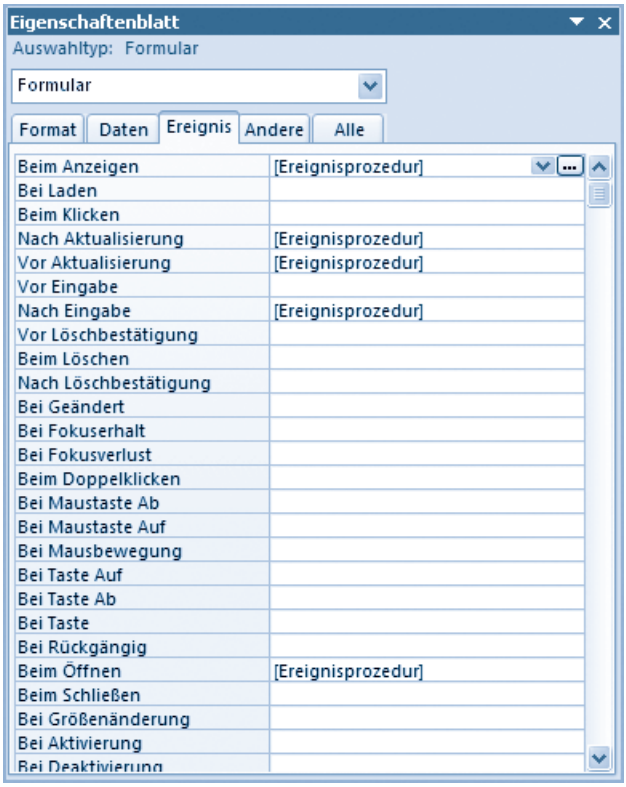


Abbildung 306: Formular-Ereignisse über das Eigenschaftenfenster einstellen

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

349 Aktivierreihenfolge bei Steuerelementen

Wenn Sie beispielsweise über die Taste Tab 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

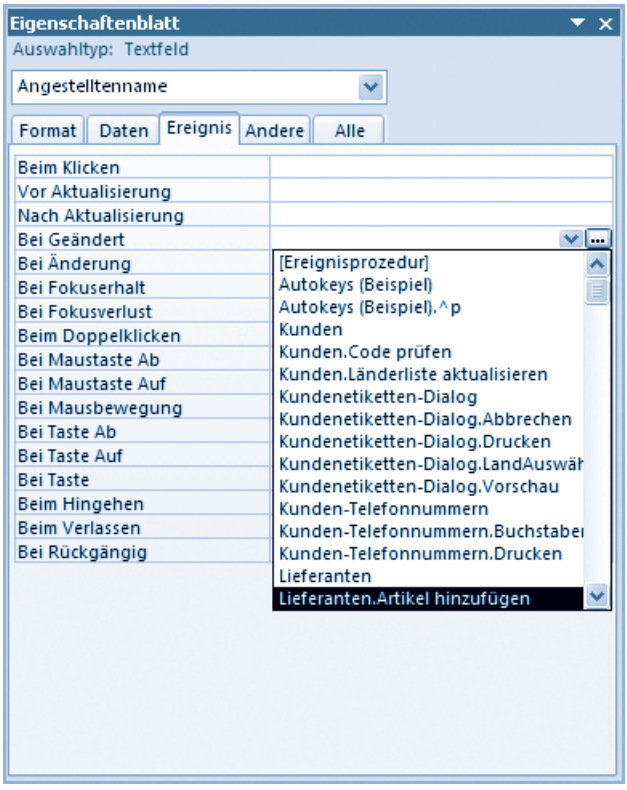


Abbildung 307: Die Steuerelement-Ereignisse

350 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 SPEICHERN aus dem Menü DATENSÄTZE speichern, tritt zunächst das `BeforeUpdate`-Ereignis des Steuerelements und unmittelbar danach das `BeforeUpdate`-Ereignis des Formulars ein.

351 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 Tab 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 & Security

Über die VBE-Programmierung können Sie direkt auf Module, Prozeduren, Funktionen, 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 dazu einsetzen, eine Datenbank von Modulen, Ereignissen und Formularen zu befreien. Eine weitere Einsatzmöglichkeit ist das gezielte Bestücken von Datenbanken mit bestimmten Prozeduren 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.

Quellcode & Datenbanken schützen

Wenn es Ihnen darum geht, Ihren Quellcode zu schützen, dann haben Sie dazu mehrere Möglichkeiten:

1. Hinterlegen eines Kennwortes, 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 Sie Ihre Datenbank.

352 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.

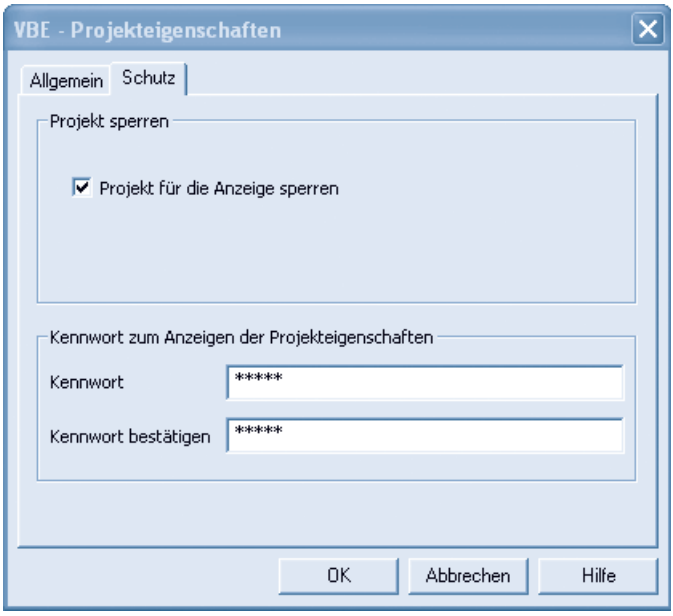


Abbildung 308: Kennwort festlegen

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.

353 Datenbank ohne Quellcode speichern

Wenn Sie sich entschließen, eine Datenbank ohne Quellcode zu speichern, oder liegt der Quellcode kompiliert vor, können Sie bei Access 2003 Ihre Datenbank im MDE-Format oder bei Access 2007 im ACCDE-Format abspeichern. Speichern Sie eine Datenbank in diesem Format, 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- bzw. ACCDE-Datei

einzufügen. Selbst das Einfügen von Modulen ist hierbei nicht möglich. Aus einer MDE- bzw. ACCDE-Datei können auch keine Tabellen, Abfragen, Formulare oder Berichte importiert bzw. exportiert werden, sodass Sie auch hierbei auf der sicheren Seite sind.

Um eine MDE- oder ACCDE-Datei aus Ihrer Datenbank zu erzeugen, befolgen Sie die nächsten Arbeitsschritte:

1. Stellen Sie vorher sicher, dass kein anderer Anwender diese Datenbank im Zugriff hat.
2. Rufen Sie bei Access 2003 aus dem Menü EXTRAS den Befehl DATENBANK-DIENSTPROGRAMME/MDE-DATEI ERSTELLEN auf. Bei Access 2007 wählen Sie auf der Registerseite DATENBANKTOOLS aus der Gruppe DATENBANKTOOLS das Element ACCDE ERSTELLEN.
3. Wählen Sie bei Access 2003 im Dialog ALS MDE ZU SPEICHERNDE DATENBANK eine Datenbank aus, die Sie umwandeln möchten. Bei Access müssen Sie den Dateinamen angeben, unter dem Sie die aktuell geöffnete Datenbank abspeichern möchten.
4. Bestätigen Sie diesen Vorgang bei Access 2003 mit Klick auf MDE ERSTELLEN bei Access 2007 mit einem Klick auf die Schaltfläche SPEICHERN.

354 Datenbanken verschlüsseln

Möchten Sie eine Datenbank vor dem unbefugten Zugriff durch andere Anwender schützen, haben Sie die Möglichkeit, die Datenbank zu verschlüsseln. Prinzipiell besteht ja, bei einer nur durch ein Kennwort geschützten Datenbank, die Möglichkeit, durch Verwendung von externen Programmen, im einfachsten Fall einem simplen Editor, die rohen Daten der Datenbankdatei zu lesen und so, zumindest teilweise, an den Inhalt der Datenbank zu gelangen. Durch die Verschlüsselung wird die Datenbank durch diese externen Programme unlesbar und so gegen ein unbefugtes Anzeigen oder Verwenden geschützt. Sie können die Datenbankdatei natürlich trotzdem in einem Editor öffnen. Sie werden dann jedoch keinen lesbaren Text vorfinden.

Wie in *Abschnitt 356* noch beschrieben wird, sind seit Access 2007 in dem neuen ACCDB-Format die Funktionen Kennwort festlegen und Verschlüsselung miteinander kombiniert. Die folgende Beschreibung trifft demnach nur zu, wenn Sie eine Datei im MDB-Format vorliegen haben.

Generell muss die Datenbank auch durch ein Kennwort geschützt werden, damit die Verschlüsselung im MDB-Format überhaupt etwas bringt. Dies wird in *Abschnitt 356* beschrieben. Um nun zunächst aus Ihrer geöffneten Datenbank eine neue, verschlüsselte Datenbankdatei zu erzeugen, befolgen Sie die nächsten Arbeitsschritte:

1. Sorgen Sie dafür, dass die Datenbank, die Sie verschlüsseln möchten, momentan von niemandem geöffnet ist.
2. Wählen Sie bei Access 2003 aus dem Menü EXTRAS den Befehl SICHERHEIT/DATENBANK VER-/ENTSCHLÜSSELN. Bei Access 2007 wählen Sie auf der Registerkarte DATENBANKTOOLS aus der Gruppe DATENBANKTOOLS das Element DATENBANK VER-/ENTSCHLÜSSELN.
3. Geben Sie im angezeigten Dialog den Namen der neuen verschlüsselten Datei an.
4. Klicken Sie auf die Schaltfläche SPEICHERN.

Beachten Sie, dass Sie nach Durchführung der Verschlüsselung zwei Dateien vorliegen haben. Zum einen Ihre originale Datenbank in unverschlüsselter Form zum anderen die neue, verschlüsselte Variante. Sie können eine verschlüsselte Datei auch wieder entschlüsseln. Die Arbeitsschritte entsprechen denen der Verschlüsselung.

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 ihre Größe daher nicht.

355 Access-Lösung mithilfe von Startparametern absichern

Startoptionen ermöglichen es Ihnen, den Zugriff auf bestimmte Standardmenüs und Symbolleisten, auf das Datenbankfenster sowie auf Spezialtasten weitreichend einzuschränken.

Im folgenden Beispiel werden Sie das Formular ARTIKEL gleich nach dem Start der Datenbank einblenden und bestimmte Aktionen verhindern. Die folgende Beschreibung der Einstellung der Startoptionen bezieht sich auf die Access-Version 2007. Bei früheren Access-Versionen ist die Einstellung jedoch ähnlich.

1. Klicken Sie auf die OFFICE-SCHALTFLÄCHE und anschließend, im sich öffnenden OFFICE-MENÜ, auf die Schaltfläche ACCESS-OPTIONEN. Bei früheren Access-Versionen wählen Sie aus dem Menü EXTRAS den Befehl START.
2. Der Dialog ACCESS-OPTIONEN wird angezeigt. Wählen Sie in der Navigationsleiste des Dialogs den Punkt AKTUELLE DATENBANK.

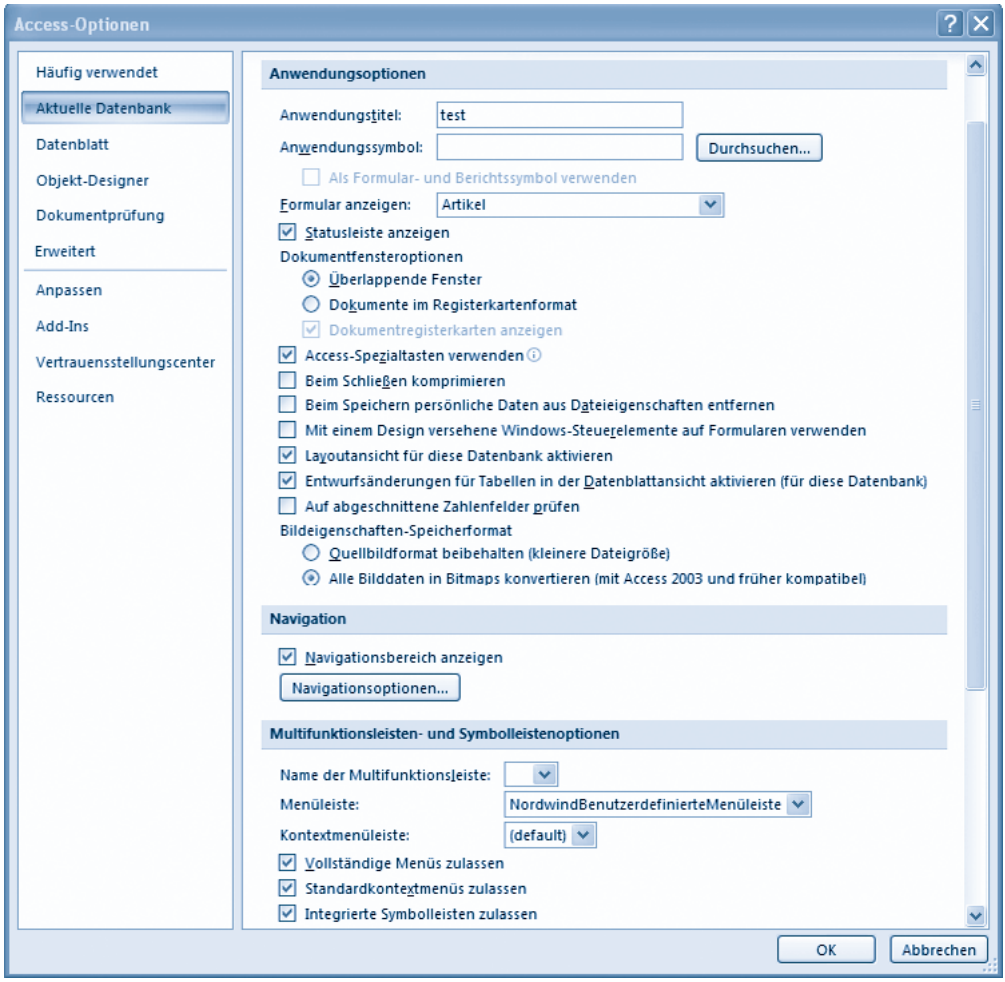


Abbildung 309: Der Dialog Access-Optionen

3. Im Drop-down-Feld FORMULAR ANZEIGEN stellen Sie das Formular ARTIKEL ein.
4. Klicken Sie im Feld MENÜLEISTE auf den Namen der gewünschten Menüleiste.
5. Deaktivieren Sie die Kontrollkästchen VOLLSTÄNDIGE MENÜS ZULASSEN, STANDARD-KONTEXTMENÜS ZULASSEN, NAVIGATIONSBEREICH ANZEIGEN, INTEGRIERTE SYMBOLLEISTEN ZULASSEN und ACCESS-SPEZIALTASTEN VERWENDEN.
6. Bestätigen Sie Ihre Einstellung mit OK.
7. Schließen Sie jetzt Ihre Datenbank.

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.

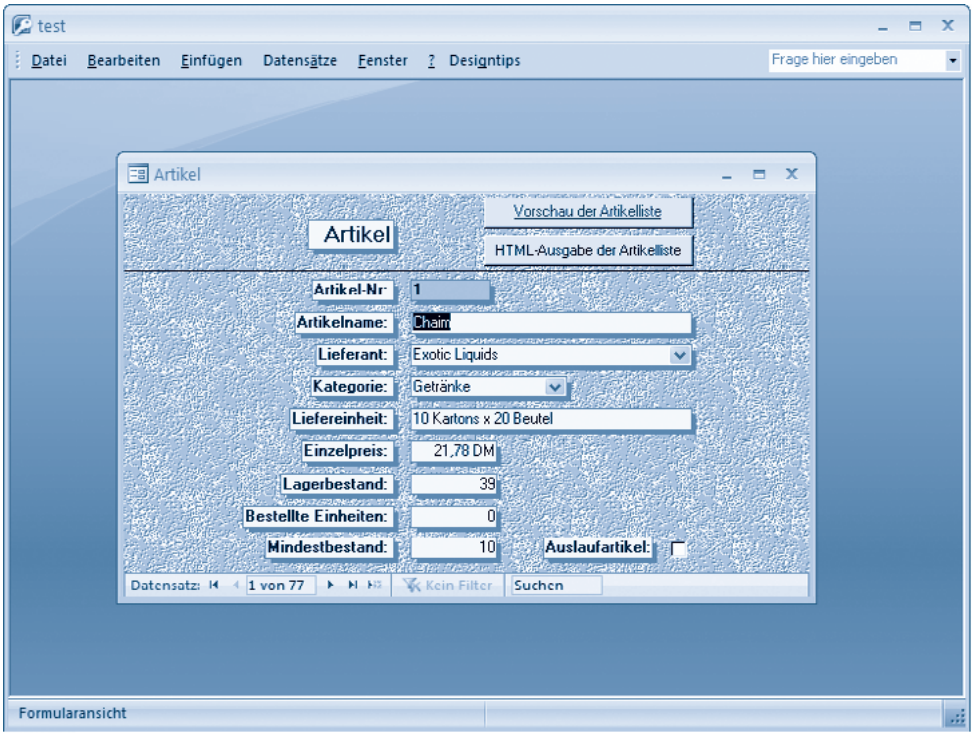


Abbildung 310: Bestimmte Standardelemente fehlen

Der Anwender kann ab sofort lediglich das Formular ARTIKEL bedienen. Alle anderen Aktionen sind hier tabu.

Wenn Sie beim Öffnen einer so geschützten Datenbank die Taste Umschalt gedrückt halten, umgehen Sie die eingestellten Startoptionen. Möchten Sie diese Möglichkeit zusätzlich ausschließen, können Sie dies prinzipiell über einen VBA-Befehl durch Setzen der Eigenschaft `AllowBypassKey` auf den Wert `False` erreichen. Sie müssten dann einen Mechanismus implementieren, wie Sie Ihre Datenbank durch diese Eigenschaft verriegeln, aber auch wieder entriegeln können. Vergessen Sie Letzteres und haben die zuvor beschriebenen Einstellungen in den Startoptionen getätigt, sind Sie Opfer Ihres eigenen Sicherheitssystems geworden. Mit anderen Worten, Sie können die von Ihnen vorgenommene Verriegelung dann selbst nicht mehr rückgängig machen. Von diesem Verfahren ist daher prinzipiell abzuraten.

356 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, müssen Sie sicherstellen, dass die Datenbank nicht gerade von einem weiteren Anwender geöffnet ist. Dazu muss die Datenbank von Ihnen im EXKLUSIVMODUS geöffnet werden. Sonst ist das Setzen eines Zugangskennworts nicht möglich. Haben Sie die Daten-

bank nicht im EXKLUSIVMODUS geöffnet und versuchen, ein Kennwort festzulegen, erhalten Sie eine Fehlermeldung.

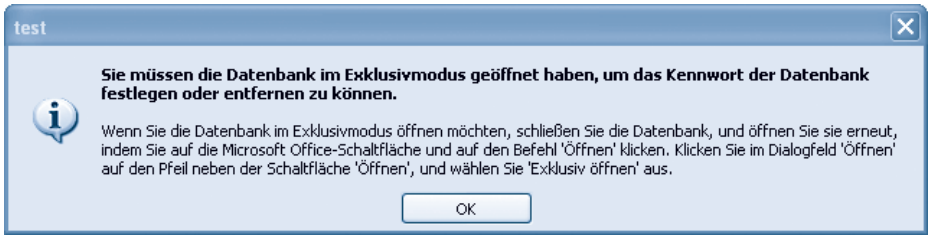


Abbildung 311: Zur Festlegung eines Kennworts muss die Datenbank im Exklusivmodus geöffnet sein.

Um eine Datenbank im EXKLUSIVMODUS zu öffnen und ein Zugangskennwort festzulegen, gehen Sie wie folgt vor. Die Beschreibung bezieht sich hier wieder auf Access 2007. Bei früheren Access-Version ist das Prinzip jedoch ähnlich:

1. Schließen Sie die Datenbank.
2. Klicken Sie auf die OFFICE-Schaltfläche und den Befehl ÖFFNEN. Es wird der Dialog ÖFFNEN angezeigt.
3. Markieren Sie die zu öffnende Datenbank, und klicken Sie auf den Pfeil neben der Schaltfläche ÖFFNEN. Wählen Sie EXCLUSIV ÖFFNEN aus.

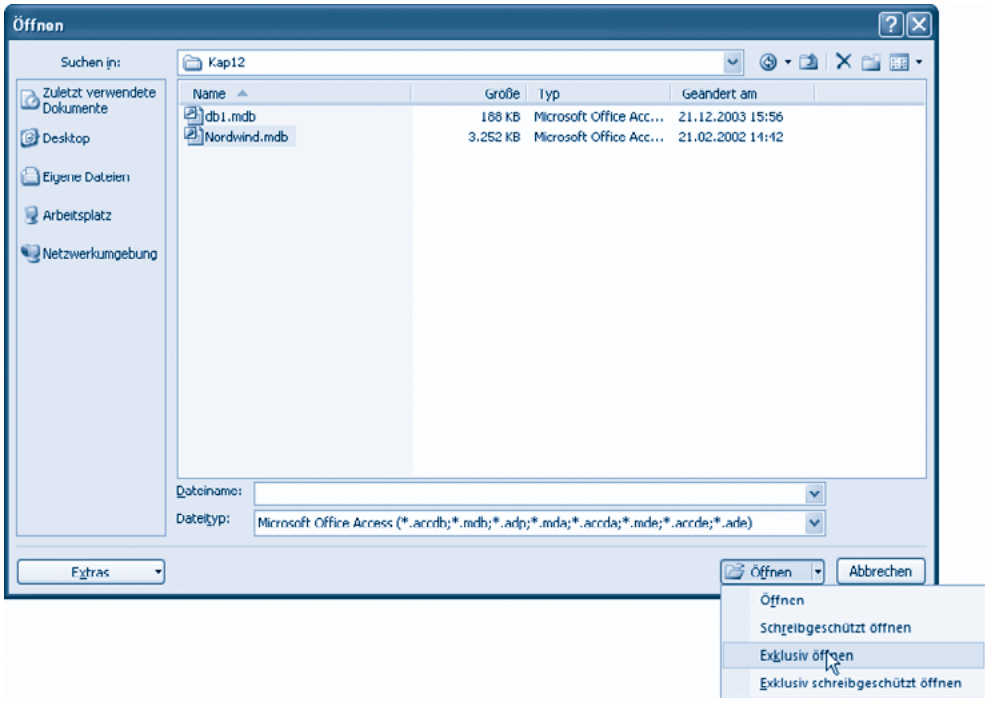


Abbildung 312: Die Datenbank im Exklusivmodus öffnen

4. Der nächste Schritt hängt davon ab, ob Sie eine Datenbank im Access 2007-Format (ACCDB) oder eine Datenbank im alten Format (MDB) im sogenannten Kompatibilitätsmodus geöffnet haben. Haben Sie eine Datenbank im alten Format geöffnet, wählen Sie auf der Registerkarte DATENBANKTOOLS aus der Gruppe DATENBANKTOOLS das Element DATENBANKKENNWORT FESTLEGEN. Haben Sie eine Datenbank im ACCDB-Format geöffnet, wählen Sie aus der Gruppe DATENBANKTOOLS das Element MIT KENNWORT VERSCHLÜSSELN.

Abbildung 313: Kennwort festlegen

5. Geben Sie im Dialogfeld DATENBANKKENNWORT FESTLEGEN ein Kennwort ein und bestätigen Sie dieses.
6. 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.

Microsoft hat im neuen ACCDB-Format das Festlegen eines Kennworts sinnvollerweise mit einer gleichzeitigen Verschlüsselung der Datenbank kombiniert. Beim alten MDB-Format waren Kennwort festlegen und verschlüsseln zwei separate Schritte.

Geschützte Datenbank per VBA öffnen (DAO)

Im Zusammenhang mit dem Öffnen geschützter Datenbanken besteht außerdem die interessante Möglichkeit, diese Aufgabe über eine VBA-Prozedur auszuführen. Die Prozedur aus *Listing 442* 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.

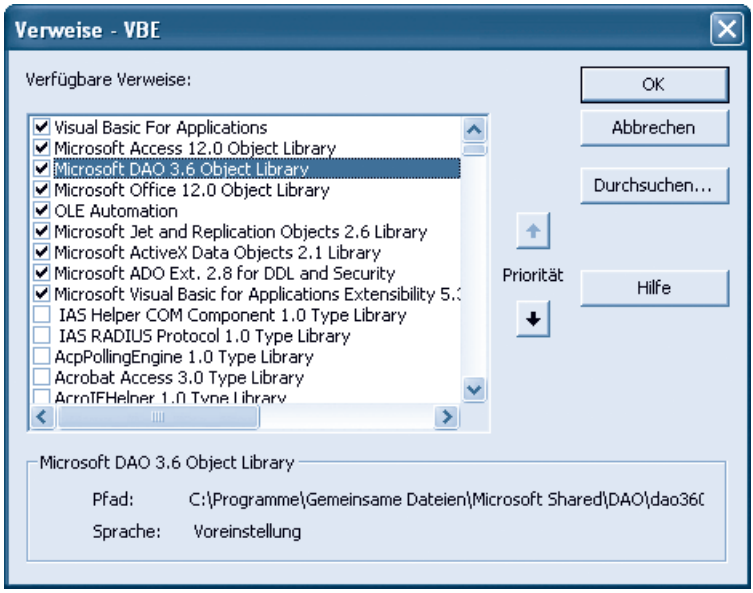


Abbildung 314: Die DAO-Bibliothek aktivieren

3. Aktivieren Sie die Bibliothek MICROSOFT DAO 3.6 OBJECT LIBRARY.
4. Bestätigen Sie Ihre Einstellung mit OK.

Erfassen Sie jetzt die Prozedur aus *Listing 442*:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=======================================================

Sub DatenbankMitKennwortÖffnen()
  Dim objAcc As Object
  Dim strPass As String
  Dim dbs As Database

  Set objAcc = CreateObject("Access.Application.12")
  strPass = ";PWD=test;"
  On Error GoTo Fehler
  Set dbs = objAcc.DBEngine.OpenDatabase _
    ("C:\Eigene Dateien\Nordwind.mdb", False, False, strPass)

  objAcc.OpenCurrentDatabase "C:\Eigene Dateien\Nordwind.mdb"
  dbs.Close
```

Listing 442: Geschützte Datenbank öffnen (DAO)

```
  Set dbs = Nothing
  Exit Sub

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

Listing 442: Geschützte Datenbank öffnen (DAO) (Forts.)

Im ersten Schritt erstellen Sie mithilfe der Methode `CreateObject` ein neues Access-Objekt. Die Versionsnummer, die Sie im Parameter `"Access.Application.12"` übergeben, müssen Sie der von Ihnen verwendeten Access-Version anpassen. Mit der Nummer `12` geben Sie z. B. an, dass Sie ein Access 2007-Objekt erzeugen möchten. 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 `OpenDatabase` einsetzen, um eine angegebene Datenbank in einem `Workspace`-Objekt zu öffnen. Dabei wird ein Verweis auf das `Database`-Objekt zurückgegeben, das diese Datenbank repräsentiert. Übergeben Sie dieser Methode als Argument das Kennwort, das 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 443* wird die Datenbank NORDWIND.MDB im Hintergrund über die MICROSOFT JET OLEDB 4.0 ENGINE geöffnet. Die Tabelle ARTIKEL wird geöffnet. Anschließend werden die Artikelnamen ausgelesen. Tabelle und Datenbank werden danach wieder geschlossen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub DatenbankÖffnen()
  Dim conn As ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = New ADODB.Connection

  With conn
    .Provider = "Microsoft.Jet.OLEDB.4.0"
```

Listing 443: Geschützte Datenbank öffnen (ADO)

```
    .Properties("Jet OLEDB:Database Password") = "test"
    .Mode = adModeReadWrite
    .Open "C:\Eigene Dateien\Nordwind.mdb"
  End With

  Set rst = New ADODB.Recordset
  rst.CursorType = adOpenDynamic
  rst.LockType = adLockPessimistic
  rst.Open "Artikel", conn, , , adCmdTable

  Do While Not rst.EOF
    Debug.Print rst!Artikelname
    rst.MoveNext
  Loop

  rst.Close
  conn.Close
  Set conn = Nothing
  Set rst = Nothing
  Exit Sub

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

Listing 443: Geschützte Datenbank öffnen (ADO) (Forts.)

Über die Eigenschaft `Provider` geben Sie den Namen des Providers für das `Connection`-Objekt an. Im Beispiel verwenden Sie, wie bereits gesagt, die MICROSOFT JET OLEDB 4.0 ENGINE. Über die `Properties`-Auflistung übergeben Sie der Verbindung das Kennwort der Datenbank. Über die Eigenschaft `Mode` können Sie die Berechtigungen für das Ändern von Daten im `Connection`-Objekt `conn` bestimmen. Dabei können Sie die folgenden Konstanten aus *Tabelle 135* einsetzen:

Konstante	Beschreibung
`adModeUnknown`	Diese Standardeinstellung gibt an, dass die Berechtigungen noch nicht festgelegt wurden oder bestimmt werden können.
`adModeRead`	Gibt schreibgeschützte Berechtigungen an.
`adModeWrite`	Gibt Nur-Lese-Berechtigungen an.
`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.

Tabelle 135: Die Konstanten der Eigenschaft Mode

Konstante	Beschreibung
adModeShareExclusive	Verhindert das Öffnen einer Verbindung durch andere Personen.
adModeShareDenyNone	Verhindert das Öffnen einer Verbindung mit beliebigen Berechtigungen durch andere Personen.

Tabelle 135: 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 136* 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 derer Daten gesucht und Berichte generiert werden können. Von anderen Benutzern vorgenommene Zusätze, Änderungen oder Löschvorgänge können nicht angezeigt werden.

Tabelle 136: Die Konstanten der Eigenschaft CursorType

Über die Eigenschaft `LockType` legen Sie die Art der Sperrung fest, die während des Bearbeitens auf Datensätze angewendet wird. Auch bei dieser Eigenschaft können Sie die Sperrung über Konstanten festlegen. *Tabelle 137* zeigt die Konstanten, die Sie verwenden 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 137: 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, in der Sie Zeile für Zeile der Tabelle aufrufen. Innerhalb der Schleife geben Sie die Artikelnamen der Tabelle ARTIKEL im Direktbereich von Access aus. Rufen Sie die Methode MoveNext auf, um jeweils zum nächsten Datensatz zu gelangen. Wenn die Eigenschaft EOF den Wert True zurückliefert, ist der letzte Datensatz erreicht.

Schließen Sie danach die Tabelle und die Datenbank über die Methode Close und heben die Objektverknüpfung auf.

357 Datenbankkennwort ändern

Im nächsten Beispiel ändern Sie ein Kennwort für eine Datenbank über eine VBA-Prozedur. 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 die Prozedur aus *Listing 444*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=======================================================

Sub KennwortÄndern()
  Dim dbs As DAO.Database
  Dim strK As String
```

Listing 444: Datenbankkennwort ändern (DAO)

```
  Dim strAltPwd As String
  Dim strNeuPwd As String

  strK = ";pwd=test"
  strAltPwd = "test"
  strNeuPwd = "testNeu"

  On Error GoTo Fehler
  Set dbs = OpenDatabase _
    (Name:="C:\Eigene Dateien\Nordwind.mdb", _
    Options:=True, ReadOnly:=False, Connect:=strK)

  With dbs
    .NewPassword strAltPwd, strNeuPwd
    .Close
  End With

  Set dbs = Nothing
  Exit Sub

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

Listing 444: 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 danach die Methode `NewPassword` ein, um ein neues Kennwort für die Datenbank zu setzen.

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, also einen leeren String übergeben.

Arbeitsgruppen in Access

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 geben.

358 Arbeitsgruppe anlegen

Das Anlegen einer neuen Gruppe kann neben der manuellen Variante auch über eine VBA-Prozedur erledigt werden. Sehen Sie sich dazu die Prozedur aus *Listing 445* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub NeueGruppeErzeugen()
  Dim wrk As Workspace
  Dim grpNeu As Group
  Dim grpNeuT As Group

  Set wrk = DBEngine.Workspaces(0)
  With wrk
    Set grpNeu = _
      .CreateGroup("NeueGruppe", "XYZ09012892")
    .Groups.Append grpNeu
    Set grpNeuT = _
      .Users("Admin").CreateGroup("NeueGruppe")
    .Users("Admin").Groups.Append grpNeuT
   End With
End Sub
```

Listing 445: 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 unter dem Menü EXTRAS und dem Befehl VERWEISE die Bibliothek MICROSOFT ADO EXT 2.5 für Office 2002, EXT 2.7 für Office 2003 bzw. EXT 2.8 für Office 2007 ein. Starten Sie danach die Prozedur aus *Listing 446*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================
```

Listing 446: Neue Arbeitsgruppe anlegen (ADO)

```
Sub GruppeAnlegenADO()
  Dim cat As ADOX.Catalog

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog

  With cat
    .ActiveConnection = CurrentProject.Connection
    .Groups.Append "ATeam"
  End With

  Set cat = Nothing
  Exit Sub

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

Listing 446: Neue Arbeitsgruppe anlegen (ADO) (Forts.)

Erstellen Sie eine neue Gruppe, indem Sie die Methode `Append` einsetzen. Geben Sie dabei den Namen der Gruppe an.

359 Arbeitsgruppe entfernen (DAO)

Um eine Gruppe wieder zu löschen, können Sie die Prozedur aus *Listing 447* verwenden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub GruppeEntfernen()
  Dim wrk As Workspace
  Dim strGruppe As String

  Set wrk = DBEngine.Workspaces(0)
  On Error GoTo Fehler
  strGruppe = wrk.Groups("NeueGruppe").Name
  wrk.Groups.Delete strGruppe
  Exit Sub

Fehler:
  MsgBox "die Gruppe " & strGruppe & _
    " konnte nicht gefunden/gelöscht werden!"
End Sub
```

Listing 447: Arbeitsgruppe löschen (DAO)

Geben Sie den Namen der Gruppe in einer String-Variablen an und übergeben Sie diese an die Methode `Delete`.

Den entsprechenden Code in ADO können Sie dem *Listing 448* entnehmen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=======================================================

Sub GruppeEntfernenADO()
  Dim cat As ADOX.Catalog

  Set cat = New ADOX.Catalog
  On Error GoTo Fehler
  With cat
    .ActiveConnection = CurrentProject.Connection
    .Groups.Delete "ATeam"
  End With
  Set cat = Nothing
  Exit Sub

Fehler:
  MsgBox "Die Gruppe konnte nicht entfernt werden!"
End Sub
```

Listing 448: Arbeitsgruppe löschen (ADO)

Greifen Sie auf das Auflistungsobjekt `Groups` zu. Dort sind alle Gruppen verzeichnet. Rufen Sie die Methode `Delete` des Objekts `Groups` auf, um die Gruppe zu löschen.

360 Benutzer anlegen

Haben Sie die Gruppe angelegt, fügen Sie mit der Prozedur aus *Listing 449* die Benutzer in die Gruppe ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=======================================================

Sub NeuenBenutzerErzeugen()
  Dim wrk As Workspace
  Dim usr As User

  On Error GoTo Fehler
```

Listing 449: Neuen Benutzer anlegen (DAO)

```
  Set wrk = DBEngine(0)
  Set usr = wrk.CreateUser("Walter", "1002", "Test")

  wrk.Users.Append usr
  usr.Groups.Append wrk.CreateGroup("Users")
  Exit Sub

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

Listing 449: 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 die Prozedur aus *Listing 450* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub BenutzerAnlegenADO()
  Dim cat As ADOX.Catalog

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog

  With cat
    .ActiveConnection = CurrentProject.Connection
    .Users.Append "Hero", "Test"
    .Groups("Users").Users.Append "Hero"
  End With

  Set cat = Nothing
  Exit Sub

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

Listing 450: 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, wiederum über die Methode Append, das neue Benutzerkonto an die Standardbenutzergruppe an.

361 Benutzer entfernen

Möchten Sie einen Benutzer per VBA wieder entfernen, starten Sie die Prozedur aus *Listing 451*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=====================================================

Sub BenutzerEntfernen()
  Dim strBenutzer As String
  Dim wrk As Workspace

  On Error GoTo Fehler
  Set wrk = DBEngine(0)
  strBenutzer = wrk.Users("Walter").Name
  wrk.Users.Delete strBenutzer
  Exit Sub

Fehler:
  MsgBox "Der Benutzer konnte nicht gelöscht werden!"
End Sub
```

Listing 451: Benutzer entfernen (DAO)

Mithilfe der Methode Delete löschen Sie einen bereits angelegten Benutzer. Geben Sie dabei den Namen des Benutzers in einem String an.

Die Prozedur in ADO sehen Sie in *Listing 452*:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=====================================================

Sub BenutzerEntfernenADO()
  Dim cat As ADOX.Catalog

  On Error GoTo Fehler
  Set cat = New ADOX.Catalog
```

Listing 452: Benutzer entfernen (ADO)

```
  With cat
    .ActiveConnection = CurrentProject.Connection
    .Users.Delete "Hero"
  End With

  Set cat = Nothing
  Exit Sub

Fehler:
   MsgBox "Der Benutzer konnte nicht entfernt werden!"
End Sub
```

Listing 452: Benutzer entfernen (ADO) (Forts.)

Setzen Sie die Methode `Delete` ein, um einen Benutzer zu löschen. Wenden Sie diese Methode innerhalb des Auflistungsobjektes `Users` an, in dem alle Benutzer verzeichnet sind.

362 Benutzer auslesen

Um zu erfahren, welche Benutzer bereits angelegt wurden, können Sie das Auflistungsobjekt `Groups` anzapfen und die Namen der Benutzer im Direktbereich ausgeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlDAO
'=======================================================

Sub Benutzeranzeigen()
  Dim wrk As Workspace
  Dim usr As User

  Set wrk = DBEngine(0)

  For Each usr In wrk.Users
    Debug.Print usr.Name
  Next usr
End Sub
```

Listing 453: Benutzer auslesen (DAO)

363 Berechtigungen zuweisen

Nachdem Sie sowohl die Gruppe(n) als auch den Benutzer angelegt haben, ordnen Sie die Benutzer einzelnen Gruppen zu. Mithilfe einer Prozedur können Sie gezielt Berechtigungen zuweisen. Im nächsten Beispiel aus *Listing 454* weisen Sie dem Benutzer SCHMIDT eine spezielle Berechtigung für die Tabelle ARTIKEL zu.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlADO
'=======================================================

Sub BerechtigungenHinzufügenADO()
  Dim conn As New ADODB.Connection
  Dim cat As New ADOX.Catalog

  On Error GoTo Fehler
  conn.Provider = "Microsoft.Jet.OLEDB.4.0"
  Set cat.ActiveConnection = CurrentProject.Connection
  cat.Users("Schmidt").SetPermissions "Artikel", _
    adPermObjTable, adAccessSet, adRightInsert
  Exit Sub

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

Listing 454: 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-Objektes an, das Sie schützen möchten.

Beim Argument `ObjectType` setzen Sie eine der in der *Tabelle 138* 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.

Tabelle 138: Die Objekttypen der Methode SetPermissions

Konstante	Beschreibung
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 138: Die Objekttypen der Methode SetPermissions (Forts.)

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 139: 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.

Tabelle 140: Die Rights-Konstanten der Methode SetPermissions

Konstante	Beschreibung
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.
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 140: 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 einzelnen Rechte durch ein Pluszeichen voneinander ab.

```
KDB.Users("Schmidt").SetPermissions "Artikel", adPermObjTable,
adAccessSet, adRightInsert + adRightRead
```

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.
VBProjects-Auflistung	In diesem Auflistungsobjekt sind alle geöffneten Projekte in der Entwicklungsumgebung verzeichnet.

Tabelle 141: Die wichtigsten Objekte in der VBE

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.

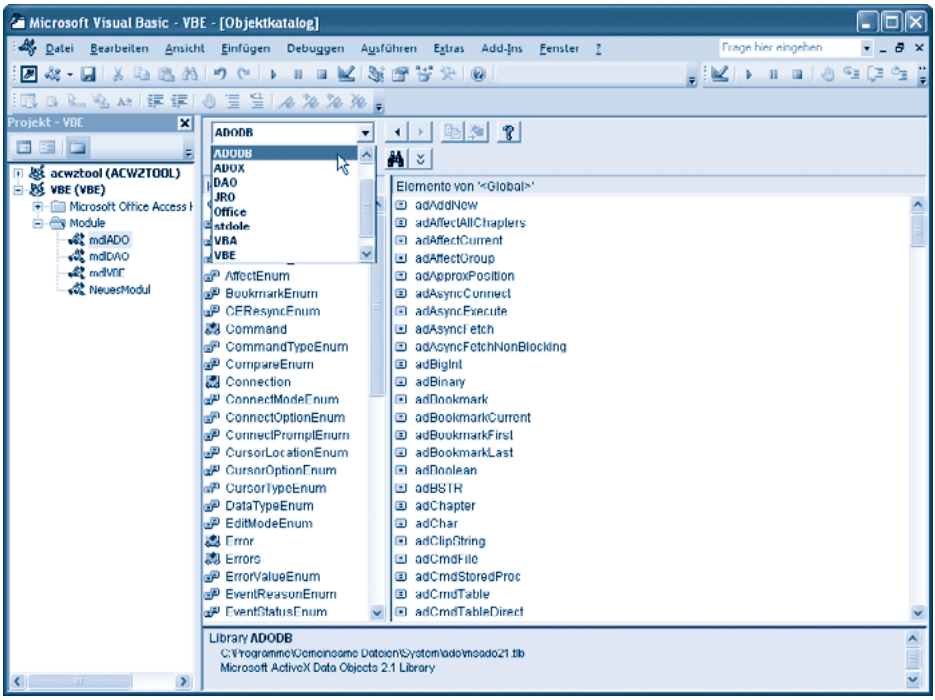


Abbildung 315: Der Objektkatalog gibt Auskunft über die einsetzbaren Befehle einer Bibliothek.

Wählen Sie aus dem Drop-down 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.

364 Die VBE-Bibliothek einbinden

Als wichtigste Bibliothek ist die VBE-Bibliothek zu nennen, die eingebunden werden muss, bevor Sie überhaupt mit der VBE-Programmierung beginnen. Mithilfe der Methoden und Eigenschaften dieser Bibliothek können Sie in der Entwicklungsumgebung »frei« 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.

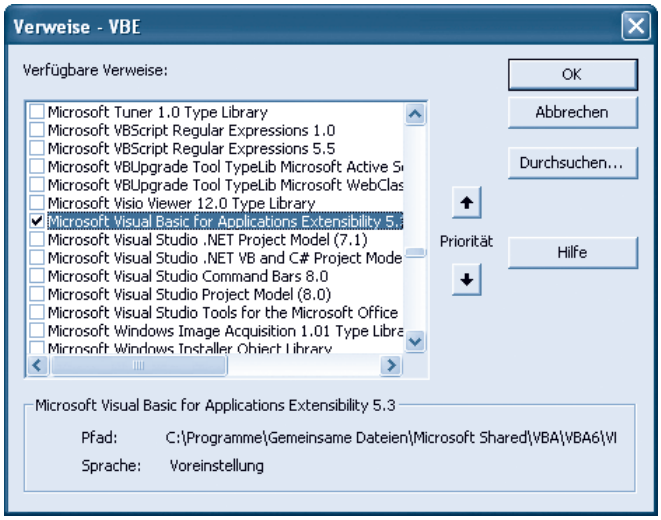


Abbildung 316: Die VBE-Bibliothek einbinden

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 die Prozedur aus *Listing 455* vornehmen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub VBEAktivieren()
  Dim objVbe As Object

  On Error Resume Next
  objVbe = Application.VBE.ActiveVBProject.References. _
```

Listing 455: Die VBE-Bibliothek über eine Prozedur aktivieren

```
    AddFromGuid("{0002E157-0000-0000-C000-000000000046}", 5, 3)
  Set objVbe = Nothing
End Sub
```

Listing 455: Die VBE-Bibliothek über eine Prozedur aktivieren (Forts.)

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, die 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 die Prozedur aus *Listing 456* an, um alle diesbezüglichen Informationen im Direktfenster der Entwicklungsumgebung auszulesen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub VerweiseGUID()
  Dim ref As Reference
```

Listing 456: Die GUIDs der eingebundenen Bibliotheken auslesen

```
  For Each ref In Application.References
    Debug.Print ref.Name
    Debug.Print "GUID" & ref.Guid
    Debug.Print "HauptNr: " & ref.Major
    Debug.Print "NebenNr: " & ref.Minor & vbLf
  Next ref
End Sub
```

Listing 456: Die GUIDs der eingebundenen Bibliotheken auslesen (Forts.)

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.

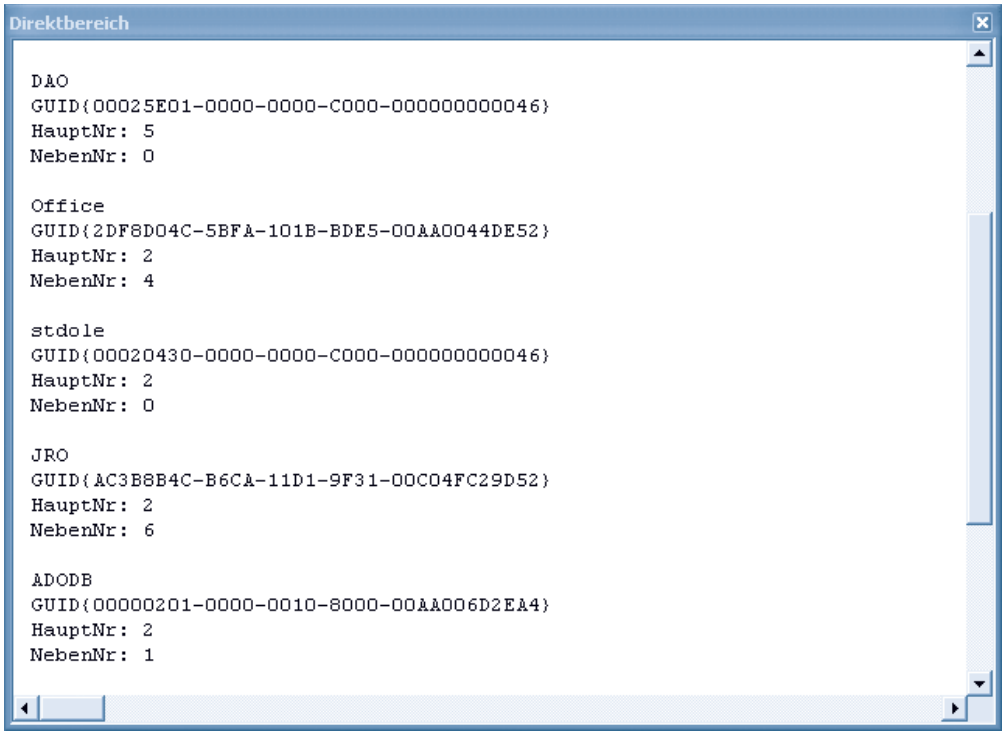


Abbildung 317: Alle GUIDs, Haupt- und Nebennummern wurden ermittelt.

Weitere Bibliotheken einbinden

Mithilfe der Prozedur aus *Listing 456* ist es jetzt nicht mehr schwer, bei Bedarf weitere Bibliotheken auszulesen und eine Prozedur zu schreiben, die die Verweise automatisch setzt.

In der folgenden Prozedur aus *Listing 457* werden die Objektbibliotheken für Microsoft Excel sowie Microsoft Word automatisch gesetzt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub WeitereBibliothekenEinbindenVar1()
  Dim objVbe As Object

  On Error Resume Next
  'Microsoft Excel 2003 einbinden
  objVbe = Application.VBE.ActiveVBProject.References. _
    AddFromGuid("{00020813-0000-0000-C000-000000000046}", 1, 5)

  'Microsoft Word 2003 einbinden
  objVbe = Application.VBE.ActiveVBProject.References. _
    AddFromGuid("{00020905-0000-0000-C000-000000000046}", 8, 3)

  Set objVbe = Nothing
End Sub
```

Listing 457: 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.

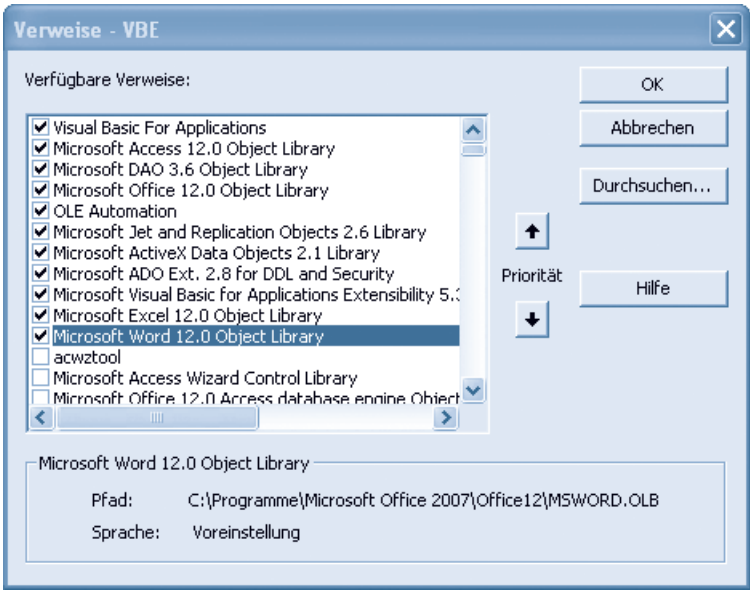


Abbildung 318: Die Bibliotheken für Excel und Word wurden eingebunden.

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 458* werden die beiden Bibliotheken für Excel und Word mithilfe der Methode `AddFromFile` eingebunden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub WeitereBibliothekenEinbindenVar2()
  Dim objVbe As Object

  Set objVbe = Application.VBE.ActiveVBProject.References
  On Error Resume Next
    objVbe.AddFromFile "Excel.exe"
    objVbe.AddFromFile "msword.olb"
  Set objVbe = Nothing
End Sub
```

Listing 458: Bibliotheken einbinden – Variante 2

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 die Prozedur aus *Listing 459*, die die Namen aller Bibliotheken ermittelt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================
```

Listing 459: Die Namen der aktiven Verweise werden ausgelesen.

```
Sub Verweisnamen()
  Dim obj As Object

  For Each obj In Application.VBE.ActiveVBProject.References
    Debug.Print obj.Name
  Next obj
End Sub
```

Listing 459: Die Namen der aktiven Verweise werden ausgelesen. (Forts.)

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.

Abbildung 319: Die Namen der Verweise wurden ermittelt.

Bibliotheken wieder deaktivieren

Für Excel wird der Objektname EXCEL, für Word der Objektname WORD verwendet. Übergeben Sie diese Namen jetzt wie in *Listing 460* gezeigt an die Methode `Remove`, um die beiden Objektbibliotheken wieder zu deaktivieren.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub BibliothekenDeaktivieren()
  Dim objVbe As Object

  On Error Resume Next
  Set objVbe = Application.VBE.ActiveVBProject.References
```

Listing 460: Bibliotheken deaktivieren

```
  objVbe.Remove objVbe("Word")
  objVbe.Remove objVbe("Excel")
  Set objVbe = Nothing
End Sub
```

Listing 460: Bibliotheken deaktivieren (Forts.)

Übergeben Sie der Methode `Remove` die jeweiligen Objektnamen, um die Bibliotheken in der Entwicklungsumgebung zu deaktivieren. Heben Sie danach den Objektverweis wieder auf.

365 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 Prozedurabstürzen kommt.

Die folgende Prozedur aus *Listing 461* listet alle eingebundenen Bibliotheken im Direktfenster der Entwicklungsumgebung auf.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub VerweiseAusgeben()
  Dim ref As Reference

  For Each ref In Application.References
    Debug.Print ref.Name & vbLf & ref.FullPath & vbLf
  Next ref
End Sub
```

Listing 461: 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 dann den Namen des Verweises mithilfe der Eigenschaft `Name` sowie den kompletten Pfad des Verweises über die Eigenschaft `FullPath` im Direktfenster der Entwicklungsumgebung aus.

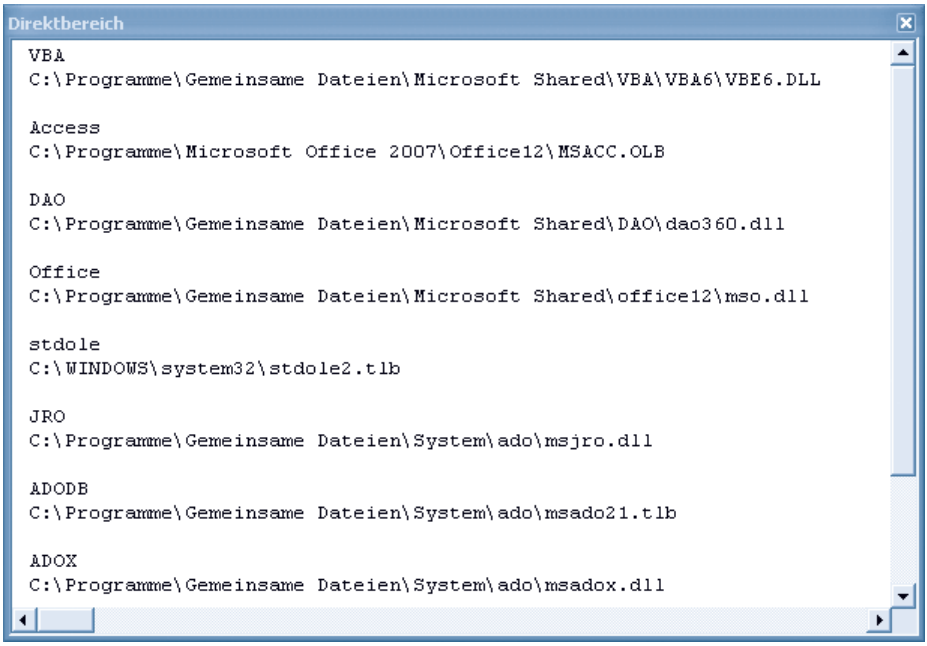


Abbildung 320: Die Namen und Speicherpfade der eingebundenen Bibliotheken ausgeben

366 Auf der Suche nach fehlerhaften Verweisen

Ist eine Bibliothek in der Entwicklungsumgebung nicht verfügbar, dann kann es beim Ausführen von Prozeduren zu Fehlern kommen. Mit der Prozedur aus *Listing 462* können Sie im Vorfeld schon fehlerhafte Verweise aufspüren und im Direktfenster der Entwicklungsumgebung ausgeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub VerweisCheck()
  Dim ref As Reference

  For Each ref In Application.References
    If ref.IsBroken = True Then
      Debug.Print ref.Name & vbLf & ref.FullPath & vbLf
    End If
  Next ref
End Sub
```

Listing 462: 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.

367 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 463* sehen Sie eine Prozedur, die die verschiedenen Komponenten der Datenbank im Direktfenster ausgibt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub KomponentenAusgeben()
  Dim vbc As VBComponent

  For Each vbc In Application.VBE.ActiveVBProject.VBComponents
    Debug.Print "Name der Komponente: " & vbc.Name & _
      vbLf & "Typ der Komponente: " & _
      vbc.Type & vbLf
  Next vbc
  Set vbc = Nothing
End Sub
```

Listing 463: Die einzelnen VB-Komponenten ermitteln

In *Abbildung 321* können Sie sehen, dass jede Komponente ihren eindeutigen Typ aufweist, der über die Eigenschaft `Type` ermittelt werden kann. Dies ermöglicht später das gezielte Ansprechen einzelner Komponenten.

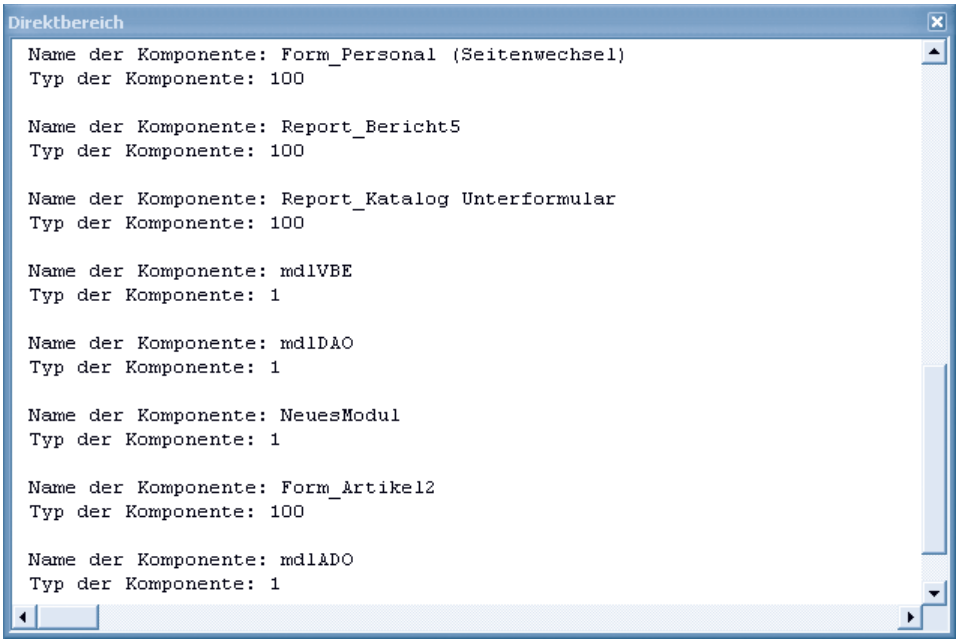


Abbildung 321: Die einzelnen Typen ermitteln

368 Die VBE aufrufen

Standardmäßig drücken Sie die Tastenkombination Alt + F11, um in die Entwicklungsumgebung zu gelangen.

Diesen Vorgang können Sie aber auch durch eine Prozedur ausführen, die Sie im Beispiel aus *Listing 464* sehen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================
Sub VBEAufrufen()
  With Application.VBE.MainWindow
    .SetFocus
    .Visible = True
  End With
End Sub
```

Listing 464: Die Entwicklungsumgebung aufrufen

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.

369 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 eine Prozedur 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 142: 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 143: Die verfügbaren Konstanten der Methode Add

Im folgenden Beispiel aus *Listing 465* wird dem aktuellen VBA-Projekt ein neues Modul mit dem Namen NEUESMODUL hinzugefügt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub ModulEinfügen()
  Dim vbc As VBComponent

  On Error GoTo Fehler
```

Listing 465: Ein neues Modul einfügen

```
  Set vbc = _
    Application.VBE.ActiveVBProject.VBComponents.Add(vbext_ct_StdModule)
  vbc.Name = "NeuesModul"
  Set vbc = Nothing
  Exit Sub

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

Listing 465: Ein neues Modul einfügen (Forts.)

Da es sich bei einem neuen Modul um eine VB-Komponente handelt, deklarieren Sie zu Beginn der Prozedur 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.

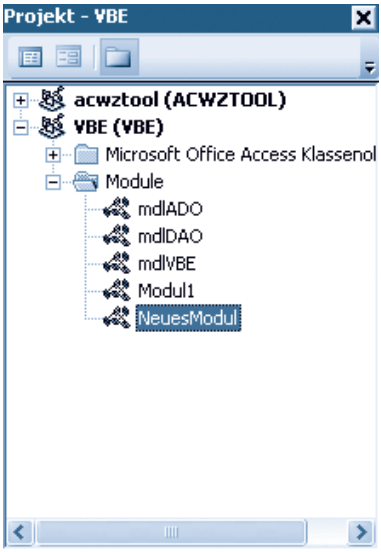


Abbildung 322: Das neue Modul wurde angelegt.

370 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 die folgende Prozedur aus *Listing 466*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub ModulLöschen()
  Dim objVbModul As Object

  On Error GoTo Fehler
  Set objVbModul = Application.VBE.ActiveVBProject

  With objVbModul
    .VBComponents.Remove .VBComponents("NeuesModul")
  End With
  Set objVbModul = Nothing
  Exit Sub

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

Listing 466: Modul löschen

Mit der Methode `Remove` können Sie ein bestimmtes Modul löschen.

371 Alle Module löschen

Die folgende Prozedur aus *Listing 467* geht noch einen Schritt weiter. Sie löscht alle Module der aktiven Datenbank bis auf das MODUL1 aus der Datenbank.

Hinweis Vorsicht bei der Anwendung dieses Prozedur! Danach sind tatsächlich alle Module weg!

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================
```

Listing 467: Alle Module bis auf das Modul1 werden ohne Rückfrage gelöscht.

```
Sub AlleModuleBisAufEinesLöschen()
  Dim vbc As VBComponent

  On Error GoTo Fehler
  With Application.VBE.ActiveVBProject
    For Each vbc In .VBComponents
      If vbc.Type = 1 Or vbc.Type = 2 Then
        If vbc.Name <> "Modul1" Then .VBComponents.Remove vbc
      End If
    Next vbc
  End With
  Exit Sub

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

Listing 467: 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.

372 Modulcheck durchführen

Im letzten Abschnitt haben Sie erfahren, wie Sie ein Modul löschen können. 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 468* durchführen. Dabei wird überprüft, ob sich das Modul mit dem Namen MODUL2 in der aktuell geladenen Datenbank befindet.

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

Listing 468: Überprüfen, ob ein Modul existiert

```
  Else
    MsgBox "Modul existiert nicht!", vbCritical
  End If
End Sub
```

Listing 468: Überprüfen, ob ein Modul existiert (Forts.)

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.

373 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 vorzugeben.

Die Lösung aus *Listing 469* 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
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub ModulDrucken()
  Dim objXlApp As Object

  Application.VBE.ActiveVBProject.VBComponents _
    ("Modul1").Export "c:\Eigene Dateien\Codes.txt"
  Set objXlApp = CreateObject("Excel.Application")

  objXlApp.Workbooks.OpenText "c:\Eigene Dateien\Codes.txt"
  With objXlApp.Activeworkbook.ActiveSheet.Cells
    .Font.Name = "Courier"
    .Font.Size = 8
    .PrintOut
  End With
  objXlApp.Activeworkbook.Close savechanges:=False
  Kill "c:\Eigene Dateien\Codes.txt"
  Set objXlApp = Nothing
End Sub
```

Listing 469: Ein Listing drucken

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 die als Text gespeicherte Codedatei mithilfe der Methode `OpenText`. Formatieren Sie danach alle Zellen der aktiven Tabelle, indem Sie die gewünschte Schriftart und 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 der Prozedur noch die Anweisung `Kill` an, um die Textdatei zu löschen.

374 Prozedur löschen

Möchten Sie aus einem Modul eine bestimmte Prozedur entfernen, dann müssen Sie diese Prozedur zuerst finden. In der folgenden Prozedur aus *Listing 470* wird in MODUL1 nach der Prozedur TEST gesucht. Wenn diese Prozedur gefunden wird, dann soll sie aus dem Modul entfernt werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub ProzedurLöschen()
  Dim objVBKomp As Object
  Dim lngStart As Long
  Dim intAnzahl As Integer

  Set objVBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("Modul1").CodeModule

  On Error GoTo Fehler
  With objVBKomp
    lngStart = .ProcStartLine("Test", vbext_pk_Proc)
    intAnzahl = .ProcCountLines("Test", vbext_pk_Proc)
    .DeleteLines lngStart, intAnzahl
  End With
  Exit Sub

Fehler:
  MsgBox "Eine Prozedur mit diesem Namen gibt es nicht!"
End Sub
```

Listing 470: Prozedur entfernen

Die Eigenschaft ProcStartLine gibt die Zeile zurück, in der die festgelegte Prozedur beginnt. Die Eigenschaft ProcCountLines gibt die Anzahl der Zeilen in der festgelegten Prozedur an. Die Syntaxen dieser Eigenschaften lauten:

```
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 144: 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 145: Die Konstanten des Arguments ProzArt

Bei erfolgreicher Aufspürung der Prozedur wenden Sie die Methode DeleteLines an, um die Prozedur 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 146: Die Argumente der Methode DeleteLines

Für den Fall, dass die gesuchte Prozedur nicht gefunden werden kann, sorgen Sie mithilfe der On Error-Anweisung dafür, dass auch diese Prozedur nicht abstürzt.

375 Prozedur-Check durchführen

Bevor Sie eine Prozedur löschen bzw. auf eine Prozedur in irgendeiner Form zugreifen, können Sie vorab eine Prüfung vornehmen, ob die entsprechende Prozedur überhaupt in der Datenbank existiert. Dazu können Sie eine Funktion schreiben, die so aussehen kann, wie in *Listing 471* dargestellt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 ProzedurDa(ProzedurName As String, _
  ModulName As String) As Boolean

  On Error Resume Next
  If ModulDa(ModulName) = True Then
    ProzedurDa = Application.VBE.ActiveVBProject _
      .VBComponents(ModulName) _
      .CodeModule.ProcStartLine(ProzedurName, vbext_pk_Proc) <> 0
  End If
End Function


Sub ProzedurCheck()
  If ProzedurDa("DatumUndZeit", "Modul1") = True Then
    MsgBox "Prozedur existiert!", vbInformation
  Else
    MsgBox "Prozedur existiert nicht!", vbCritical
  End If
End Sub
```

Listing 471: Existiert eine bestimmte Prozedur in einem Modul?

Übergeben Sie der Funktion `ProzedurDa` den Namen der Prozedur sowie den Namen des Moduls, in dem die Prozedur gespeichert ist. Über die Eigenschaft `ProcStartLine` wird dann in der Funktion versucht, die Prozedur zu finden. Ist die Suche erfolgreich, dann wird eine Länge ungleich null zurückgegeben. Den Abgleich mit dem Modul können Sie in der Beschreibung der Funktion aus *Listing 468* nachlesen.

376 Formular-Ereignisse löschen

Neben Prozeduren, die in Standardmodulen oder Klassenmodulen gespeichert sind, gibt es auch noch Ereignisprozeduren, die sich direkt hinter Formularen oder Berichten befinden.

Um alle Prozeduren, die hinter einem Formular liegen, zu entfernen, starten Sie die Prozedur aus *Listing 472*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularCodeLöschen()
  Dim objVBKomp As Object

  Set objVBKomp = _
    Application.VBE.ActiveVBProject. _
      VBComponents("Form_Artikel").CodeModule

  On Error GoTo Fehler
  With objVBKomp
    .DeleteLines 1, objVBKomp.CountOfLines
  End With
  Set objVBKomp = Nothing
  Exit Sub

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

Listing 472: Alle Prozeduren 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 472* 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?

In der Prozedur aus *Listing 473* werden alle Steuerelemente des Formulars ARTIKEL2 gelöscht.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub EinzelneSteuerelementeEnfernen()
  Dim astrNamen(50) As Variant
  Dim ctl As Control
  Dim intZ As Integer
  Dim frm As Form

  DoCmd.OpenForm "Artikel2", acDesign

  Set frm = Forms!Artikel2

  intZ = 1
  For Each ctl In frm.Controls
    astrNamen(intZ) = ctl.Name
    intZ = intZ + 1
    Debug.Print astrNamen(intZ)
  Next ctl

  On Error Resume Next
  For intZ = LBound(astrNamen) To UBound(astrNamen)
    DeleteControl "Artikel2", astrNamen(intZ)
  Next intZ
  Set frm = Nothing
End Sub
```

Listing 473: Alle Steuerelemente eines Formulars entfernen

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 `astrNamen`. In einer zweiten Schleife, die beginnend beim ersten Eintrag (`Lbound`) bis zum letzten Eintrag im Datenfeld (`Ubound`) läuft, werden die einzelnen Steuerelemente über die Methode `DeleteControl` gelöscht.

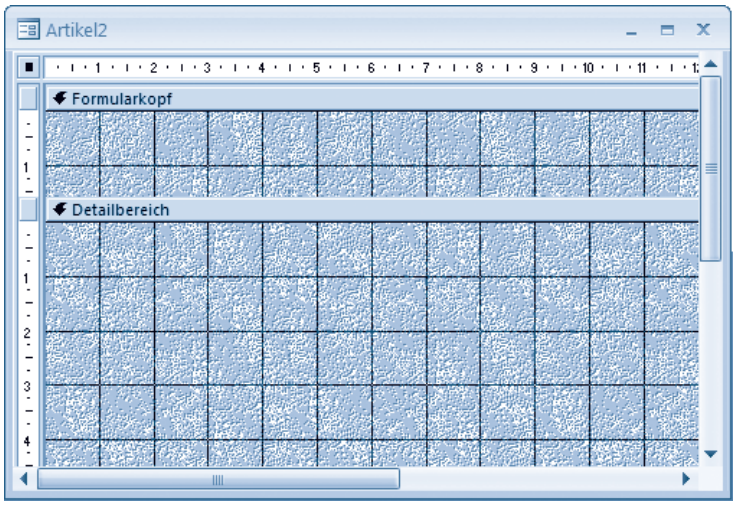


Abbildung 323: Das Formular wurde geputzt.

Formular löschen

Soll das ganze Formular gelöscht werden, dann wenden Sie die Prozedur aus *Listing 474* an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularLöschen()
  DoCmd.DeleteObject acForm, "Artikel2"
End Sub
```

Listing 474: Ein Formular löschen

Setzen Sie die Methode `DeleteObject` ein, um das Formular zu löschen.

377 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 475* wird nach der Methode `ProcCountLines` im Modul `mdlVBE` gesucht. Ist die Suche erfolgreich verlaufen, dann soll der entsprechende Befehl im Quellcode markiert werden.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub BefehlFinden()
  Dim objVBKomp As Object
  Dim lngZeileStart As Long
  Dim lngZeileEnde As Long
  Dim lngSpalteStart As Long
  Dim lngSpalteEnde As Long
  Dim bool As Boolean

  Set objVBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("mdlVBE").CodeModule

  With objVBKomp
    bool = .Find("ProcCountLines", lngZeileStart, lngSpalteStart, _
      lngZeileEnde, lngSpalteEnde)
    If bool = True Then
      .CodePane.SetSelection lngZeileStart, lngSpalteStart, _
        lngZeileEnde, lngSpalteEnde
    Else
      MsgBox "Der Befehl konnte nicht gefunden werden!"
    End If
  End With
  Set objVBKomp = Nothing
  Exit Sub

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

Listing 475: 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. Sie finden in diesem Beispiel den Text `ProcCountLines` mindestens einmal im Modul `mdlVBE`, nämlich mindestens in der Prozedur aus *Listing 475*.

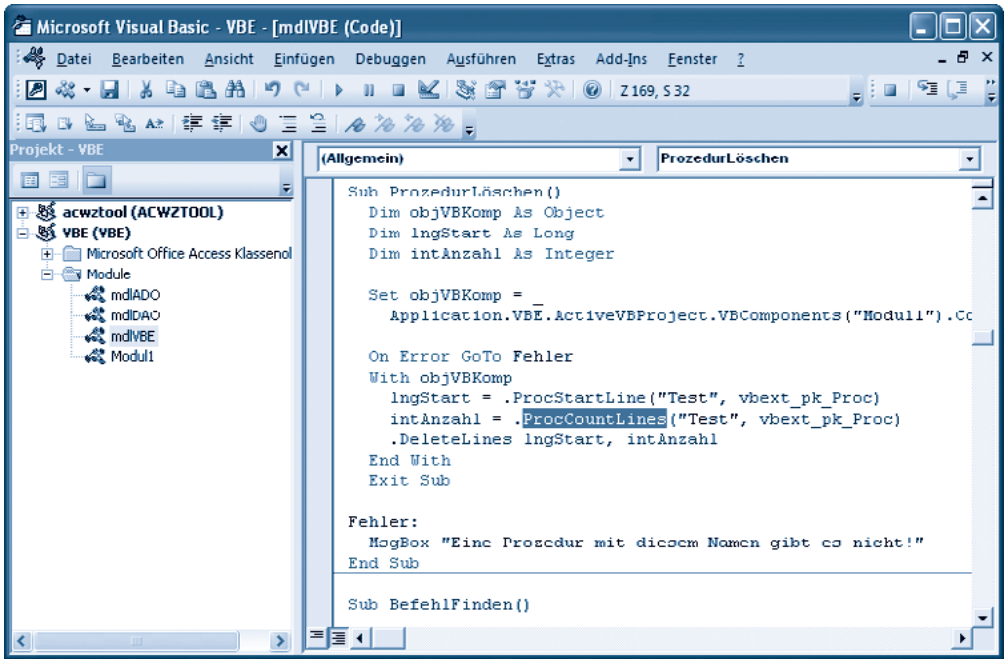


Abbildung 324: Der Befehl wurde gefunden und markiert.

378 Texte ersetzen

Beim folgenden Beispiel aus *Listing 476* soll eine E-Mail-Adresse in einem bestimmten Modul ausgetauscht werden. Da diese Mailadresse dort mehrfach vorkommen kann, muss die Lösung dynamisch programmiert werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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 cdm As CodeModule
  Dim bool As Boolean
```

Listing 476: Textteile im Code suchen und ersetzen

```
  Const SuchText = "Machero@aol.com"
  Const ErsetzText = "Held-office@t-online.de"

  On Error GoTo Fehler
  Set cdm = Application.VBE.ActiveVBProject.VBComponents _
    ("mdlVBE").CodeModule
  bool = cdm.Find(SuchText, lngZeileStart + 1, _
    lngSpalteStart, lngZeileEnde, lngSpalteEnde, False, False)

  While bool = True
    strAlt = cdm.Lines(lngZeileEnde, 1)
    strNeu = Replace(strAlt, SuchText, ErsetzText)
    cdm.ReplaceLine lngZeileEnde, strNeu
    lngZeileStart = lngZeileEnde + 1
    lngZeileEnde = cdm.CountOfLines
    lngSpalteStart = 1
    lngSpalteEnde = 250
    bool = cdm.Find(SuchText, lngZeileStart + 1, _
      lngSpalteStart, lngZeileEnde, lngSpalteEnde)
  Wend
  Exit Sub

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

Listing 476: Textteile im Code suchen und ersetzen (Forts.)

Deklarieren Sie zu Beginn der Prozedur zunächst den Text, der gesucht 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, sodass feststeht, wo sich der gesuchte Text 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.

Tabelle 147: Die Argumente der Methode Find

Teil	Beschreibung
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.
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 147: 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 148: 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 149: 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 die Suche so lange in einer Schleife, bis kein Suchergebnis mehr gefunden werden kann. In diesem Fall wird die boolesche Variable `bool` auf den Wert `False` gesetzt, was den Abbruch der Schleife zur Folge hat.

379 Prozeduren eines Moduls auflisten

Um einen Überblick über alle in einem Modul enthaltenen Prozeduren zu gewinnen, können Sie die Prozedur aus *Listing 477* einsetzen:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub ProzedurenAuflisten()
  Dim objVBKomp As Object
  Dim lngStart As Long

  Set objVBKomp = Application.VBE.ActiveVBProject _
    .VBComponents("mdlVBE").CodeModule

  With objVBKomp
    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 objVBKomp = Nothing
End Sub
```

Listing 477: Eine Prozedurliste erstellen

Zu Beginn der Prozedur erfahren Sie mithilfe der Eigenschaft `CountOfDeclarationLines` die Anzahl der Codezeilen im Deklarationsabschnitt des Code-Moduls. Mithilfe der Eigenschaft `ProcOfLine` können Sie den Namen der jeweiligen Prozedur im Modul MDLVBE ermitteln. Diese Eigenschaft benötigt zwei Argumente: Im ersten Argument geben Sie die Startzeile an, ab der nach dem Prozedurnamen gesucht werden soll. Im zweiten Argument wird die Art der zu suchenden Prozedur festgelegt. Über die Konstante `vbext_pk_Proc` wird dabei fixiert, dass Prozeduren und Funktionen ermittelt werden sollen. Über die Eigenschaft `CountOfLines` ermitteln Sie die Gesamtzahl der Codezeilen in einem Modul.

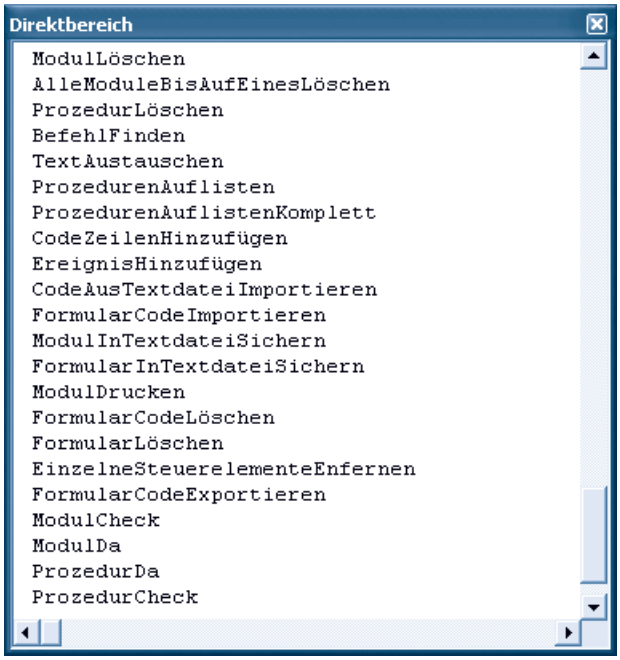


Abbildung 325: Alle Prozeduren aus dem Modul1

Und jetzt alle ...

Um die Suche auf alle Module der Datenbank auszuweiten, starten Sie die Prozedur aus *Listing 478*:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub ProzedurenAuflistenKomplett()
  Dim vbc As VBComponent
```

Listing 478: Eine komplette Prozedurliste erstellen

```
  Dim obj As Object
  Dim lngStart As Long

  For Each vbc In Application.VBE.ActiveVBProject.VBComponents
  If vbc.Type = 1 Then
    Set obj = vbc.CodeModule
    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 vbc
  Set obj = Nothing
  Set vbc = Nothing
End Sub
```

Listing 478: Eine komplette Prozedurliste 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.

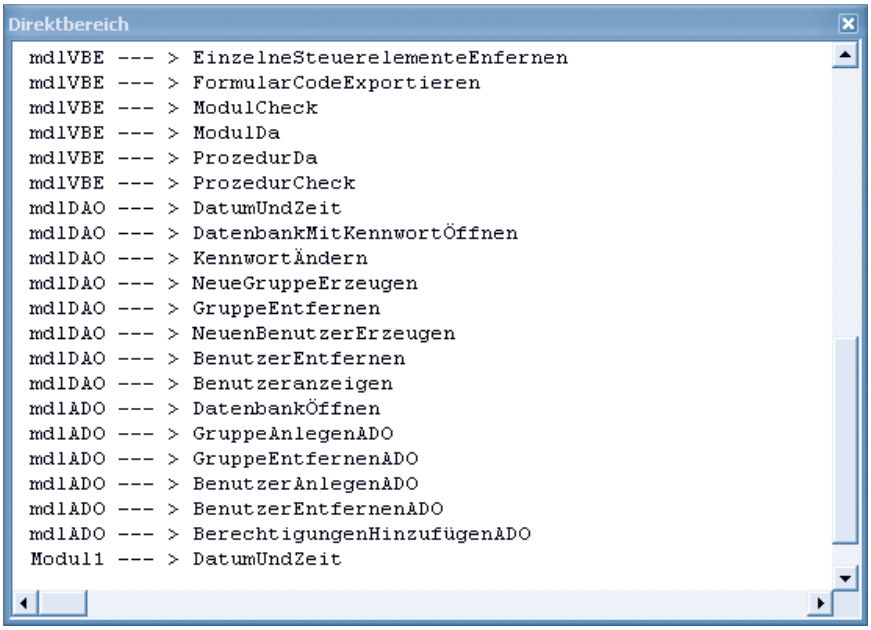


Abbildung 326: Die Prozedurliste mit Modulbezug wurde erstellt.

380 Prozeduren einfügen

Wenn Sie Prozeduren per Code in eine Datenbank einfügen möchten, dann stehen Ihnen zwei Möglichkeiten zur Verfügung:

- zeilenweises Einfügen in das Modul,
- Import einer Textdatei, die die komplette Prozedur 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 479* an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub ProzedurZeilenHinzufügen()
  Dim cdm As CodeModule
  Dim intZ As Integer

  On Error GoTo Fehler
  Set cdm = Application.VBE.ActiveVBProject.VBComponents _
   ("Modul1").CodeModule

   With cdm
     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 479: Eine Prozedur einfügen – Variante 1

In der Prozedur wird ein Code zeilenweise übertragen. Die Eigenschaft `CountOfLines` ermittelt, wie viele Codezeilen im Modul bereits enthalten sind, und addiert den Wert `1` dazu. Diese Maßnahme ist notwendig, um eventuell bereits bestehende Prozeduren nicht zu überschreiben. Die Konstante `vbLF` in der Prozedur sorgt jeweils für den Zeilenvorschub. Die Eigenschaft `Now` gibt das aktuelle Datum inklusive der Uhrzeit aus.

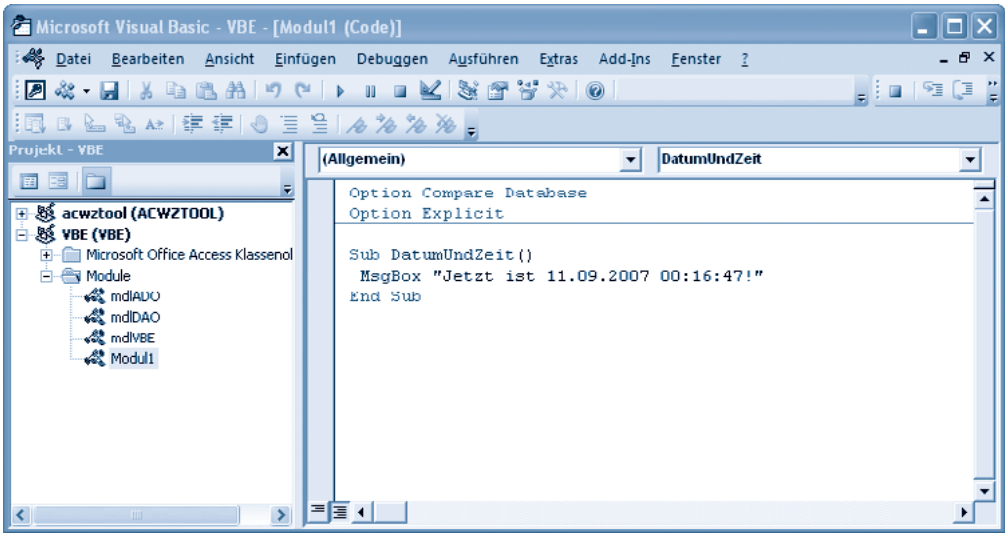


Abbildung 327: Eine kürzere Prozedur per Quellcode einfügen

Variante 2 – die Lösung über AddFromFile

Selbstverständlich ist die Variante 1 nur bei kürzeren Prozeduren geeignet. Bei der folgenden zweiten Variante speichern Sie eine Prozedur in einer Textdatei. Legen Sie zu diesem Zweck über den Editor aus dem Windows-Zubehör eine Textdatei mit dem Namen CODE.TXT an und geben dann folgenden Code aus *Abbildung 328* ein.

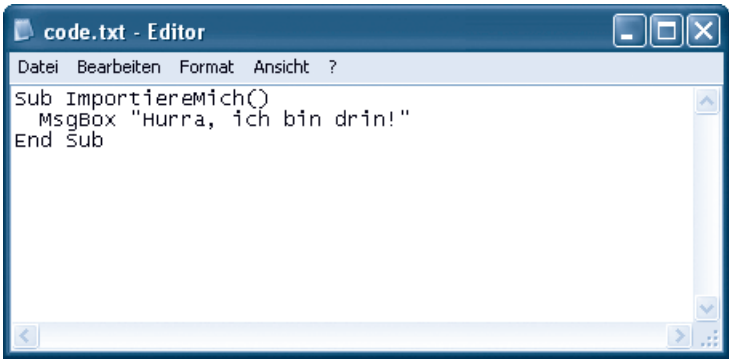


Abbildung 328: Diese Prozedur soll importiert werden.

Um den Programmcode aus der Textdatei CODE.TXT in das Modul NEUESMODUL einzufügen, starten Sie die Prozedur aus *Listing 480*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub CodeAusTextdateiImportieren()
  Dim cdm As CodeModule
  Const strImportDatei = "C:\Eigene Dateien\code.txt"

  On Error GoTo Fehler
  Set cdm = Application.VBE.ActiveVBProject _
    .VBComponents("NeuesModul").CodeModule

  With cdm
    .AddFromFile strImportDatei
  End With
  Set cdm = Nothing
  Exit Sub

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

Listing 480: Eine Prozedur einfügen – Variante 2

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 `CodeModule`-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 150: Die Argumente der Methode AddFromFile

Nach dem Import der Textdatei kann der Objektverweis wieder aufgehoben werden und die Prozedur über die Anweisung `Exit Sub` auf direktem Wege verlassen werden.

381 Formularcode importieren

Das manuelle Einstellen von einzelnen Ereignissen, das Sie im *Kapitel Die Ereignisprogrammierung* nachschlagen können, ist eine Lösung.

Eine zweite Lösung ist es, Ereignisse in einem Formular einzustellen und diese dabei aus einer Textdatei zu importieren.

Im Beispiel aus *Listing 481* wird die Textdatei ARTIKELFORMULAR.TXT direkt hinter das Formular ARTIKEL importiert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=====================================================

Sub FormularCodeImportieren()
  Dim cdm As CodeModule
  Const strImportDatei = "C:\Eigenen Dateien\Artikelformular.txt"

  On Error GoTo Fehler
  Set cdm = Application.VBE.ActiveVBProject.VBComponents _
    ("Form_Artikel").CodeModule

  With cdm
    .AddFromFile strImportDatei
  End With
  Set cdm = Nothing
  Exit Sub

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

Listing 481: 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.

382 Formular-Ereignisprozedur einstellen

Zum Einfügen einer einzelnen Ereignisprozedur bietet sich auch die Methode `CreateEventProc` an.

Beim folgenden Beispiel aus *Listing 482* wird dem Formular ARTIKEL2 eine Ereignisprozedur hinzugefügt. Dabei soll beim Starten des Formulars das aktuelle Datum am Bildschirm angezeigt werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub EreignisProzedurHinzufügen()
  Dim cdm As CodeModule
  Dim intZ As Integer

  On Error GoTo Fehler
  Set cdm = Application.VBE.ActiveVBProject _
    .VBComponents("Form_Artikel2").CodeModule

  With cdm
    intZ = .CountOfLines + 1
    intZ = .CreateEventProc("Load", "Form") + 1
      .InsertLines intZ, " Msgbox ""Hallo " _
      & Date & "!"" "
  End With
  Set cdm = Nothing
  Exit Sub

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

Listing 482: Ein Ereignis hinzufügen

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 151: Die Argumente der Methode CreateEventProc

Über die Methode `InsertLines` fügen Sie danach innerhalb des Ereignisses die gewünschten Zeilen ein.

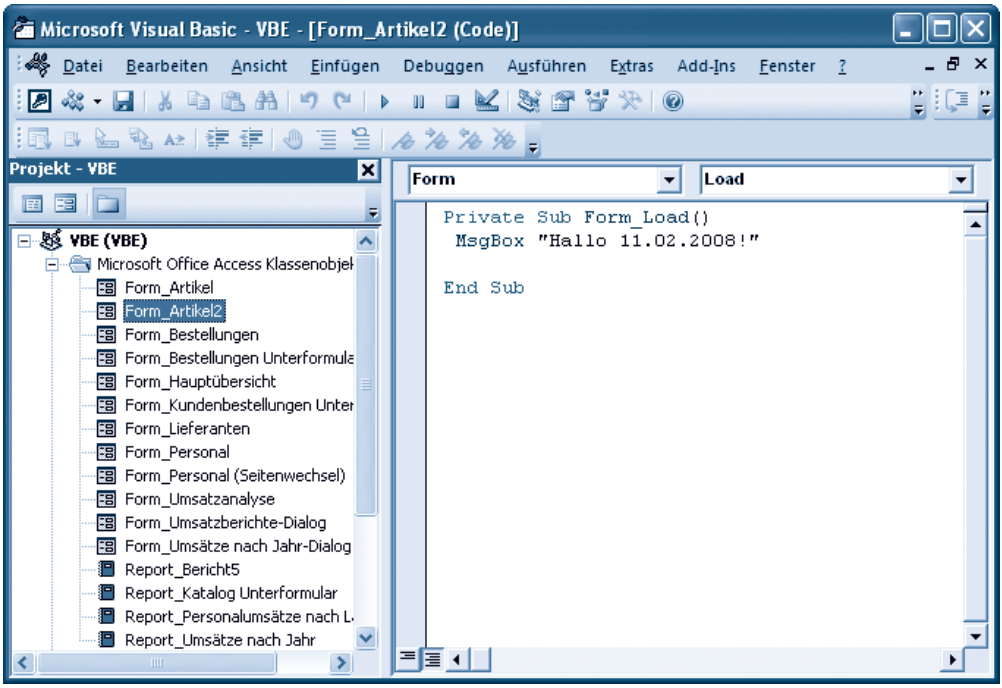


Abbildung 329: Die Ereignisprozedur wurde eingefügt.

383 Prozeduren exportieren

Neben dem Import von Textdateien in Modulen ist in Access auch eine Exportfunktion für Module 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 152: Die Argumente der Methode Export

Die Prozedur aus *Listing 483* exportiert alle Prozeduren im Modul MDLVBE in eine Textdatei und speichert diese im Verzeichnis C:\EIGENE DATEIEN unter dem Namen SICHERUNG.TXT.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub ModulInTextdateiSichern()
  Dim objVBKomp As Object
  Const strExportDatei = "C:\Eigene Dateien\Sicherung.txt"

  On Error GoTo Fehler
  Set objVBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("mdlVBE")

  objVBKomp.Export strExportDatei
  Set objVBKomp = Nothing
  Exit Sub

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

Listing 483: Ein Modul in eine Textdatei exportieren

Mit der Methode `Export` sichern Sie alle Prozeduren aus MODUL1 als Textdatei.

384 Formularcodes sichern

Hinter den einzelnen Formularen kann sich Quellcode in Form von Ereignisprozeduren befinden. Diese Codes können Sie über die Methode `Export` ebenso in eine Textdatei überführen.

Die folgende Prozedur aus *Listing 484* exportiert alle Ereignisprozeduren, die sich hinter dem Formular ARTIKEL befinden, und speichert sie in der Datei ARTIKELFORMULAR.TXT ab.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  VBE.mdb
' Modul      mdlVBE
'=======================================================

Sub FormularInTextdateiSichern()
  Dim objVBKomp As Object
  Const strExportDatei = "C:\Eigene Dateien\ArtikelFormular.txt"

  On Error GoTo Fehler
```

Listing 484: Auch Formularcodes können exportiert werden.

```
  Set objVBKomp = _
    Application.VBE.ActiveVBProject.VBComponents("Form_Artikel")

  With objVBKomp
    .Export strExportDatei
  End With
  Set objVBKomp = Nothing
  Exit Sub

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

Listing 484: Auch Formularcodes können exportiert werden. (Forts.)

Geben Sie den Codenamen des Formulars bei der Anweisung `Set` an.

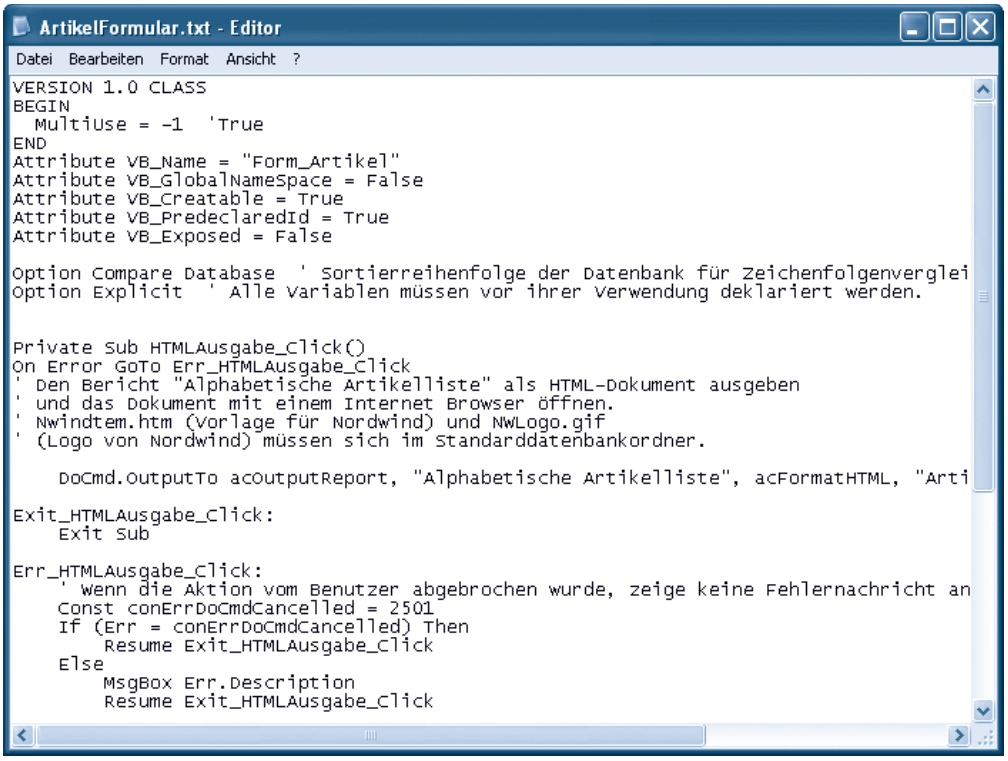


Abbildung 330: Die Ereignisse des Formulars Artikel wurden gesichert.

Die Alternative

Eine alternative Möglichkeit, die ohne den Einsatz von VBE auskommt, können Sie in *Listing 485* sehen. Dabei werden alle Formulare der Datenbank, die Quellcode enthalten, als separate Textdateien gespeichert.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 485: 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 den Code zu exportieren. Dabei speichern Sie die Dateien im Verzeichnis C:\Eigene Dateien unter dem jeweiligen Formularnamen und der Endung .txt. Schließen Sie danach das jeweilige Formular über die Methode `Close`.

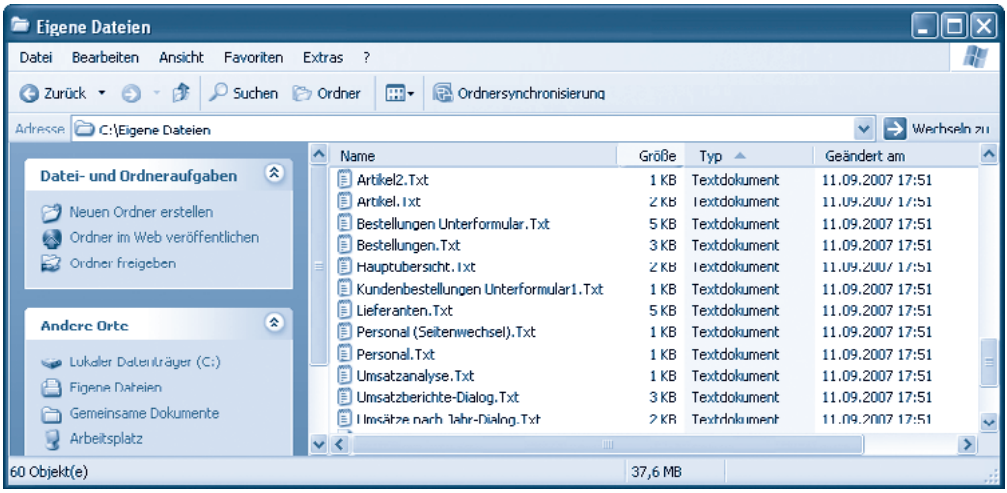


Abbildung 331: Alle »Code-Formulare« wurden exportiert.

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 können somit die Stärke einer jeden Office-Anwendung einsetzen.

Early- 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.

385 Early-Binding

Beim Early-Binding werden bereits vor Ausführung des VBA-Codes 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 Ausführung des VBA-Codes zur Verfügung.

Bei dieser Methode ist zu bemerken, dass sie die schnellere ist. Des Weiteren stehen Ihnen schon beim Editieren des Quellcodes die neuen Methoden und Eigenschaften sowie die Online-Hilfe zu der eingebundenen Bibliothek zur Verfügung.

386 Late-Binding

Beim Late-Binding wird zur Laufzeit einer Prozedur 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 486* wird ein Word-Objekt erstellt und danach ein Dokument geöffnet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================
```

Listing 486: Eine Word-Sitzung wird erstellt.

```
Sub LateBinding()
  Dim objWordApp As Object

  Set objWordApp = CreateObject("Word.Application")
  objWordApp.Documents.Open "C:\Eigene Dateien\Makrostartfehler.doc"
  objWordApp.Visible = True

  'weitere Befehle

  Set objWordApp = Nothing
End Sub
```

Listing 486: Eine Word-Sitzung wird erstellt. (Forts.)

Access & Outlook

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.

387 Ist Outlook bereits gestartet?

Im ersten Beispiel zum Zusammenspiel von Access mit Outlook schreiben Sie die Funktion aus *Listing 487*, über die Sie feststellen können, ob Outlook bereits gestartet ist.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Function OutlookGeöffnet() As Boolean
  Dim objOutApp As Outlook.Application

  On Error Resume Next
  Set objOutApp = GetObject(, "Outlook.Application")

  If Err.Number = 0 Then
    OutlookGeöffnet = True
  Else
    OutlookGeöffnet = False
  End If
End Function
```

Listing 487: 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 die aufrufende Prozedur aus *Listing 488* zurück.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub StartOutlook()
  Dim objOutApp As Object
  Dim objOrdner As Object
  Dim objNameSpace As Object
  Dim objMails As Object

  If Not OutlookGeöffnet Then
    Set objOutApp = CreateObject("Outlook.Application")
    Set objNameSpace = objOutApp.GetNamespace("MAPI")
    Set objOrdner = objNameSpace.GetDefaultFolder(olFolderInbox)
    objOrdner.Display
  End If
End Sub
```

Listing 488: Outlook starten oder nicht

In der Prozedur aus *Listing 488* wurde das Late-Binding durchgeführt und es wurden einige Objektvariablen deklariert. Meldet die Funktion `OutlookGeöffnet` den Rückgabewert `False`, dann ist Outlook noch nicht geöffnet. In diesem Fall rufen Sie die Funktion `CreateObject` auf, 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 über die Methode `GetDefaultFolder` den Ordner, den Sie anzeigen möchten.

Abbildung 332: 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 POSTEINGANG-Ordner anzeigen.

Weitere Ordner können Sie aus der folgenden *Tabelle 153* ablesen.

Konstante	Ordner
`olFolderCalendar`	KALENDER
`olFolderContacts`	KONTAKTE
`olFolderDeletedItems`	GELÖSCHTE OBJEKTE
`olFolderDrafts`	ENTWÜRFE
`olFolderInbox`	POSTEINGANG
`olFolderJournal`	JOURNAL
`olFolderNotes`	NOTIZEN
`olFolderOutbox`	POSTAUSGANG
`olFolderSendMail`	GESENDETE OBJEKTE
`olFolderTasks`	AUFGABEN
`olFolderJunk`	JUNK-E-MAILS

Tabelle 153: Die wichtigsten Ordner von Outlook

388 E-Mails verschicken

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 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`: 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.

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-Objektes 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 Text-Editor, beispielsweise mit dem Microsoft Editor (Notepad) im Zubehör-Programm 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-Nachricht aufgenommen werden sollen. Die Empfängernamen in diesem Argument müssen durch Semikola (;) oder durch jenes Listentrennzeichen voneinan-

der 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`.

Beim 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-Nachricht. 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-Nachricht aufgenommen.

Mithilfe des Arguments `NachrichtBearbeiten` 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.

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 489* wird eine Text-E-Mail an mehrere Personen versendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=====================================================

Sub TextEMailVersenden()
  DoCmd.SendObject , , , "Held-office@t-online.de", _
    "Machero@aol.com", , "Feedback erwünscht", _
    "Sehr geehrter Herr Müller," & vbCrLf & vbCrLf & _
    "bitte senden Sie mir ein Feedback " & _
    "zu meiner Anfrage vom 03.07.2008." & _
    vbCrLf & vbCrLf & _
    "Beste Grüße," & vbCrLf & _
    "Bernd Held", True
End Sub
```

Listing 489: E-Mail über eine Prozedur versenden

Die ersten drei Argumente der Methode `SendObject` werden nicht benötigt. Geben Sie daher drei Kommas nacheinander ein. Geben Sie dann erst die Empfänger-E-Mail-Adresse und nachfolgend die E-Mail-Adresse an, an die die E-Mail als Kopie gehen soll. Geben Sie dann den Titel sowie den eigentlichen E-Mail-Text an. Um Zeilen-

umbrüche im E-Mail-Text einzufügen, arbeiten Sie mit der Konstanten `vbCrLf`. Damit wird ein Zeilenumbruch in der E-Mail erzeugt. Setzen Sie das Argument `Nachricht Bearbeiten` auf den Wert `True`, um die E-Mail vor dem Versenden für eine weitere Bearbeitung zu öffnen. Diese Funktionalität ist übrigens nicht an Outlook gekoppelt, sondern es wird vielmehr die auf Ihrem PC registrierte E-Mail-Anwendung verwendet. In *Abbildung 333* sehen Sie beispielsweise ein automatisch geöffnetes Fenster der Thunderbird E-Mail-Client-Anwendung.

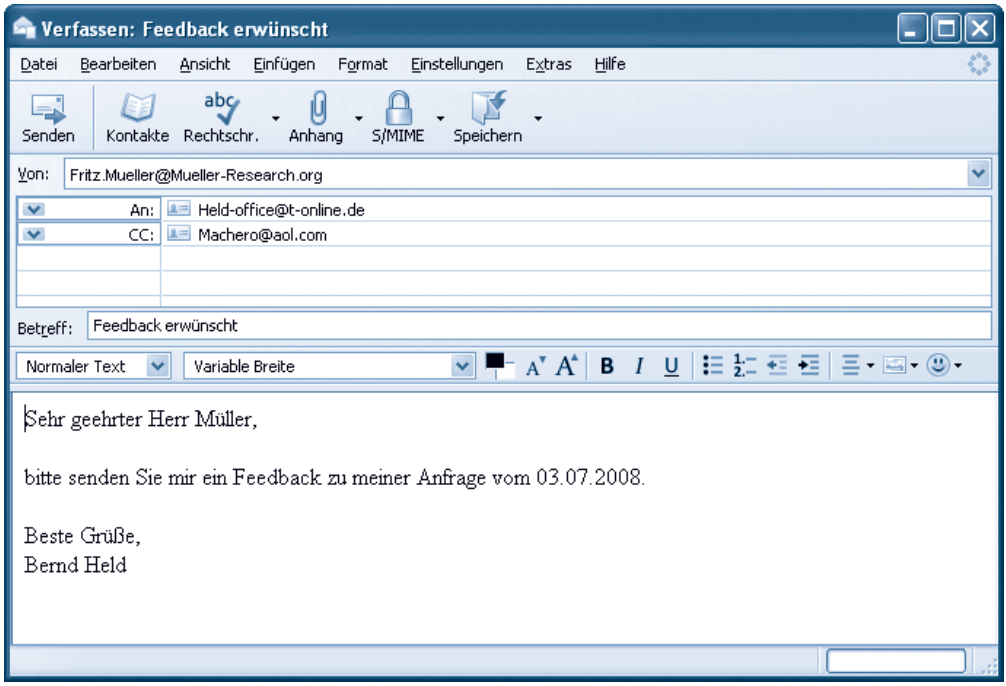


Abbildung 333: Die E-Mail ist so weit vorbereitet und muss nur noch versendet werden.

Als Alternative können Sie eine einfache Text-E-Mail auch mithilfe der API-Funktion `ShellExecute` aus *Listing 490* versenden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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, _
```

Listing 490: E-Mail über eine API-Funktion versenden

```
  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 490: E-Mail über eine API-Funktion versenden (Forts.)

Übergeben Sie der API-Funktion Ihre E-Mail-Adresse sowie den gewünschten Titel und den Text als Argument.

389 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 491* wird die Tabelle ARTIKEL im Excel-Format an mehrere E-Mail-Empfänger versendet.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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", True
End Sub
```

Listing 491: 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 das Ausgabeformat fest. Setzen Sie das Argument `NachrichtBearbeiten` auf den Wert `True`, um die E-Mail vor dem Senden noch einmal anzuzeigen.

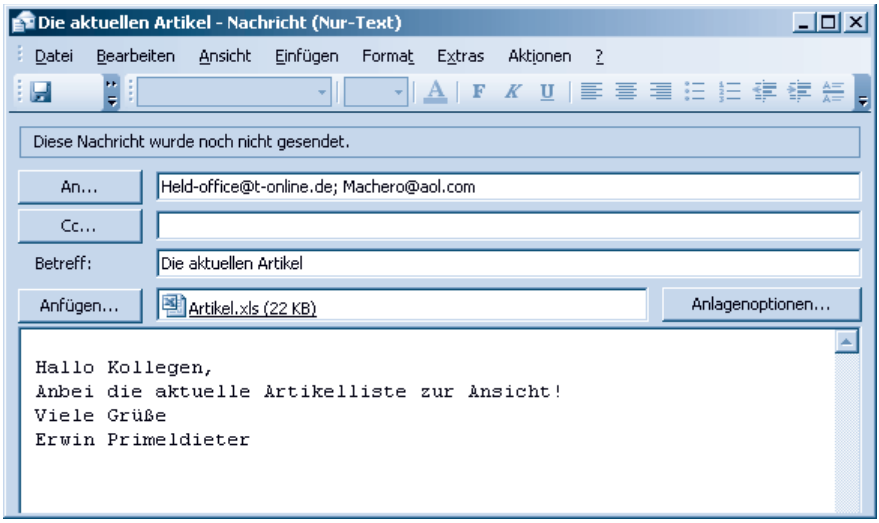


Abbildung 334: An die E-Mail wurde eine Excel-Datei angehängt.

390 E-Mail-Adressen aus Outlook herauslesen (Early-Binding)

Im nächsten Beispiel aus *Listing 492* 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, MICROSOFT OUTLOOK 11.0 OBJECT LIBRARY für Outlook 2003 bzw. MICROSOFT OUTLOOK 12.0 OBJECT LIBRARY für Outlook 2007 im Menü EXTRAS über den Befehl VERWEISE ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub EMailAdressenAuslesen()
  Dim objOutArbeitsverz As Object
  Dim objOutApp As New Outlook.Application
  Dim objKon As Object
  Dim intZ As Integer
  Dim strName As String
  Dim strVorname As String
  Dim strMail As String
```

Listing 492: Kontaktdaten ermitteln (Early-Binding)

```
  On Error GoTo Fehler
  Set objOutArbeitsverz = _
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

  For intZ = 1 To objOutArbeitsverz.Items.Count
    Set objKon = objOutArbeitsverz.Items(intZ)
    strName = objKon.LastName
    strVorname = objKon.FirstName
    strMail = objKon.Email1Address
    Debug.Print strVorname & ", " & strName & "; " & strMail
  Next intZ

  Set objKon = Nothing
  Set objOutArbeitsverz = Nothing
  Exit Sub

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

Listing 492: 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, Vornamen sowie die E-Mail-Adresse der jeweiligen Kontaktperson und schreiben diese Informationen über die Anweisung `Debug.Print` in das Direktfenster der Entwicklungsumgebung.

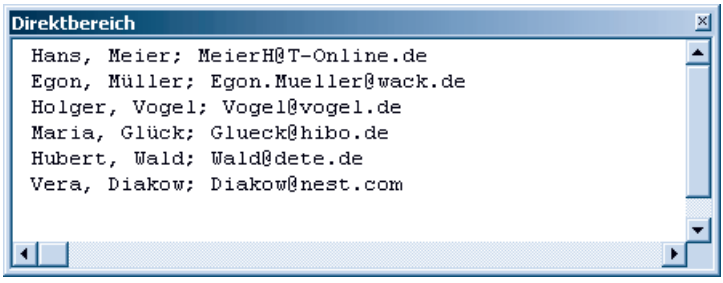


Abbildung 335: Die Kontaktdaten wurden ausgelesen.

391 Kontaktdaten aus Outlook herauslesen (Late-Binding)

Im nächsten Beispiel aus *Listing 493* wird das vorherige Beispiel aus *Listing 492* noch um ein paar Informationen erweitert. Außerdem wird dieses Mal die Aufgabe über Late-Binding realisiert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub KontaktdatenAuslesen()
  Dim objOutApp As Object
  Dim objOutArbeitsverz As Object
  Dim objOutKon As Object
  Dim intZ As Integer

  On Error GoTo Fehler
  Set objOutApp = CreateObject("Outlook.application")
  Set objOutArbeitsverz = _
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

  For intZ = 1 To objOutArbeitsverz.Items.Count
    Set objOutKon = objOutArbeitsverz.Items(intZ)
    With objOutKon
      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 objOutArbeitsverz = Nothing
  Set objOutKon = Nothing
  Set objOutApp = Nothing
  Exit Sub

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

Listing 493: Kontaktdaten ermitteln (Late-Binding)

Deklarieren Sie zunächst drei Objektvariable vom Typ `Object`. Über die Funktion `CreateObject` erstellen Sie danach ein Outlook-Objekt, das Sie der Objektvariablen `objOutApp` 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.

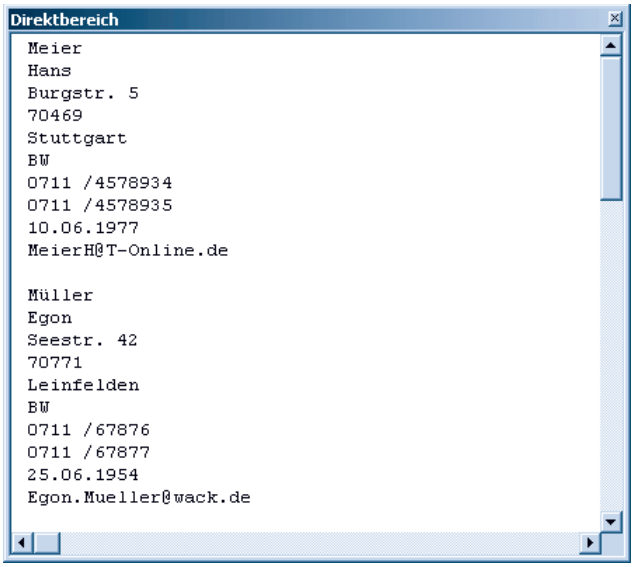


Abbildung 336: Die Kontakte wurden aus Outlook ausgelesen.

392 Kontaktdaten nach Outlook übertragen

Beim nächsten Beispiel liegen Kontaktdaten in einer Access-Tabelle vor. Diese Tabelle hat folgenden Aufbau.

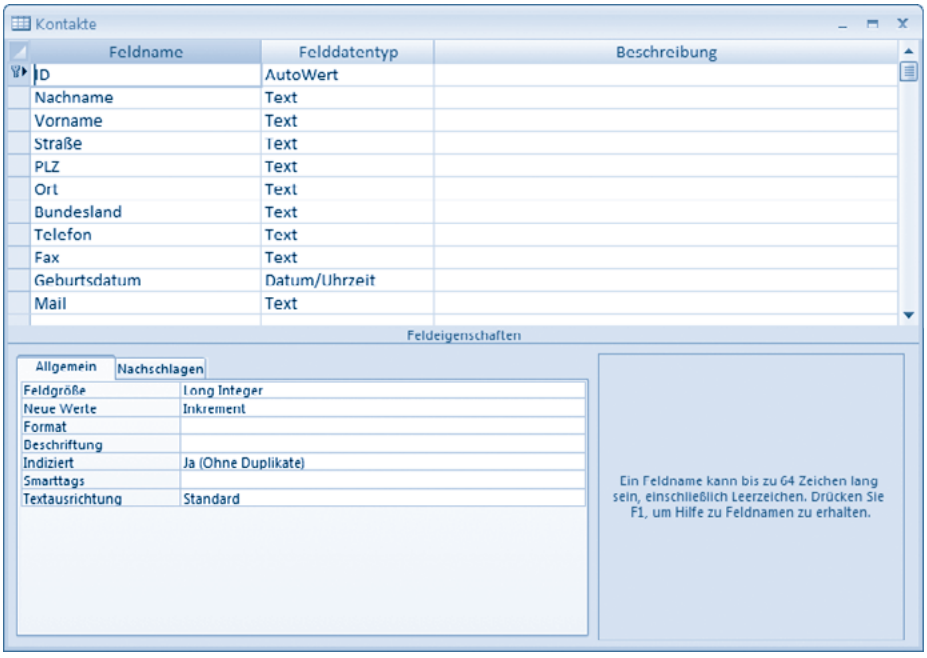


Abbildung 337: Der Aufbau der Tabelle Kontakte

Die Aufgabe besteht nun darin, die in der Tabelle KONTAKTE enthaltenen Adressen in den Kontaktordner von Outlook zu transferieren. Binden Sie vor dem Start der Prozedur aus *Listing 494* die Objektbibliothek MICROSOFT OUTLOOK ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub KontakteVonAccessInOutlookÜbertragen()
  Dim objOutApp As Outlook.Application
  Dim objOutCon As Outlook.ContactItem
  Dim rst As New ADODB.Recordset
  Dim intZ  As Integer

  Set objOutApp = CreateObject("Outlook.Application.12")
  rst.Open "Kontakte", CurrentProject.Connection
  intZ = 0

  On Error GoTo Fehler
  Do While Not rst.EOF
    Set objOutCon = objOutApp.CreateItem(olContactItem)
    objOutCon.Display
    With objOutCon
      .LastName = rst!Nachname
      .FirstName = rst!Vorname
      .BusinessAddressStreet = rst!Straße
      .BusinessAddressPostalCode = rst!PLZ
      .BusinessAddressCity = rst!Ort
      .BusinessAddressState = rst!Bundesland
      .BusinessTelephoneNumber = rst!Telefon
      .BusinessFaxNumber = rst!Fax
      .Birthday = rst!Geburtsdatum
      .Email1Address = rst!Mail
      .Save
      intZ = intZ + 1
      rst.MoveNext
    End With
  Loop

  MsgBox "Es wurden " & intZ & " Kontakte übertragen!"
  rst.Close

  Set objOutCon = Nothing
  Set objOutApp = Nothing
  Exit Sub
```

Listing 494: Kontaktdaten von Access nach Outlook übertragen

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

Listing 494: Kontaktdaten von Access nach Outlook übertragen (Forts.)

Definieren Sie im ersten Schritt zwei Objektvariable für Outlook. Die eine Variable, `objOutApp`, gibt Ihnen die Möglichkeit, die Outlook-Applikation direkt über VBA-Befehle anzusprechen. Die zweite Objektvariable, `objOutCon`, 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 `12`, dass Outlook 2007 gemeint ist. Outlook 2003 weist die Zahl `11`, Outlook 2002 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` aufzurufen.

Mithilfe der Methode `CreateItem` erzeugen Sie ein neues Outlook-Objekt. Welches Outlook-Objekt Sie genau erzeugen möchten, 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`. Damit wird der Kontakt 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 154: 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 der Prozedur ausgeben zu können, wie viele Kontakte übertragen wurden, erhöhen Sie die Zählvariable `intZ` nach jedem Schleifendurchlauf um den Wert `1`. Wurden alle Datenbankfelder der Tabelle KONTAKTE abgearbeitet, wird die Schleife verlassen. Geben Sie am Ende der Prozedur 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.

393 Kontaktdaten von Outlook importieren

Das folgende Beispiel aus *Listing 495* stellt 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 338* vor.

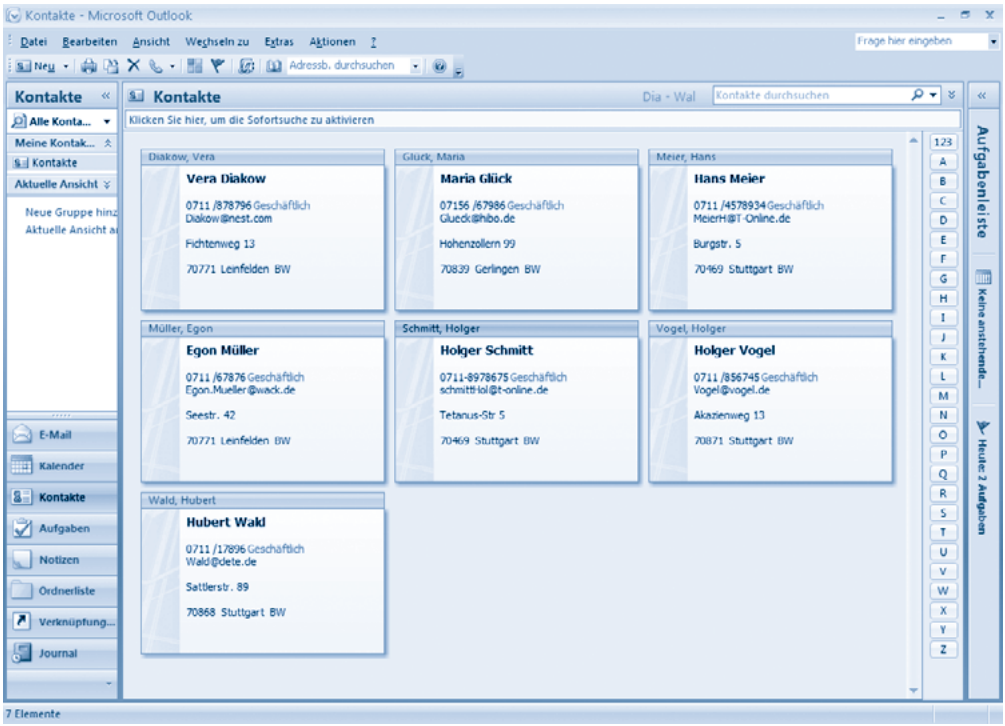


Abbildung 338: Diese Kontakte sollen in eine Access-Tabelle übertragen werden.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub KontakteAusOutlookÜbernehmen()
  Dim objOutApp As Object
  Dim objOutArbeitsverz As Object
  Dim objOutKon As Object
  Dim rst As Recordset
  Dim dbs 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
```

Listing 495: Kontaktdaten aus Outlook nach Access transferieren

```
Set dbs = CurrentDb
Set rst = dbs.OpenRecordset("Kontakte", dbOpenDynaset)
Set objOutApp = CreateObject("Outlook.Application.12")
Set objOutArbeitsverz = _
  objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

For intZ = 1 To objOutArbeitsverz.Items.Count
  Set objOutKon = objOutArbeitsverz.Items(intZ)
  strNachname = objOutKon.LastName
  strVorname = objOutKon.FirstName
  strNachname = "Nachname = '" & strNachname & "'"

  With objOutKon
    rst.FindFirst strNachname
    If rst.NoMatch Then
      rst.AddNew
      rst!Nachname = .LastName
      rst!Vorname = .FirstName
      rst!Straße = .BusinessAddressStreet
      rst!PLZ = .BusinessAddressPostalCode
      rst!Ort = .BusinessAddressCity
      rst!Bundesland = .BusinessAddressState
      rst!Telefon = .BusinessTelephoneNumber
      rst!Fax = .BusinessFaxNumber
      rst!Geburtsdatum = .Birthday
      rst!Mail = .Email1Address
      intNeu = intNeu + 1
    Else
      If rst!Vorname = strVorname Then
        rst.Edit
        rst!Straße = .BusinessAddressStreet
        rst!PLZ = .BusinessAddressPostalCode
        rst!Ort = .BusinessAddressCity
        rst!Bundesland = .BusinessAddressState
        rst!Telefon = .BusinessTelephoneNumber
        rst!Fax = .BusinessFaxNumber
        rst!Geburtsdatum = .Birthday
        rst!Mail = .Email1Address
        intUpd = intUpd + 1
      Else
        rst.AddNew
        rst!Nachname = .LastName
        rst!Vorname = .FirstName
        rst!Straße = .BusinessAddressStreet
        rst!PLZ = .BusinessAddressPostalCode
        rst!Ort = .BusinessAddressCity
        rst!Bundesland = .BusinessAddressState
```

Listing 495: Kontaktdaten aus Outlook nach Access transferieren (Forts.)

```
            rst!Telefon = .BusinessTelephoneNumber
            rst!Fax = .BusinessFaxNumber
            rst!Geburtsdatum = .Birthday
            rst!eMail = .Email1Address
            intNeu = intNeu + 1
          End If
        End If
      End With
      rst.Update
    Next intZ
    MsgBox "Datentransfer erfolgreich beendet! " & vbLf & _
      "Es wurden " & intNeu & " Sätze angelegt, " & vbLf & _
      "Es wurden " & intUpd & " Sätze upgedatet!"
    rst.Close

    Set objOutKon = Nothing
    Set objOutArbeitsverz = Nothing
    Set objOutApp = Nothing
    Set dbs = Nothing
    Exit Sub

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

Listing 495: Kontaktdaten aus Outlook nach Access transferieren (Forts.)

Die Prozedur 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, der in der Variablen `strNachname` zwischengespeichert wird. Mit der `FindFirst`-Methode wird der erste Datensatz in der Tabelle gesucht, der 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, sondern lediglich der bereits bestehende Satz aktualisiert werden. Als aktualisierbare Informationen sind hier die Felder STRAßE, PLZ, ORT BUNDESLAND, TELEFON und FAX vorgesehen. Die Modifikation wird über die Methode `Edit` eingeleitet, die 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 Vorname nicht mit dem in Outlook gespeicherten VORNAMEN überein, muss der Satz in der Kontakttabelle neu angelegt werden. Dies geschieht über die `AddNew`-Methode. Gleich danach werden alle Kontaktfelder aus Outlook in den Kopierpuffer kopiert.

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 für den Neuanlage-Fall werden zwei verschiedene Zähler verwendet, die am Ende des Datenaustausches 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.

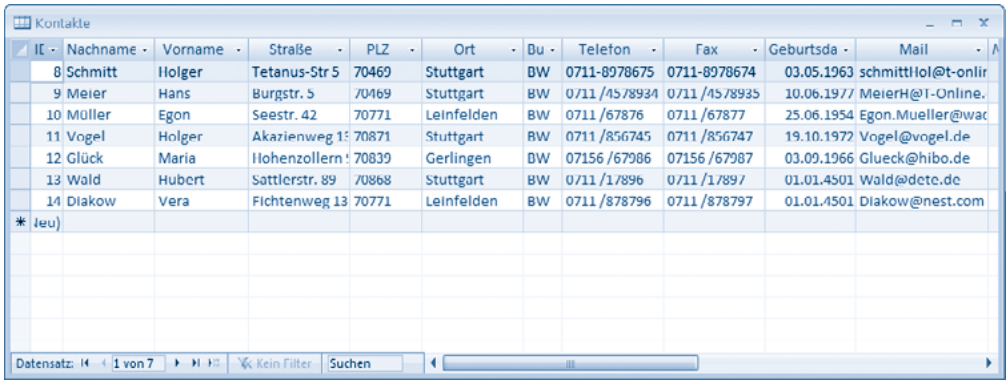


Abbildung 339: Die Kontakte wurden erfolgreich übertragen.

394 Termine aus dem Outlook-Kalender abfragen

Im nächsten Beispiel aus *Listing 496* werden Termine aus dem Outlook-Kalender ausgelesen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub TermineAuslesen()
  Dim objOutApp As Object
  Dim objOutKal As Object
  Dim objOutTermin As Object
  Dim intZ  As Integer

  Set objOutApp = CreateObject("Outlook.Application.11")
  Set objOutKal = _
```

Listing 496: Termine aus dem Outlook-Kalender abfragen

```
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderCalendar)

  For intZ = 1 To objOutKal.Items.Count
    Set objOutTermin = objOutKal.Items(intZ)
    With objOutTermin
      Debug.Print .Subject
      Debug.Print .Body
      Debug.Print .Categories
      Debug.Print .Start & vbLf
    End With
  Next intZ

  MsgBox "Es wurden " & intZ & " Termine gefunden!"
  Set objOutApp = Nothing
  Set objOutKal = Nothing
  Set objOutTermin = Nothing
  Exit Sub

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

Listing 496: 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 155*.

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.

Tabelle 155: Einige Angaben zum Terminkalender

Eigenschaft	Beschreibung
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.
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 155: Einige Angaben zum Terminkalender (Forts.)

395 Aufgaben abfragen

Im folgenden Beispiel aus *Listing 497* 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 340* ausgegangen.

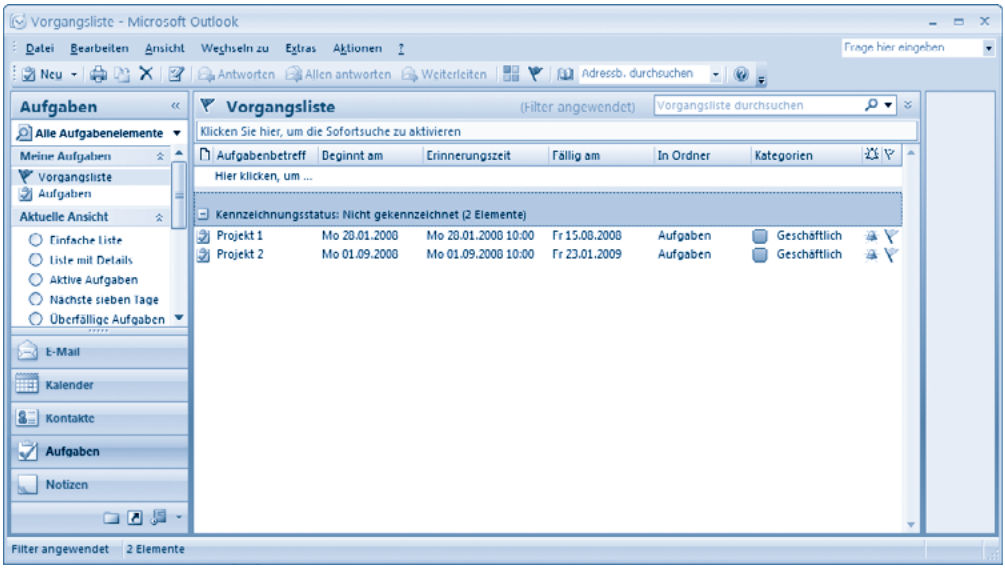


Abbildung 340: Diese Aufgabenliste wird ausgelesen.

```
'=====================================================
' Auf CD      Buchdaten\Beispiele
' Dateiname   Umfeld.mdb
' Modul       mdlOut
'=====================================================
```

Listing 497: Aufgaben aus Outlook auslesen

```
Sub AufgabenAuslesen()
  Dim objOutApp As Object
  Dim objOutTask As Object
  Dim objOutAufgabe As Object
  Dim intZ  As Integer

  Set objOutApp = CreateObject("Outlook.Application.12")
  Set objOutTask = _
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderTasks)

  For intZ = 1 To objOutTask.Items.Count
    Set objOutAufgabe = objOutTask.Items(intZ)
    With objOutAufgabe
      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 - 1 & " Aufgaben gefunden!"
  Set objOutAufgabe = Nothing
  Set objOutTask = Nothing
  Set objOutApp = Nothing
  Exit Sub

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

Listing 497: Aufgaben aus Outlook auslesen (Forts.)

Möchten Sie die Aufgabenliste von Outlook ansprechen, dann müssen Sie der Methode `GetDefaultFolder` die Konstante `olFolderTasks` übergeben. 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.

Tabelle 156: Einige Eigenschaften zur Aufgabenliste von Outlook

Eigenschaft	Beschreibung
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.
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 156: Einige Eigenschaften zur Aufgabenliste von Outlook

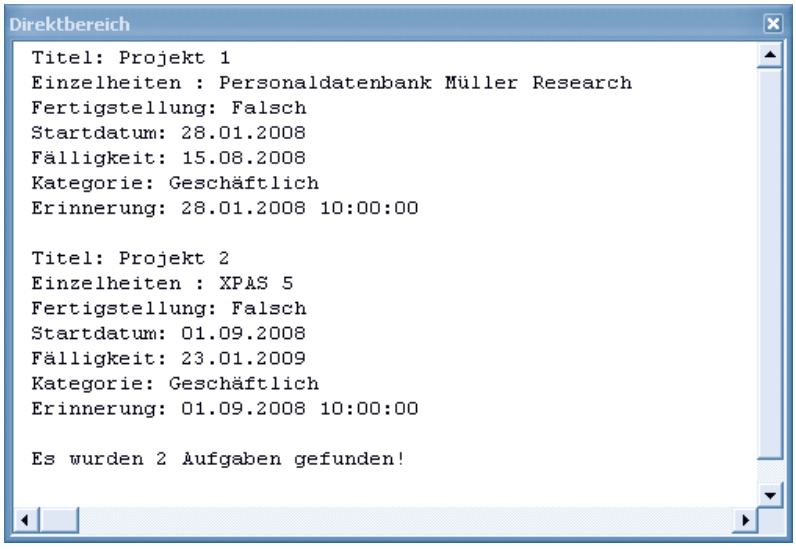


Abbildung 341: Die Aufgaben wurden direkt aus Outlook extrahiert.

396 E-Mails in Access-Tabelle importieren

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 per VBA-Prozedur in diese Tabelle einfügen. Im nächsten Beispiel sollen die folgenden Felder dokumentiert werden:

- Der Titel der E-Mails
- Der Name des Absenders der E-Mail
- Der Inhalt der E-Mail
- Das Datum/Uhrzeit des E-Mail-Eingangs
- Die Anzahl der Dateianhänge der E-Mail
- Den Lesestatus der E-Mail

Legen Sie zunächst eine Tabelle nach dem Vorbild aus *Abbildung 342* an.

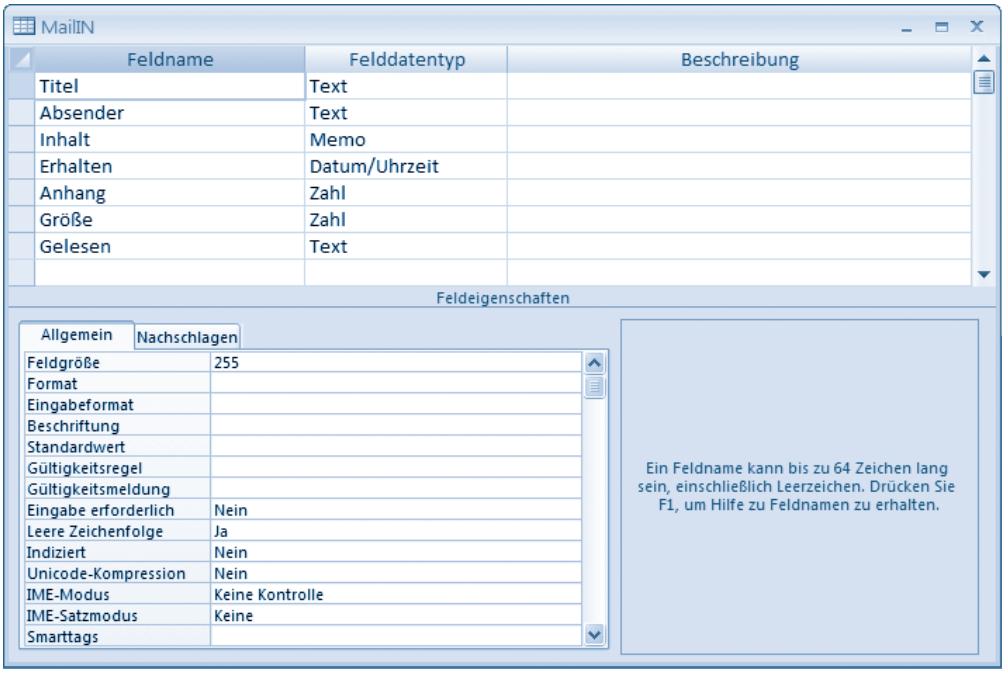


Abbildung 342: Der Aufbau der Tabelle MailIN

Erfassen Sie jetzt die Prozedur aus *Listing 498*, die die Tabelle MAILIN öffnet, im Hintergrund auf Ihren Posteingangsordner zugreift und die einzelnen E-Mail-Informationen in die Tabelle einträgt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub EMailsSichern()
  Dim objOutFold As Outlook.MAPIFolder
  Dim intGes As Integer
  Dim intZ As Integer
  Dim conn As New ADODB.Connection
  Dim rst As ADODB.Recordset

  On Error GoTo Fehler
  Set conn = CurrentProject.Connection
  Set rst = New ADODB.Recordset
  rst.Open "MailIN", conn, adOpenKeyset, adLockOptimistic

  Set objOutFold = GetObject("", "Outlook.Application").GetNamespace _
    ("MAPI").GetDefaultFolder(olFolderInbox)

  intGes = objOutFold.Items.Count
  intZ = 0

  For intZ = 1 To intGes
    With objOutFold.Items(intZ)
      rst.AddNew
      rst!Titel = .Subject
      rst!Absender = .SenderName
      rst!Inhalt = .Body
      rst!Erhalten = Format(.ReceivedTime, "dd.mm.yyyy hh:mm")
      rst!Anhang = .Attachments.Count
      rst!Größe = .Size
      If Not .UnRead = -1 Then
        rst!Gelesen = "JA"
      Else
        rst!Gelesen = "Nein"
      End If
      rst.Update
    End With
  Next intZ

  rst.Close

  With rst
    .CursorLocation = adUseClient
```

Listing 498: Alle Mails aus dem Posteingang werden in eine Access-Tabelle geschrieben.

```
    .Open "MailIN", conn, adOpenKeyset, adLockOptimistic
    .Sort = "Erhalten ASC"
  End With

  Set objOutFold = Nothing
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 498: Alle Mails aus dem Posteingang werden in eine Access-Tabelle geschrieben. (Forts.)

Über das Objekt `CurrentProject` erhalten Sie eine Referenz auf die aktuell geöffnete Datenbank und öffnen über ein Objekt vom Typ `ADODB.Recordset` die gerade angelegte Tabelle durch Aufruf der Methode `Open`.

Mithilfe der Methoden `GetObject` und `GetDefaultFolder`, der Sie die Konstante `olFolderInbox` übergeben, gewinnen Sie direkten Zugriff auf Ihren Posteingangsordner von Outlook.

Über die Eigenschaft `Count` ermitteln 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 rufen Sie die Methode `AddNew` auf, 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 dem 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`, die 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 Wertes schreiben Sie entweder den Text JA bzw. den Text NEIN in das Datenbankfeld GELESEN.

Rufen Sie die Methode `Update` auf, 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 direkt im Anschluss gleich wieder und setzen die Eigenschaft `Sort` ein, um die dokumentierten E-Mails nach dem Posteingangsdatum zu sortieren.

MailIN

Titel	Absender	Inhalt	Erhalten	Anhang	Größe	Gelesen
test	MacHero@aol.com		12.10.2008 22:28:00	0	3565	JA
Neue Version	Rene	Hallo, Herr Held	14.10.2008 18:30:00	1	106524	JA
SmartTools Excel Weekly vom 15.10.2003	excel-weekly-html-reque	http://www.smarttools	15.10.2008 01:25:00	0	79775	JA
gleichzeitig "Enter" und "Doppelklick" ???	Weiss2, Armin		15.10.2008 11:31:00	1	55912	JA
MVP Bücherbestellungen	Angela Knoeckel-Reinoel	Hallo liebe MVPs,	15.10.2008 15:11:00	0	8454	JA
AW: gleichzeitig "Enter" und "Doppelklick" ?	Weiss2, Armin	Hallo Bernd,	15.10.2008 16:47:00	0	10027	JA
Newsgroup Emailadresse wird abgefälscht	Evelyn Ruf	Liebe MVPs,	16.10.2008 12:22:00	0	9220	JA
für MVPs, die auf der PDC in Los Angeles sind	Evelyn Ruf	Hallo MVPs,	16.10.2008 12:35:00	0	13817	JA
Excel-Problem	info@ttcomm.de	Schönen guten Tag, Her	17.10.2008 12:16:00	1	24136	JA
Niedzielne spotkanie	Joanna Frey	Hi Babki,	18.10.2008 10:47:00	0	7511	JA
Schreiben Sie zu uns eine E-Mail	info@rwe.com	Sehr geehrte Dame, sel	18.10.2008 14:07:00	0	13704	JA
Formel SUMME	"Carsta & René"	Hallo Bernd,	18.10.2008 21:57:00	0	4645	JA
(Kein Thema)	JensSa@aol.com	Hallo Bernd,	19.10.2008 15:41:00	0	6343	JA
Re: Anfrage	Eric Leupold	Guten Tag,	20.10.2008 09:36:00	0	16016	JA
Willkommen bei Microsoft Office Outlook 20	Outlook 2003 Team	Herzlich willkommen b	16.12.2008 10:53:00	11	52903	JA

Datensatz: 1 von 15 | Kein Filter | Suchen

Abbildung 343: Alle E-Mails werden in Access gesichert.

397 Sammel-E-Mails versenden

Bei der folgenden Lösung aus *Listing 499* werden gleich mehrere E-Mails an unterschiedliche Adressaten gesendet. Dabei werden die einzelnen E-Mail-Adressen aus der Access-Tabelle KONTAKTE entnommen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub MehrereEMailsVersenden()
  Dim objOutVerz As Object
  Dim objOutMail As Object
  Dim objOutApp As New Outlook.Application
  Dim conn As Database
  Dim rst As Recordset
  Dim strText As String
  Const Titel = "ACHTUNG"
  Const MailText = "Auf dem Server befinden sich neue Updates!"

  On Error GoTo Fehler
  Set objOutVerz = _
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderContacts)

  Set conn = CurrentDb()
```

Listing 499: Eine Sammel-E-Mail versenden

```
  strText = "SELECT Mail from Kontakte"
  Set rst = conn.OpenRecordset(strText)

  Do Until rst.EOF
  If Not IsNull(rst!Mail) Then
    Set objOutMail = CreateItem(olMailItem)
    With objOutMail
      .Subject = Titel
      .Body = MailText
      .Attachments.Add "C:\Eigene Dateien\Info.txt"
      .To = rst!Mail
      .Save
    End With
  End If
  rst.MoveNext
  Loop

  Set objOutVerz = Nothing
  Set objOutMail = Nothing
  Set rst = Nothing
  Set conn = Nothing
  Exit Sub

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

Listing 499: Eine Sammel-E-Mail versenden (Forts.)

Den Titel sowie den eigentlichen Text der E-Mail können Sie zu Beginn der Prozedur als Konstanten 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, um festzustellen, ob eine E-Mail-Adresse angegeben ist. Damit erreichen Sie, dass Datensätze, die keine E-Mail-Adresse beinhalten, nicht berücksichtigt werden. Danach erstellen Sie über die Methode `CreateItem` ein neue, noch leere E-Mail-Nachricht, indem Sie der Methode die Konstante `olMailItem` übergeben. 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 `Attachments` angewendet wird, geben Sie den Pfad zu einer Datei an, die als Dateianhang an die Mail angehängt wird. Über den Einsatz der Methode `Save` speichern Sie zunächst alle E-Mails im Outlook-Ordner ENTWÜRFE. Von dort können sie dann nach einer kurzen Überprüfung, zu einem späteren Zeitpunkt, direkt aus Outlook versendet werden.

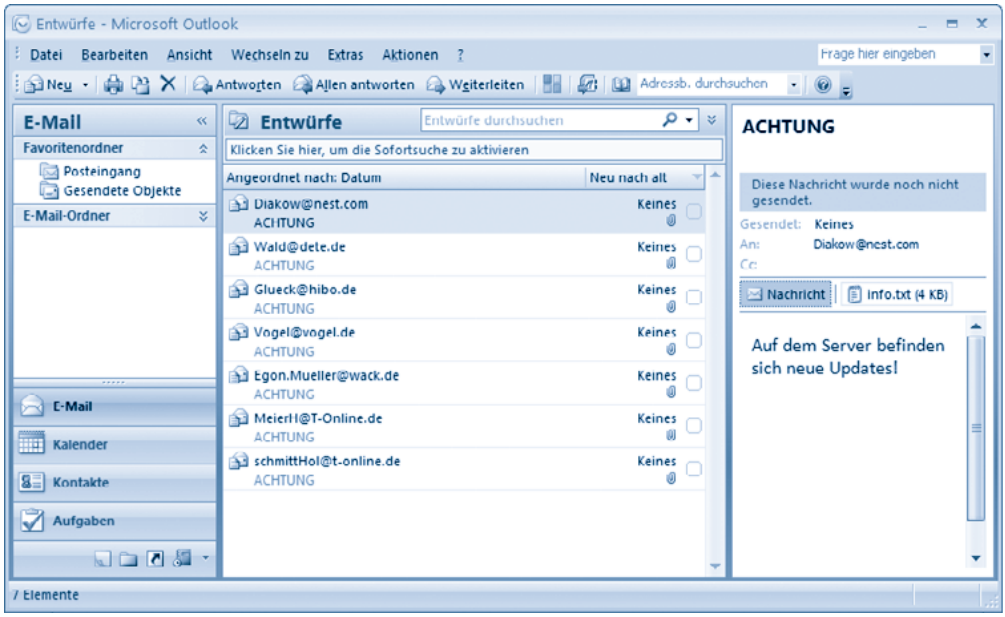


Abbildung 344: Mehrere Mails wurden in den Ordner Entwürfe geschrieben.

398 E-Mail-Anhänge speichern

Bei der folgenden Lösung aus *Listing 500* handelt es sich um einen Automatismus, der auf den Posteingangsordner von Outlook zugreift und alle Mails abarbeitet. Dabei werden alle E-Mail-Anhänge in einem bestimmten Ordner gespeichert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlOut
'=======================================================

Sub AnhängeSpeichern()
  Dim objOutApp As New Outlook.Application
  Dim objOutVerz As Object
  Dim objOutMail As Object
  Dim intZ As Integer
  Dim intAnhZ As Integer

  On Error GoTo Fehler
  Set objOutVerz = _
    objOutApp.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox)
```

Listing 500: E-Mail-Anhänge werden separat gespeichert.

```
  For intZ = 1 To objOutVerz.Items.Count
    Set objOutMail = objOutVerz.Items(intZ)

    If objOutMail.Attachments.Count > 0 Then
      For intAnhZ = 1 To objOutMail.Attachments.Count
        objOutMail.Attachments.Item(intAnhZ).SaveAsFile _
          "C:\Test\" & objOutMail.Attachments.Item(intAnhZ)
      Next intAnhZ
    End If
    Set objOutMail = Nothing
  Next intZ

  Set objOutVerz = Nothing
  Set objOutMail = Nothing
  MsgBox "Verarbeitung beendet!"
  Exit Sub

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

Listing 500: E-Mail-Anhänge werden separat gespeichert. (Forts.)

In der Prozedur aus *Listing 500* werden alle E-Mails des aktiven E-Mail-Ordners gezählt, das heißt, man muss vor dem Start der Prozedur den gewünschten Ordner in Outlook aktivieren. Das Zählen der E-Mails wird über die Funktion `Count` durchgeführt. Danach werden alle E-Mails des vorher eingestellten Ordners in Outlook über eine Schleife abgearbeitet. Im Fall, dass eine E-Mail einen Anhang hat, meldet die Anweisung `OutMail.Attachments.Count` einen Wert größer als null. In diesem Fall wird der Anhang über die Methode `SaveAsFile` im angegebenen Verzeichnis gespeichert. Am Ende der Prozedur werden die Objektverweise über das Schlüsselwort `Nothing` wieder aufgehoben, um den für die Objekte belegten Speicher wieder freizugeben.

Access & Internet Explorer

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 aus dem Kombinationsfeld den Eintrag SHDocVw aus.

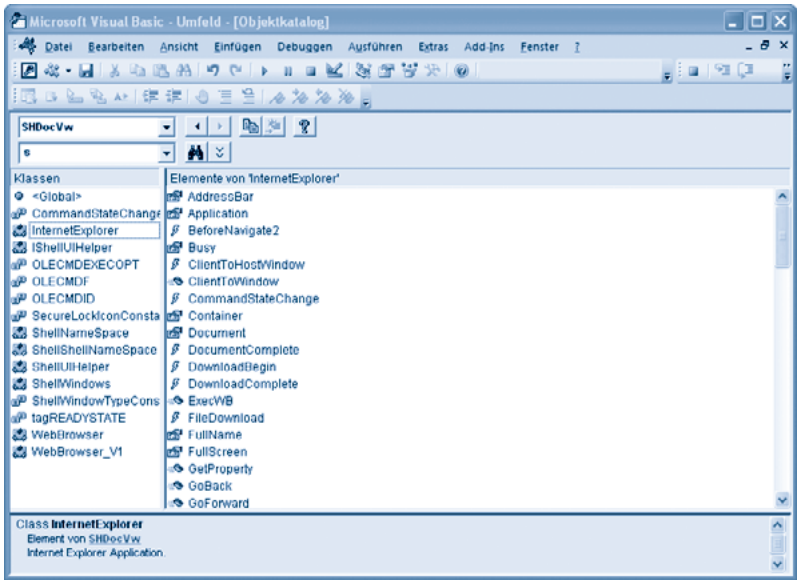


Abbildung 345: Alle verfügbaren Befehle des Internet Explorers

399 Eine Diashow programmieren

Im folgenden Beispiel aus *Listing 501* wird ein Verzeichnis nach Grafikdateien durchsucht. Die gefundenen Grafikdateien werden im Internet Explorer im Abstand von jeweils zwei Sekunden ähnlich einer Diashow angezeigt.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlIE
'=======================================================

Declare Sub Sleep Lib "Kernel32.dll" (ByVal SchlafZeit As Long)

Sub InternetExplorer()
  Dim objIE As New InternetExplorer
  Dim intZ As Integer
  Dim varDat As Variant
  Dim strDir As String
  Dim strDatei As String

  strDir = "C:\Eigene Dateien\"
  objIE.Visible = True

  strDatei = Dir(strDir & "*.jpg")
```

Listing 501: Eine Diashow im Internet Explorer durchführen

```
  Do Until strDatei = ""
    objIE.Navigate strDir & strDatei
    Sleep 2000
    strDatei = Dir
  Loop
  objIE.Quit
  Set objIE = Nothing
End Sub
```

Listing 501: Eine Diashow im Internet Explorer durchführen (Forts.)

Erstellen Sie zunächst über die Deklaration `Dim IE as New InternetExplorer` ein neues Internet Explorer-Objekt . Dadurch wird eine neue Internet Explorer-Instanz gestartet. Über die Objektvariable vom Typ `InternetExplorer` haben Sie dann Zugriff auf alle Methoden und Eigenschaften, die das `InternetExplorer`-Objekt standardmäßig anbietet. Weisen Sie der `InternetExplorer`-Eigenschaft `Visible` den Wert `True` zu, um den Internet Explorer anzuzeigen. Im nächsten Schritt führen Sie über die Funktion `Dir` eine Suche in einer Schleife durch und ermitteln dabei alle Grafikdateien mit der Endung .JPG innerhalb des Verzeichnisses C:\EIGENE DATEIEN.

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, wird eine Pause von 2.000 Millisekunden eingelegt. Diese Pause wird über die API-Funktion `Sleep` realisiert.

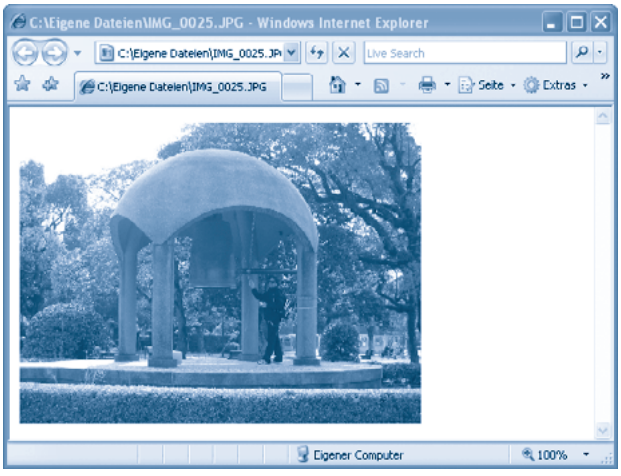


Abbildung 346: Die Diashow über den Internet Explorer

400 URLs auslesen

Im folgenden Beispiel wird der Text einer Internetseite ausgelesen. Dazu erfassen Sie die Funktion aus *Listing 502*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlIE
'=====================================================

Function URLText(sURL As String) As String
  Dim objIE As Object

  Set objIE = CreateObject("InternetExplorer.Application")
  objIE.Navigate sURL
  URLText = objIE.Document.Body.InnerText

  objIE.Quit
  Set objIE = Nothing
End Function
```

Listing 502: Internetseitentext auslesen

Durch die Anweisung `CreateObject` wird eine neues Internet Explorer-Objekt erstellt. Über die Methode `Navigate` laden Sie die angegebene Seite. Mithilfe der Eigenschaft `InnerText` des Objekts `Body` können Sie den Text der Internetseite abfragen. Die Internet Explorer-Anwendung wird am Ende über die Methode `Quit` beendet. Danach wird der Objektverweis durch die Anweisung `Set IE = Nothing` aufgehoben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlIE
'=====================================================

Sub OeffnenIE()
  Dim strSeite As String

  strSeite = URLText(Application.CurrentProject.Path & "\Artikel.htm")
  MsgBox strSeite
End Sub
```

Listing 503: URL-Text einer HTML-Seite auslesen

Access & Word

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. 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` ein neues Word-Objekt. Eine praktische Anwendung liefert das Beispiel aus *Listing 506*.

401 Ist Word bereits gestartet?

Als erstes Beispiel für das Zusammenspiel von Access und Word schreiben Sie die Funktion aus *Listing 504*, die überprüft, ob Word bereits gestartet ist. Dabei muss als Voraussetzung die Word-Bibliothek unter EXTRAS/VERWEISE eingebunden sein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================

Function WordOffen() As Boolean
  Dim objWordApp As Word.Application

  On Error Resume Next
  Set objWordApp = GetObject(, "Word.Application")

  If Err.Number = 0 Then
    WordOffen = True
  Else
    WordOffen = False
  End If
  Set objWordApp = Nothing
End Function
```

Listing 504: 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 die aufrufende Prozedur aus *Listing 505* zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================

Sub StartWord()
  Dim objWordApp As Object

  If WordOffen = False Then
    Set objWordApp = CreateObject("Word.Application")
    objWordApp.Documents.Add
```

Listing 505: Word wird gestartet.

```
    objWordApp.Visible = True
  End If
End Sub
```

Listing 505: Word wird gestartet. (Forts.)

In der Prozedur aus *Listing 505* wurde Late-Binding eingesetzt und die Objektvariable `objWordApp` 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. Rufen Sie im Anschluss daran die Methode `Add` des Objekts `Documents` auf, um ein neues Dokument zu erstellen. Wenden Sie die Eigenschaft `Visible` an, um die Applikation anzuzeigen.

402 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. Die etwas längere Prozedur für diese Aufgabe sehen Sie in *Listing 506*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=====================================================

Sub AccessTabelleNachWord()
  Dim objWordApp As Object
  Dim objWordDoc As Object
  Dim intPos As Object
  Dim objWordTab As Object
  Dim rst As New ADODB.Recordset
  Dim intZ As Integer
  On Error GoTo Fehler

  rst.Open "Artikel", CurrentProject.Connection

  On Error Resume Next
  Set objWordApp = GetObject(, "Word.Application")
  If Err.Number = 429 Then
    Set objWordApp = CreateObject("Word.Application")
    Err.Number = 0
  End If

  objWordApp.Visible = True
```

Listing 506: Eine Access-Tabelle in ein Word-Dokument übertragen – Variante 1

```
  Set objWordDoc = objWordApp.Documents.Add

  With objWordApp.Selection
    .TypeText Text:="Artikelliste aus : " & _
      CurrentProject.Name
    .TypeParagraph
    .TypeText Text:="vom  " & Format(Now(), "dd-mmm-yyyy")
    .TypeParagraph
  End With

  Set intPos = objWordDoc.Range(Start:=0, End:=0)
  Set objWordTab = objWordDoc.Tables.Add(Range:=intPos, NumRows:=80, _
    NumColumns:=6)
  With objWordTab
    .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 rst.EOF
    With objWordTab
      .Cell(intZ, 1).Range.Text = rst!Artikelname
      .Cell(intZ, 2).Range.Text = rst!Liefereinheit
      .Cell(intZ, 3).Range.Text = rst!Einzelpreis
      .Cell(intZ, 4).Range.Text = rst!Mindestbestand
      .Cell(intZ, 5).Range.Text = rst!Lagerbestand
      .Cell(intZ, 6).Range.Text = rst!BestellteEinheiten
      intZ = intZ + 1
      rst.MoveNext
    End With
  Loop

  objWordDoc.SaveAs "C:\Eigene Dateien\Artikel.doc"
  objWordDoc.Close

  Set objWordApp = Nothing
  Set objWordDoc = Nothing
  Set objWordTab = Nothing
  Exit Sub

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

Listing 506: 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 Word-Applikation 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 Datensätze in der Tabelle ARTIKEL programmiertechnisch abarbeiten 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 weiterverarbeiten können, speichern Sie dieses in der Objektvariablen `objWordDoc`.

Sie haben jetzt eine Mischung aus Access- und Word-VBA-Befehlen. Unterscheiden können Sie diese, indem Sie sich immer das anführende Objekt ansehen. `objWordObj`, `objWordDoc` und `objWordTab` beinhalten Word-VBA-Befehle, das Objekt `rst` beinhaltet 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` gibt in Verbindung mit der Eigenschaft `Name` 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, an der die neue Tabelle eingefügt werden soll. Diese Information speichern Sie in der Objektvariablen `IntPos`. Mithilfe der Methode `Add` des Objekts `Tables` 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` legt die Anzahl der Spalten fest, 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 `objWordTab` und speichern in ihr die erste Tabelle Ihres Dokuments.

In einer anschließenden Schleife wird die Access-Tabelle ARTIKEL durchlaufen und die einzelnen Feldinhalte in die dafür vorgesehen 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, die Sie über die Eigenschaft `Text` füllen können. Als Text setzen Sie dabei natürlich die einzelnen Datenfelder Ihrer Access-Tabelle ein. Denken Sie daran, 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 ARTIKEL-Tabelle 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.

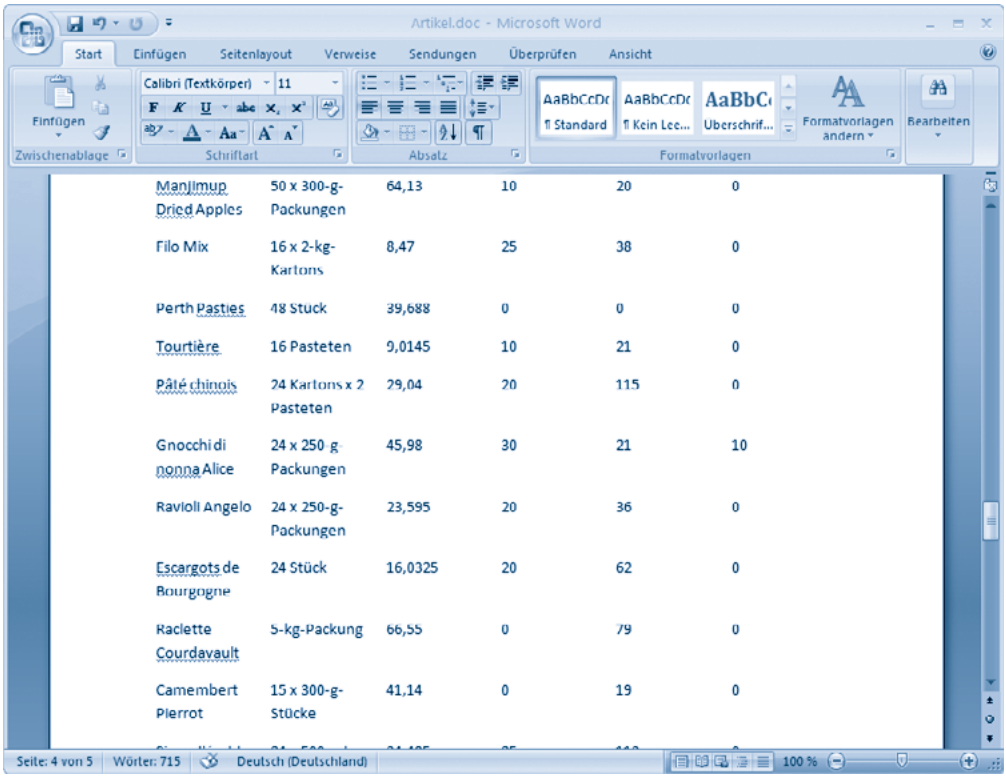


Abbildung 347: Die Tabelle wurde nach Word übertragen.

403 Access-Tabellen transferieren

Eine weitere sehr komfortable Art und Weise, eine Tabelle in ein Word-Dokument zu transferieren, können Sie verwenden, wenn Sie die Methode `OutputTo` einsetzen. Bei dieser Methode wird Ihnen der komplette Datentransfer weitestgehend abgenommen. Dabei wird die komplette Tabelle übertragen.

Im folgenden Beispiel aus *Listing 507* wird die Tabelle ARTIKEL als Word-Dokument ausgegeben.

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

Listing 507: 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 inkl. aller Funktionen und Prozeduren
- `acOutputQuery`: Export der Ergebnisse eine 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. In der Prozedur aus *Listing 507* ist dies der Name des Objekts `acOutputTable` ARTIKEL.

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 der Datei an, in die Sie die Daten transferieren möchten. Dabei muss die Datei noch nicht vorhanden sein. Access legt diese bei Bedarf selber 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.

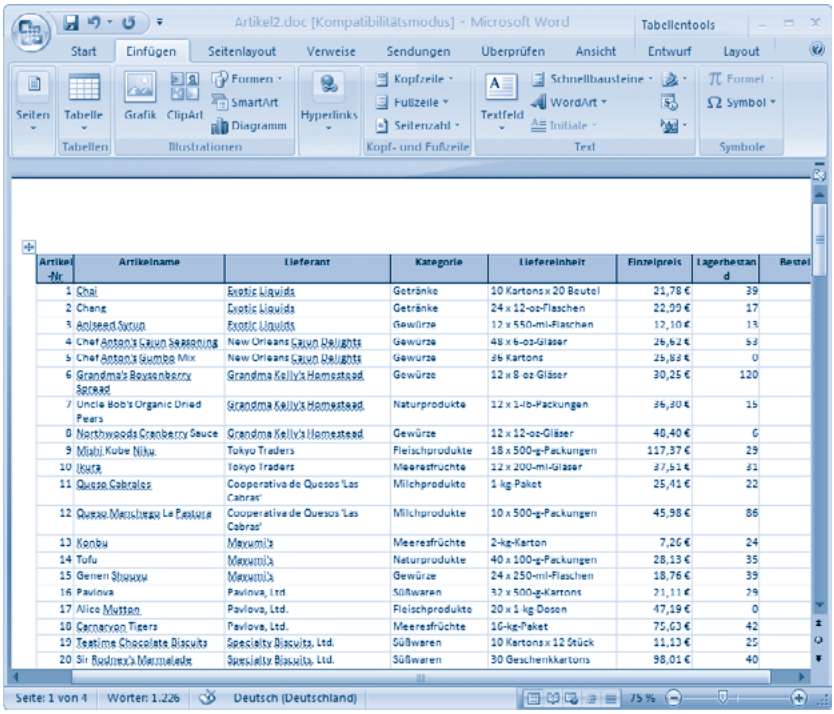


Abbildung 348: Die Ausgabe einer Tabelle über OutputTo

404 Word-Prozeduren starten

Über den Einsatz der Methode `Run` können Sie Prozeduren starten. Wenn sich beispielsweise die Prozedur `Konvertierung` in einem Word-Dokument befindet, und Sie diese von Access aus starten möchten, dann wenden Sie die Prozedur aus *Listing 508* an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlWord
'=======================================================

Sub WordProzedurStarten()
  Dim objWordApp As Object

  Set objWordApp = CreateObject("Word.Application")

  objWordApp.Documents.Open "C:\Eigene Dateien\Dok1.Doc"
  objWordApp.Run "Modul1.DatumAnzeigen"
  objWordApp.Close
  objWordApp.Quit

  Set objWordApp = Nothing
End Sub
```

Listing 508: Eine Word-Prozedur von Access aus starten

Setzen Sie die Methode `Open` ein, um das Word-Dokument zu öffnen, das die zu startende Prozedur enthält. Führen Sie danach die Methode `Run` aus, der Sie den Namen der Prozedur bekannt geben. Es kann dabei auch nicht schaden, genau anzugeben, in welchem Modul sich die Prozedur befindet. Schließen Sie am Ende der Prozedur das Dokument über die Methode `Close` und beenden Sie die Word-Anwendung mithilfe der Methode `Quit`. Heben Sie danach den Objektverweis `objWordApp` wieder auf, um den reservierten Speicher wieder freizugeben.

405 Datenbankzugriff mit Word durchführen

Der Datentransfer von Access nach Word läuft 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.

Beim folgenden Beispiel wird von der Tabelle KONTAKTE ausgegangen, die Sie sich in *Abbildung 349* vorab mal ansehen können.

Abbildung 349: Die Ausgangstabelle mit den Kontaktdaten

Im nächsten Schritt legen Sie ein Word-Dokument mit dem Namen MEMO.DOC an und fügen über den Menübefehl EINFÜGEN/TEXTMARKE einige Textmarken ein.

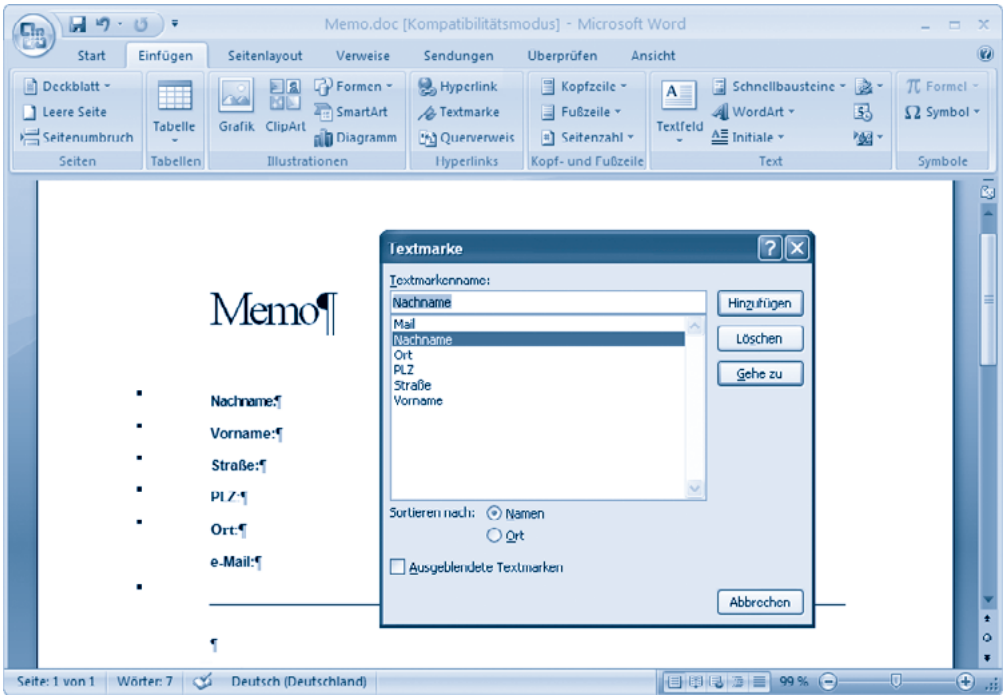


Abbildung 350: Die Vorbereitung im Dokument

Verwenden Sie bei den Textmarken die in *Abbildung 350* vorgeschlagenen Textmarken-Namen, damit der anschließende Code ohne weitere Anpassung läuft.

Da die Kontaktdaten-Abfrage-Prozedur von Word aus gestartet werden soll, ist ein Dialog vorgesehen, in 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 509* genau hinter die Userform.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Memo.Doc
' Klasse     Userform1
'=======================================================
Private Sub UserForm_Initialize()
  Dim con As ADODB.Connection
  Dim rst As ADODB.Recordset

  Set con = New ADODB.Connection
  With con
    .Provider = "Microsoft.Jet.OLEDB.4.0"
    .Open ThisDocument.Path & "\Umfeld.mdb"
  End With

  Set rst = New ADODB.Recordset
  With rst
    .Open Source:="Kontakte", ActiveConnection:=con, _
      CursorType:=adOpenKeyset, LockType:=adLockOptimistic
    Do While Not .EOF
      UserForm1.ComboBox1.AddItem .Fields("Nachname").Value & ", " & _
        .Fields("Vorname").Value
      .MoveNext
    Loop
  .Close
  End With
  con.Close
  UserForm1.ComboBox1.ListIndex = 0

  Set rst = Nothing
End Sub
```

Listing 509: Das Kombinationsfeld beim Aufruf der Userform füllen

Damit die Prozedur aus *Listing 509* korrekt abläuft, binden Sie in der Entwicklungsumgebung von Word die Bibliothek MICROSOFT ACTIVEX DATA OBJECT 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 bei diesem Beispiel im gleichen Verzeichnis wie das Word-Dokument MEMO.DOC. Daher können Sie dieses einheitliche Verzeichnis über die Eigenschaft Path, die Sie auf das aktuelle Dokument

anwenden, abfragen und anschließend mit der Methode `Open` die Datenbank öffnen. Ebenso wird die Tabelle KONTAKTE über die Methode `Open` 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 Kombinationsfeldes auf den Wert `0`.

Die Prozedur für den Start der Userform können Sie aus *Listing 510* entnehmen. Diese Prozedur wird in einem Standardmodul erfasst.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Memo.Doc
' Modul      Modul1
'=====================================================

Sub DatenAbfrage()
  UserForm1.Show
End Sub
```

Listing 510: Die Word-Userform starten

Über den Einsatz der Methode `Show` starten Sie die Userform. Kurz darauf wird das Ereignis `Initialize` ausgelöst, das im Hintergrund auf die Access-Datentabelle KONTAKTE zugreift und das Kombinationsfeld füllt.

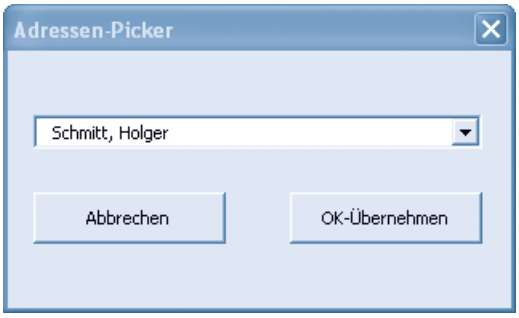


Abbildung 351: Alle Namen stehen im Kombinationsfeld zur Auswahl bereit.

Was jetzt noch fehlt, ist die Prozedur für die Übernahme der Kontaktdaten. Dabei wird die Prozedur aus *Listing 511* hinter die Schaltfläche OK-ÜBERNEHMEN gelegt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Memo.Doc
' Klasse     Userform1
'=====================================================
```

Listing 511: Kontaktdaten in Access-Tabelle finden und ins Dokument übertragen

```
Private Sub cmd_OK_Click()
  Dim objWord As Object
  Dim con As ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim strName As String
  Dim strNachname As String
  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 con = New ADODB.Connection
  With con
    .Provider = "Microsoft.Jet.OLEDB.4.0"
    .Open ThisDocument.Path & "\Umfeld.mdb"
  End With

  Set rst = New ADODB.Recordset
  With rst
    .Open Source:="Kontakte", ActiveConnection:=con, _
      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
  con.Close

  Set rst = Nothing

  On Error Resume Next
  Set objWord = GetObject(, "Word.Application")
  With objWord
    .Selection.GoTo What:=wdGoToBookmark, Name:="Nachname"
```

Listing 511: Kontaktdaten in Access-Tabelle finden und ins Dokument übertragen (Forts.)

```
    .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
    '.ActiveDocument.Save
  End With

  Set objWord = Nothing
  Unload Me
End Sub
```

Listing 511: 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 dann die beteiligte Datenbank sowie die Tabelle KONTAKTE. Setzen Sie die Methode `Find` ein und übergeben 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 die Access-Tabelle gleich danach über die Methode `Close`. Im selben Atemzug schließen Sie auch gleich die Datenbank.

Über die Funktion `GetObject` stellen Sie jetzt 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 den Inhalt der Variablen mit der Methode `TypeText` unmittelbar nach den Textmarken ins Dokument ein.

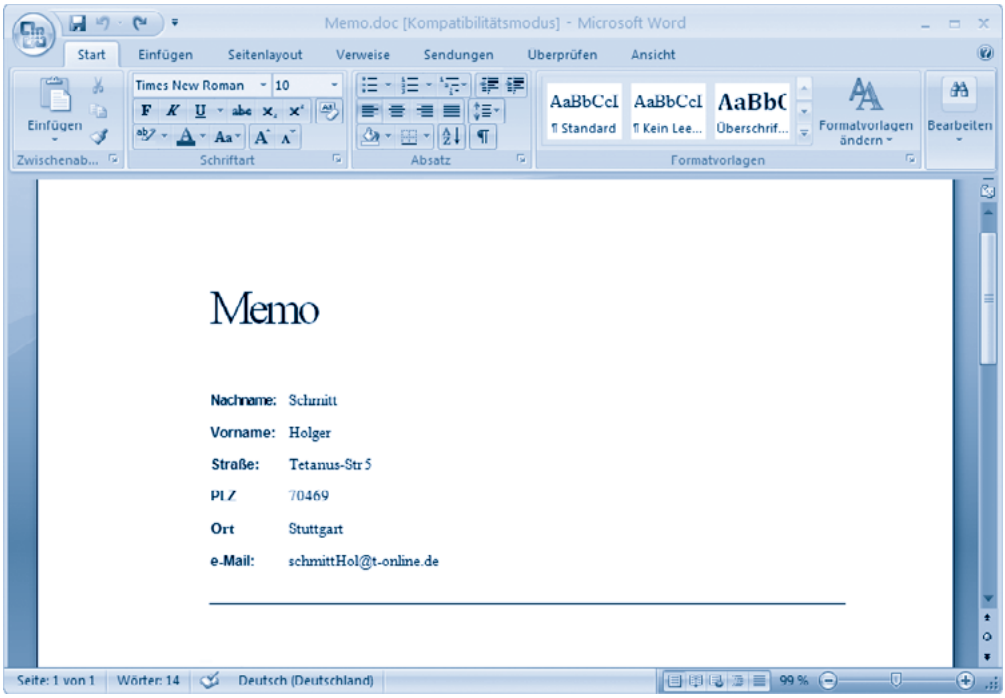


Abbildung 352: Die Daten wurden richtig übertragen.

Access & Excel

Beim Zusammenspiel von Access und Excel haben Sie sowohl die Möglichkeit, Daten nach Excel zu exportieren, als auch, Daten von Excel zu 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.

406 Ist Excel bereits gestartet?

Zu Beginn der Access/Excel-Beispiele prüfen Sie mithilfe der Funktion aus *Listing 512*, ob Excel bereits gestartet ist. Dabei muss als Voraussetzung die Excel-Bibliothek unter EXTRAS/VERWEISE eingebunden sein.

```
'=======================================================
' Auf CD      Buchdaten\Beispiele
' Dateiname   Umfeld.mdb
' Modul       mdlExcel
'=======================================================
```

Listing 512: Funktion, die prüft, ob Excel gestartet ist

```
Function ExcelOffen() As Boolean
  Dim objXlApp As Excel.Application

  On Error Resume Next
  Set objXlApp = GetObject(, "Excel.Application")

  If Err.Number = 0 Then
    ExcelOffen = True
  Else
    ExcelOffen = False
  End If
  Set objXlApp = Nothing
End Function
```

Listing 512: Funktion, die prüft, ob Excel gestartet ist (Forts.)

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 die aufrufende Prozedur aus *Listing 513* zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub StartExcel()
  Dim objXlApp As Object

  If ExcelOffen = False Then
  Set objXlApp = CreateObject("Excel.Application")
    objXlApp.Workbooks.Add
    objXlApp.Visible = True
  Else
    Set objXlApp = GetObject(, "Excel.Application")
    objXlApp.Visible = True
  End If
  Set objXlApp = Nothing
End Sub
```

Listing 513: Soll Excel gestartet werden oder nicht?

In der Prozedur aus *Listing 513* wurde Late-Binding angewendet und die Objektvariable `objXlApp` 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. Erzeugen Sie im Anschluss daran über die Methode `Add` des Objekts `Workbooks` eine neue Arbeitsmappe. Weisen Sie dann der Eigenschaft `Visible` den Wert `True` zu, um die Applikation anzuzeigen.

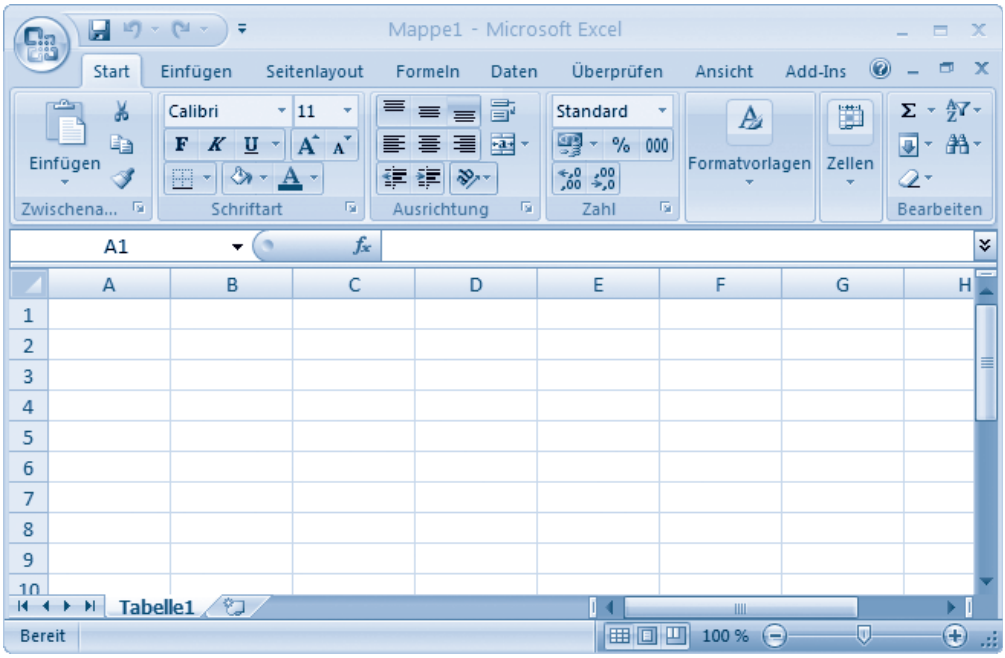


Abbildung 353: Excel wurde mit einer neuen Arbeitsmappe gestartet.

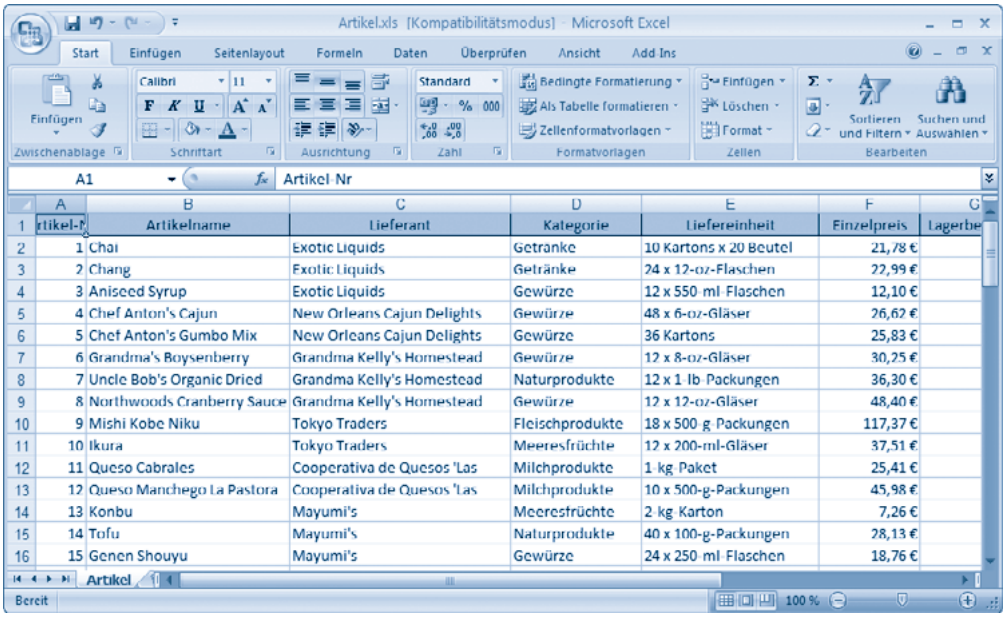
407 Access-Tabelle in eine Excel-Tabelle umwandeln

Im folgenden Beispiel aus *Listing 514* 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 festzulegen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=======================================================

Sub TabelleNachExcelTranferieren()
  DoCmd.OutputTo acOutputTable, "Artikel", acFormatXLS, _
    "C:\Eigene Dateien\Artikel.xls", True
End Sub
```

Listing 514: 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	F	G
1	tikel-N	Artikelname	Lieferant	Kategorie	Liefereinheit	Einzelpreis	Lagerbe
2	1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	21,78 €	
3	2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	22,99 €	
4	3	Aniseed Syrup	Exotic Liquids	Gewürze	12 x 550-ml-Flaschen	12,10 €	
5	4	Chef Anton's Cajun	New Orleans Cajun Delights	Gewürze	48 x 6-oz-Gläser	26,62 €	
6	5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	36 Kartons	25,83 €	
7	6	Grandma's Boysenberry	Grandma Kelly's Homestead	Gewürze	12 x 8-oz-Gläser	30,25 €	
8	7	Uncle Bob's Organic Dried	Grandma Kelly's Homestead	Naturprodukte	12 x 1-lb-Packungen	36,30 €	
9	8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	12 x 12-oz-Gläser	48,40 €	
10	9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	18 x 500-g-Packungen	117,37 €	
11	10	Ikura	Tokyo Traders	Meeresfrüchte	12 x 200-ml-Gläser	37,51 €	
12	11	Queso Cabrales	Cooperativa de Quesos 'Las	Milchprodukte	1-kg-Paket	25,41 €	
13	12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las	Milchprodukte	10 x 500-g-Packungen	45,98 €	
14	13	Konbu	Mayumi's	Meeresfrüchte	2-kg-Karton	7,26 €	
15	14	Tofu	Mayumi's	Naturprodukte	40 x 100-g-Packungen	28,13 €	
16	15	Genen Shouyu	Mayumi's	Gewürze	24 x 250-ml-Flaschen	18,76 €	

Abbildung 354: 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 darauffolgenden Argument `Tabellenname` geben Sie den Namen der zu exportierenden Access-Tabelle an. Über das Argument `Dateiname` geben Sie das Ziel für den Datenexport bekannt. Dabei muss die angegebene Datenquelle nicht einmal existieren. Access legt diese für Sie neu 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.

Achtung

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.

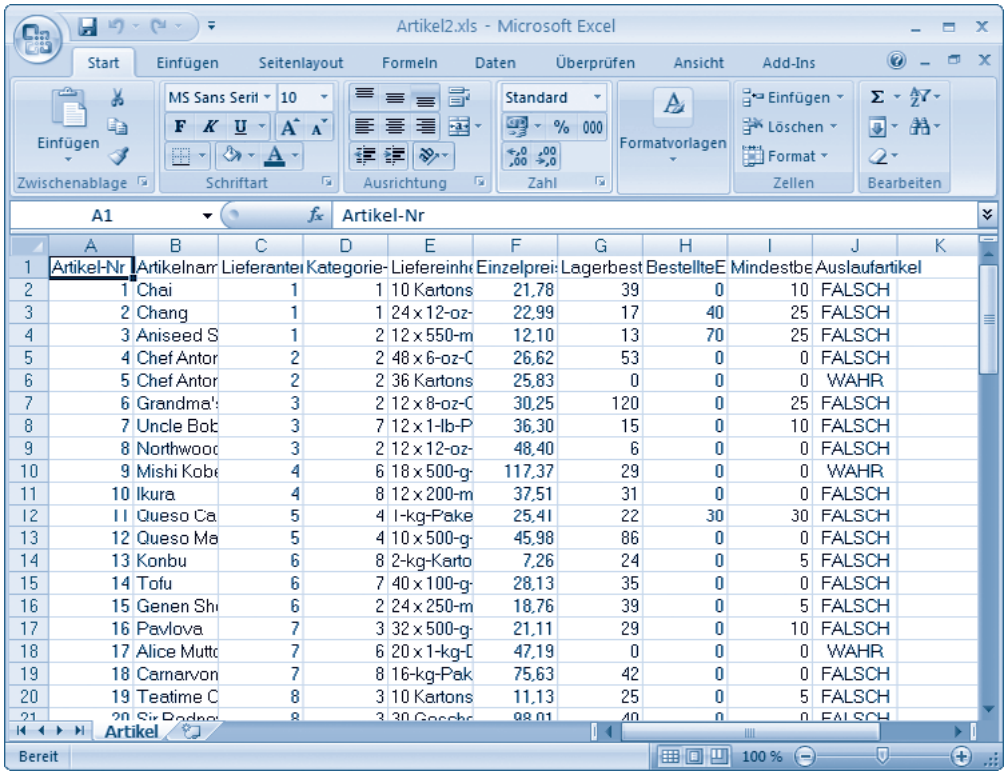


Abbildung 355: Das Ergebnis der Methode TransferSpreadsheet

408 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. Die Prozedur für diese Aufgabe lautet:

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================
```

Listing 515: Eine Excel-Tabelle nach Access importieren

```
Sub TabelleVonExcelImportieren()
  DoCmd.TransferSpreadsheet acImport, acSpreadsheetTypeExcel12, _
    "ArtikelAusExcel", "C:\Eigene Dateien\Artikel2.xls", _
    True
End Sub
```

Listing 515: Eine Excel-Tabelle nach Access importieren (Forts.)

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 der 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)
- `acSpreadsheetTypeExcel12` (Excel 2007)

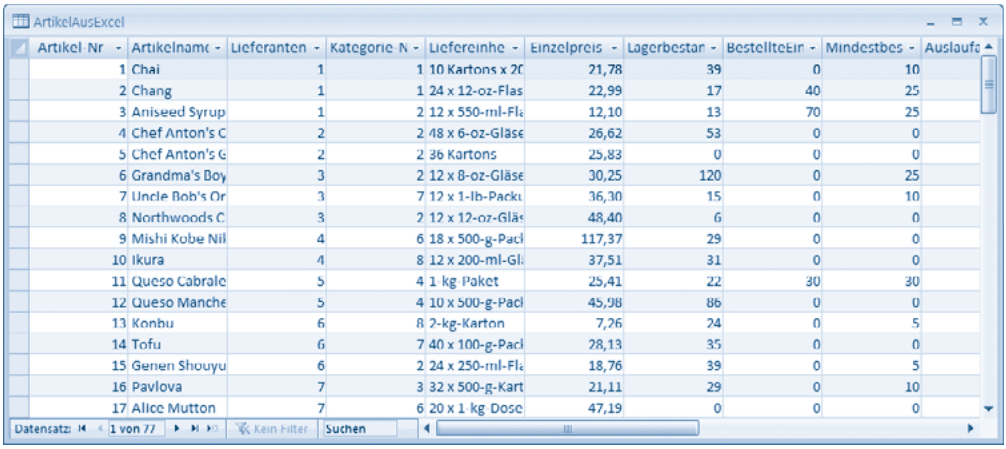


Abbildung 356: Die Excel-Tabelle wurde erfolgreich in Access importiert.

409 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. Dazu 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 in der höchsten auf Ihrem PC zur Verfügung stehenden Version 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 357* sehen können.

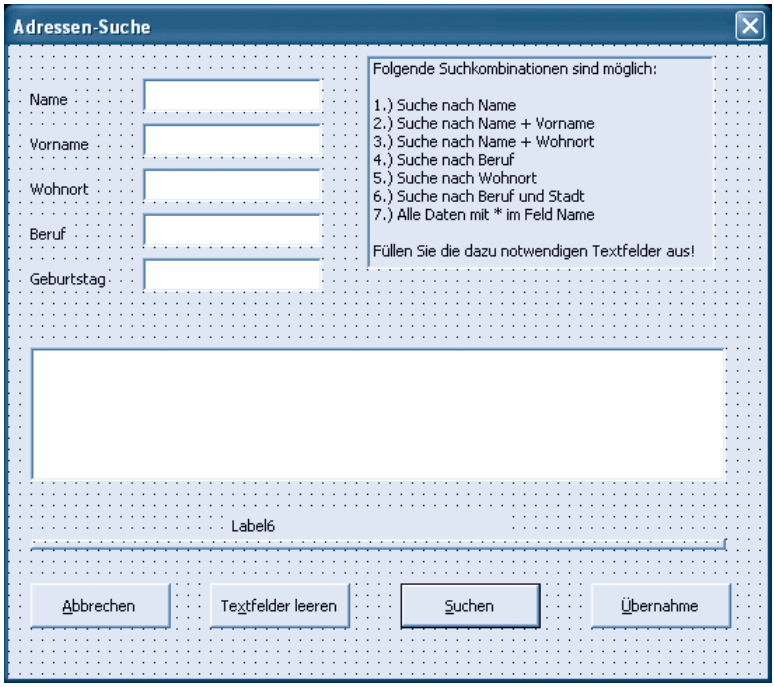


Abbildung 357: Die komfortable Adressen-Suche über Excel

Diese Datenmaske enthält vier Texteingabefelder, über die die diversen Suchen durchgeführt werden können. 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 Name 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 und füllen dazu beide Felder aus.
- Suche nach Wohnort. Hier werden alle Personen aus einer Stadt gesucht.

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 516* sehen können.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' 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
    .Label6.Visible = False
    .Label6.Caption = ""
    .Label6.ForeColor = RGB(0, 0, 255)
    .Label6.Font.Bold = True
  End With
End Sub
```

Listing 516: Einige Aufgaben bereits beim Starten der Userform ausführen

Über die Eigenschaft `ColumnCount` geben Sie an, wie viele Spalten im Listenfeld angelegt werden sollen. Mithilfe der Eigenschaft `ColumnWidth` können Sie die Spaltenbreiten festlegen. Setzen Sie die Eigenschaft `Default` der Schaltfläche SUCHEN auf den Wert `True` um diese zur Standard-Befehlsschaltfläche zu machen.

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 Bezeichnungsfeldes 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.

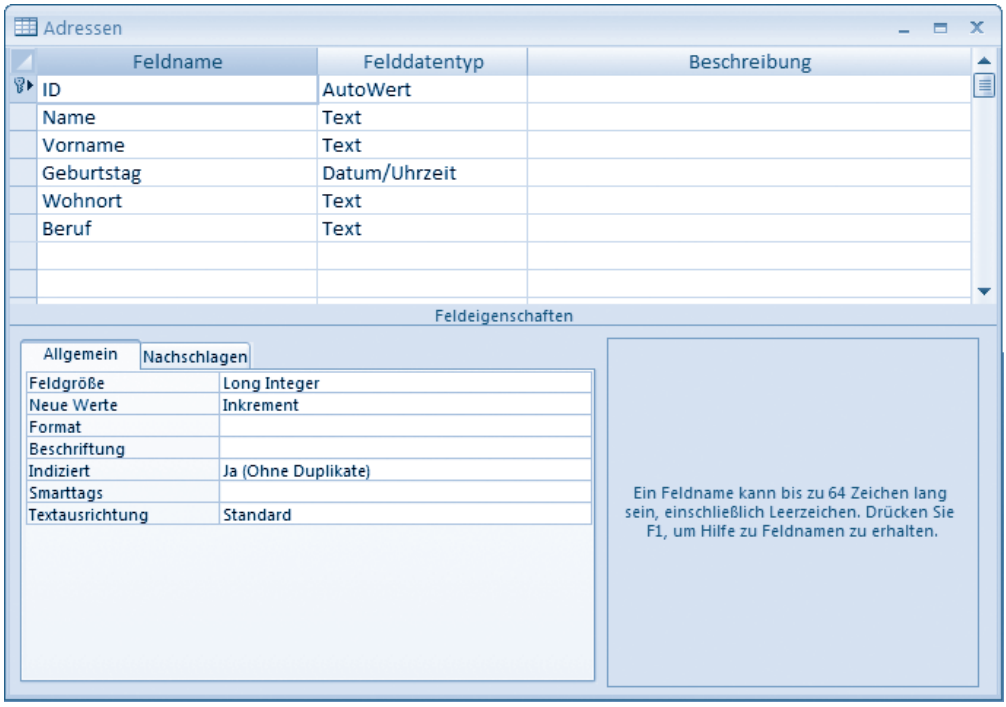


Abbildung 358: Die Datenstruktur der Access-Tabelle Adressen

Die Suchen-Funktion

Die diversen Suchen-Funktionen werden hinter die Schaltfläche SUCHEN gelegt, die Sie übrigens auch über die Tastenkombination Alt + S direkt ausführen können. Hierfür ist die Eigenschaft Accelerator verantwortlich. Diese Eigenschaft können Sie direkt im Eigenschaftenfenster der Entwicklungsumgebung einstellen, indem Sie die Schaltfläche markieren und dann im Eigenschaftenfenster den Buchstaben eingeben, über den Sie in Verbindung mit der Taste Alt die Schaltfläche per Shortcut ausführen möchten. Danach wird der angegebene Buchstaben in der Schaltflächenbeschriftung automatisch unterstrichen.

Legen Sie jetzt den Code aus *Listing 517* direkt hinter die Schaltfläche SUCHEN.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub CommandButton1_Click()
  'Daten in Access-Datenbank suchen und übergeben
  Dim objAccApp As Object
  Dim con As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim str As String
  Dim str2 As String
  Dim intZ As Integer
  Dim bool As Boolean

  intZ = 0
  bool = False

  With UserForm1
    'Listenfeld löschen
    .ListBox1.Clear

    'Alle Datensätze
    If .TextBox1.Value = "*" Then
      str = ""
      str = "Name='" & str & "'"
      bool = True
      GoTo Weiter
    End If

    'Suche nach Beruf
    If .TextBox4.Value <> "" And .TextBox1.Value = "" _
      And .TextBox2.Value = "" And .TextBox3.Value = "" Then
      str = UserForm1.TextBox4.Value
      str = "Beruf='" & str & "'"
    End If

    'Suche nach Beruf und Stadt
    If .TextBox4.Value <> "" And .TextBox1.Value = "" _
      And .TextBox2.Value = "" And .TextBox3.Value <> "" Then
      str = UserForm1.TextBox4.Value
      str2 = UserForm1.TextBox3.Value
      str = "Beruf='" & str & "' AND Wohnort='" & str2 & "'"
    End If
```

Listing 517: Die Suche in der Access-Tabelle durchführen

```
    'Suche nach Stadt
    If .TextBox4.Value = "" And .TextBox1.Value = "" _
      And .TextBox2.Value = "" And .TextBox3.Value <> "" Then
      str = UserForm1.TextBox3.Value
      str = "Wohnort='" & str & "'"
    End If

    'Suche nach Name
    If .TextBox1.Value <> "" And .TextBox2.Value = "" _
      And .TextBox3.Value = "" And .TextBox4.Value = "" Then
      str = UserForm1.TextBox1.Value
      str = "Name='" & str & "'"
    End If

    'Suche nach Name und Vorname
    If .TextBox1.Value <> "" And .TextBox2.Value <> "" _
      And .TextBox3.Value = "" And .TextBox4.Value = "" Then
      str = UserForm1.TextBox1.Value
      str2 = UserForm1.TextBox2.Value
      str = "Name='" & str & "' AND Vorname='" & str2 & "'"
    End If

    'Suche nach Name und Wohnort
    If .TextBox1.Value <> "" And .TextBox2.Value = "" _
      And .TextBox3.Value <> "" And .TextBox4.Value = "" Then
      str = UserForm1.TextBox1.Value
      str2 = UserForm1.TextBox3.Value
      str = "Name='" & str & "' AND Wohnort='" & str2 & "'"
    End If

    '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
```

Listing 517: Die Suche in der Access-Tabelle durchführen (Forts.)

```
Weiter:
  Set objAccApp = CreateObject("Access.Application")
  'objAccApp.Visible = True
  Set rst = New ADODB.Recordset
  con.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
    "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

  'Tabelle öffnen und abfragen
  If bool = False Then
    rst.Open "SELECT * FROM Adressen WHERE (" & str & ")", _
      con, adOpenKeyset, adLockOptimistic
  Else
    rst.Open "SELECT * FROM Adressen", con, adOpenKeyset, adLockOptimistic
  End If

  'Gefundene Sätze in mehrspaltiges Listenfeld einfügen
  With UserForm1
    Do Until rst.EOF
      .ListBox1.AddItem rst!Name
      .ListBox1.Column(1, intZ) = rst!Vorname
      .ListBox1.Column(2, intZ) = rst!Geburtstag
      .ListBox1.Column(3, intZ) = rst!Wohnort
      .ListBox1.Column(4, intZ) = rst!Beruf
      intZ = intZ + 1
     rst.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

  rst.Close
  objAccApp.Quit
  Set rst = Nothing
  Set con = Nothing
End Sub
```

Listing 517: 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 Listenfeldes ansprechen. Des Weiteren setzen Sie die boolesche Variable `bool` standardmäßig auf den Wert `False`. Diese 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. Je nach Fall 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-Datentabellenfeldes mit angegeben werden.

Mithilfe der Funktion `CreateObject` wird ein neues Access-Objekt erstellt. Die Zeile `objAccApp.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. Im Falle, 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 `rst` 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 Listenfeldes füllen. Mithilfe der Methode `MoveNext` wird jeweils der nächste gefundene Datensatz im `Recordset` eingestellt. So werden alle Elemente nach und nach ins Listenfeld übertragen, bis die Bedingung `EOF` (End of File) den Wert `True` besitzt. Konnten über die Suche keine Datensätze ermittelt werden, dann bleibt das Listenfeld leer. In diesem Fall liefert die Eigenschaft `ListCount` den Wert `0`, was eine Beschriftung des Bezeichnungsfeldes mit »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 dann die Access-Anwendung, indem Sie die Methode `Quit` einsetzen, und heben Sie die Objektverweise über das Schlüsselwort `Nothing` auf, um reservierten Arbeitsspeicher wieder freizugeben.

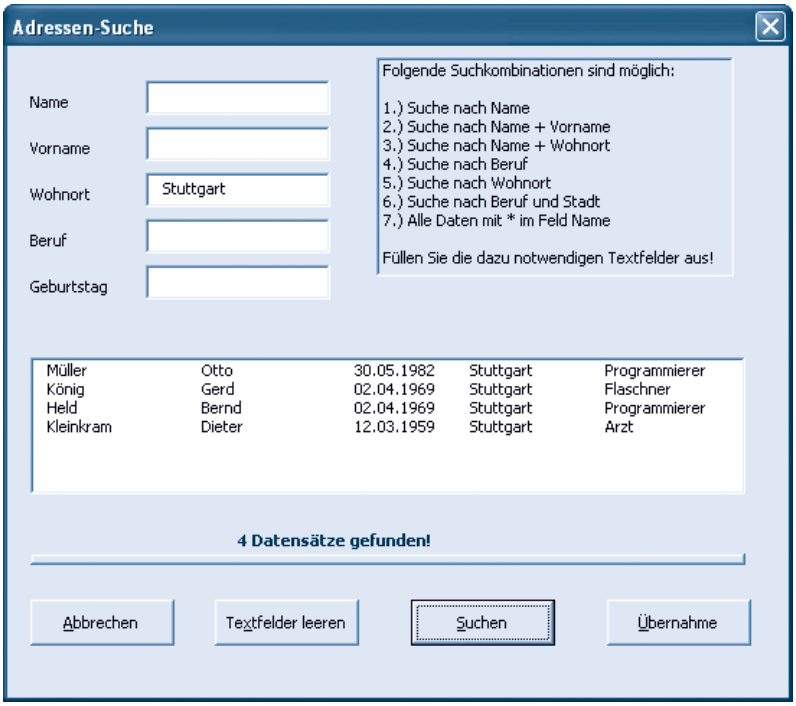


Abbildung 359: Die gefundenen Sätze werden im Listenfeld angezeigt.

Den 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 360*). Dazu können Sie die Einträge direkt aus dem mehrspaltigen Listenfeld der Userform nehmen und in die Textfelder übertragen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub ListBox1_Click()
  'Daten aus Listenfeld in Textfelder übertragen
  Dim intEintrag As Integer
  With UserForm1
    intEintrag = .ListBox1.ListIndex
    .TextBox1.Value = .ListBox1.Column(0, intEintrag)
    .TextBox2.Value = .ListBox1.Column(1, intEintrag)
    .TextBox4.Value = .ListBox1.Column(4, intEintrag)
```

Listing 518: Die Ergebnisse aus dem Listenfeld in die Textfelder übertragen

```
    .TextBox3.Value = .ListBox1.Column(3, intEintrag)
    .TextBox5.Value = .ListBox1.Column(2, intEintrag)
  End With
End Sub
```

Listing 518: 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 518* 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 360: Infos mit einem Klick übertragen

Der Optik zuliebe

Beim Verlassen eines Textfeldes bzw. beim Aktivieren eines Textfeldes werden die beteiligten Textfelder dynamisch formatiert. So bekommt ein Textfeld beim Aktivieren eine rote Hintergrundfarbe, die Schriftfarbe wird auf Weiß gesetzt und das Feld optisch hervorgehoben. Dazu schreiben Sie zwei Prozeduren, die Sie dem *Listing 519* entnehmen können.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 519: Felder optisch hervorheben

Über die Eigenschaft `BackColor` lässt sich der Hintergrund eines Textfeldes 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 Farbmischung `RGB(255, 255, 255)` zuweisen. Zusätzlich geben Sie den Text des Feldes 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 Textfeldes müssen diese Einstellungen wieder zurückgesetzt werden. Die beiden Prozeduren aus *Listing 519* müssen jetzt den Textfeld-Ereignissen `Enter` (= beim Aktivieren) bzw. `Exit` (= beim Verlassen) zugewiesen werden. Exemplarisch sehen Sie diese Zuweisung in *Listing 520* für die beiden ersten Textfelder.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================
```

Listing 520: Den Eintritt und das Verlassen der Textfelder abfangen

```
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 520: Den Eintritt und das Verlassen der Textfelder abfangen (Forts.)

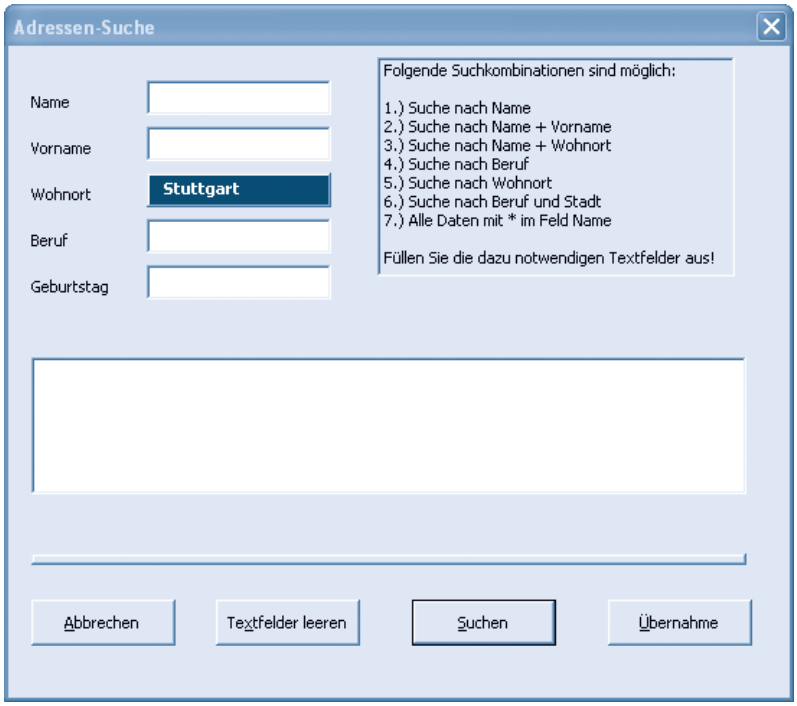


Abbildung 361: Das jeweils aktive Feld wird optisch hervorgehoben.

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 521*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' 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 521: 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 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 Listenfeldes 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 die Prozedur aus *Listing 522*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================
```

Listing 522: 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 522: 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 über eine Zuweisung direkt in diese Zellen »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 liegende Access-Tabelle geschrieben. Dazu wird die Userform aus *Abbildung 362* eingesetzt und die folgende Prozedur aus *Listing 523* ausgeführt.

Abbildung 362: Die Userform für die Datenerfassung

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=====================================================

Private Sub CommandButton2_Click()
  'Daten zurückschreiben
  Dim objAccApp As Object
  Dim con As New ADODB.Connection
  Dim rst As ADODB.Recordset

  Set objAccApp = CreateObject("Access.Application")
  Set rst = New ADODB.Recordset
  con.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
    "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

  rst.Open "Adressen", con, adOpenKeyset, adLockOptimistic
  rst.AddNew
  With UserForm2
    rst!Name = .TextBox1.Value
    rst!Vorname = .TextBox2.Value
    rst!Wohnort = .TextBox3.Value
    rst!Beruf = .TextBox4.Value
    rst!Geburtstag = .TextBox5.Value
    rst.Update
    UserForm2.Label5.Visible = True
    UserForm2.Label5.Caption = "Satz erfasst!"
    rst.Close
  End With

  objAccApp.Quit
  Set rst = Nothing
  Set con = Nothing
End Sub
```

Listing 523: 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 sowie die Datentabelle über die Methode `Open` geöffnet werden. Mithilfe der Methode `AddNew` wird ein neuer, noch leerer Datensatz am Ende der Access-Datentabelle angehängt. Dieser Satz wird danach gefüllt. Dabei werden die Inhalte der Textfelder direkt in die Datenbankfelder geschrieben. Erst mit dem Einsatz der Methode `Update` wird der neue Satz endgültig in die Tabelle geschrieben. Zum Abschluss werden, wie bereits vorher beschrieben, die Datentabelle geschlossen, die Access-Applikation beendet sowie die Objektverweise aufgehoben.

Abbildung 363: Der Datensatz wurde erfolgreich in Access gespeichert.

Beenden der Userform verhindern

Möchten Sie das Beenden der Userform lediglich über die Schaltfläche ABBRECHEN zulassen, dann erfassen Sie folgendes Ereignis aus *Listing 524*:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Klasse     Userform1
'=======================================================

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
  If CloseMode <> 1 Then Cancel = True
End Sub
```

Listing 524: 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 die Prozedur aus *Listing 525*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Modul      Modul1
'=======================================================

Sub DieÄltestenFinden()
  Dim objAccApp As Object
  Dim con As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim intZ As Integer
  Dim varrPersonName(2) As Variant
  Dim varrPersonVorname(2) As Variant
  Dim varrDatum(2) As Variant

  Set objAccApp = CreateObject("Access.Application")
  Set rst = New ADODB.Recordset
  con.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
    "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

  rst.CursorLocation = adUseClient
  rst.Open "Adressen", con, adOpenKeyset, adLockOptimistic
  rst.Sort = "Geburtstag Asc"

  For intZ = 0 To 2
    varrPersonName(intZ) = rst!Name
    varrPersonVorname(intZ) = rst!Vorname
    varrDatum(intZ) = rst!Geburtstag
    rst.MoveNext
  Next intZ

  MsgBox "Die älteste Person heißt " & varrPersonVorname(0) & " " & _
    varrPersonName(0) & " und ist am " & varrDatum(0) & " geboren!" & vbLf & _
    vbLf & "Die zweit-älteste Person heißt " & varrPersonVorname(1) & " " & _
    varrPersonName(1) & " und ist am " & varrDatum(1) & " geboren!" & vbLf & _
    vbLf & "Die dritt-älteste Person heißt " & varrPersonVorname(2) & " " & _
    varrPersonName(2) & " und ist am " & varrDatum(2) & " geboren!", _
    vbInformation, "Alters-Check"

  rst.Close
  objAccApp.Quit

  Set rst = Nothing
  Set con = Nothing
End Sub
```

Listing 525: Die drei ältesten Personen werden ermittelt.

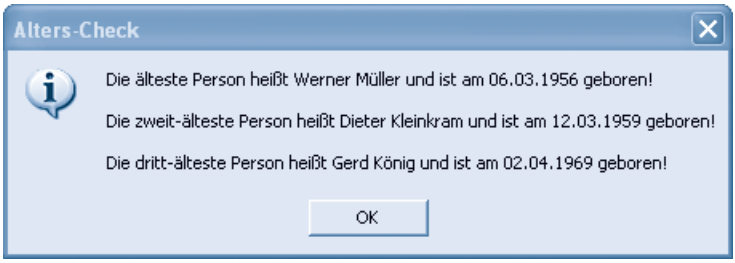


Abbildung 364: Altersauswertung von Excel aus durchführen

Beim Öffnen der Datentabelle wird eine Sortierung durchgeführt und dabei werden in der Prozedur aus *Listing 525* die ersten drei Datensätze, in der Prozedur aus *Listing 526* der erste Datensatz abgegriffen, in einem Datenfeld gespeichert und am Bildschirm ausgegeben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Abfrage.xls
' Modul      Modul1
'=======================================================

Sub JuniorCheck()
  Dim objAccApp As Object
  Dim con As New ADODB.Connection
  Dim rst As ADODB.Recordset
  Dim strPersonName As String
  Dim strPersonVorname As String
  Dim dtmPerson As Date

  Set objAccApp = CreateObject("Access.Application")
  Set rst = New ADODB.Recordset
  con.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
    "Data source=" & ThisWorkbook.Path & "\Umfeld.mdb;"

  rst.CursorLocation = adUseClient
  rst.Open "Adressen", con, adOpenKeyset, adLockOptimistic
  rst.Sort = "Geburtstag Desc"

  strPersonName = rst!Name
  strPersonVorname = rst!Vorname
  dtmPerson = rst!Geburtstag

  MsgBox "Die jüngste Person heißt " & strPersonVorname & " " & _
  strPersonName & " und ist am " & dtmPerson & " geboren!", _
  vbInformation, "Junior-Check"
```

Listing 526: Die jüngste Person ermitteln

```
  rst.Close
  objAccApp.Quit

  Set rst = Nothing
  Set con = Nothing
End Sub
```

Listing 526: Die jüngste Person ermitteln (Forts.)

410 Excel-Prozedur starten

Soll von Access aus auf eine Excel-Prozedur zugegriffen werden, dann wenden Sie, wie in *Listing 527* gezeigt, die Methode `Run` an.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub ExcelProzedurStarten()
  Dim objXlApp As Object

  On Error GoTo Fehler
  Set objXlApp = CreateObject("Excel.Application")

  objXlApp.Workbooks.Open "C:\Eigene Dateien\Mappe1.xls"
  objXlApp.Run "Modul1.MeldungAnzeigen"
  objXlApp.activeworkbook.Close
  objXlApp.Quit

  Set objXlApp = Nothing
  Exit Sub

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

Listing 527: Excel-Prozedur aus Access starten

Setzen Sie die Methode `Open` ein, um die Excel-Arbeitsmappe zu öffnen, die die zu startende Prozedur enthält. Führen Sie danach die Methode `Run` aus, der Sie den Namen der Prozedur als Parameter übergeben. Es kann dabei auch nicht schaden, genau anzugeben, in welchem Modul sich die Prozedur befindet. Schließen Sie nach dem Ablauf der Prozedur die Excel-Arbeitsmappe über die Methode `Close` und beenden Sie die Excel-Anwendung mithilfe der Methode `Quit`. Heben Sie danach den Objektverweis `objXlApp` wieder auf, um den reservierten Speicher wieder freizugeben.

411 Auf Excel-Funktionen zugreifen

Als Worksheet-Funktionen stehen in Excel einige Funktionen zur Verfügung, die in Access 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
' Dateiname  Umfeld.mdb
' Modul      mdlExcel
'=====================================================

Sub ExcelFunktionAufrufen()
  Dim objXlApp As Object

  On Error GoTo Fehler
  Set objXlApp = CreateObject("Excel.Application")

  MsgBox objXlApp.WorksheetFunction.Rept("*-*", 10)
  objXlApp.Quit

  Set objXlApp = Nothing
  Exit Sub

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

Listing 528: 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 365: Eine Zeichenfolge wird wiederholt.

Access & PowerPoint

Zwischen dem reinen Präsentationsprogramm PowerPoint und der Datenbank Access können Daten ausgetauscht werden oder es kann sogar eine Steuerung von Access direkt aus PowerPoint erfolgen.

412 Ist PowerPoint bereits gestartet?

Auch bei der Anwendung PowerPoint wird im Zusammenspiel mit Access zuerst einmal geprüft, ob PowerPoint bereits gestartet ist. Diese Aufgabe erledigt die Funktion aus *Listing 529*. Dabei muss die PowerPoint-Bibliothek unter EXTRAS/VERWEISE als Voraussetzung eingebunden sein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlPP
'=====================================================

Function PowerpointOffen() As Boolean
  Dim objPpApp As PowerPoint.Application

  On Error Resume Next
  Set objPpApp = GetObject(, "Powerpoint.Application")

  If Err.Number = 0 Then
    PowerpointOffen = True
  Else
    PowerpointOffen = False
  End If
  Set objPpApp = Nothing
End Function
```

Listing 529: 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 die aufrufende Prozedur aus *Listing 530* zurück.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlPP
'=====================================================
```

Listing 530: PowerPoint wird je nach Status gestartet oder nicht.

```
Sub StartPowerpoint()
  Dim objPpApp As Object

  If PowerpointOffen = False Then
    Set objPpApp = CreateObject("Powerpoint.Application")
    objPpApp.Presentations.Add
    objPpApp.Visible = True
  End If
  Set objPpApp = Nothing
End Sub
```

Listing 530: PowerPoint wird je nach Status gestartet oder nicht. (Forts.)

In der Prozedur aus *Listing 530* wurde Late-Binding eingesetzt und die Objektvariable `objPpApp` 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 die Methode `Add` des Objekts `Presentations` an, um eine neue Präsentation einzufügen. Weisen Sie danach der Eigenschaft `Visible` den Wert `True` zu, um die Applikation anzuzeigen.

413 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 366* integriert.

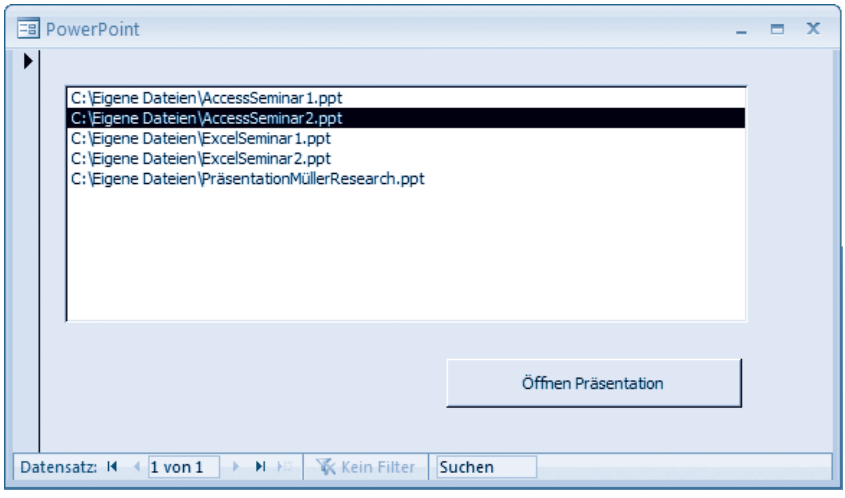


Abbildung 366: Das Formular zum Anzeigen von PowerPoint-Dateien

Beim Öffnen des Formulars sollen alle in einem bestimmten Verzeichnis gefundenen PowerPoint-Präsentationen im Listenfeld angezeigt werden. Für diese Aufgabe setzen Sie das Ereignis `Form_Load` aus *Listing 531* ein, das automatisch ausgeführt wird, wenn das Formular geöffnet wird.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Klasse     Form_PowerPoint
'=====================================================

Private Sub Form_Load()
  Dim strVerz As String
  Dim strDatei As String

  strVerz = "C:\Eigene Dateien\"
  strDatei = Dir(strVerz & "*.ppt")

  Do Until strDatei = ""
    Liste0.AddItem strVerz & strDatei
    strDatei = Dir
  Loop
  Exit Sub

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

Listing 531: PowerPoint-Präsentationen suchen

Führen Sie eine Suche über die Funktion `Dir` durch und ermitteln dabei mit einer Schleife alle PowerPoint-Präsentationen mit der Dateinamenerweiterung .PPT innerhalb eines Verzeichnisses. Innerhalb der Schleife wenden Sie die Methode `AddItem` an, um das Listenfeld zu füllen.

Wenn eine PowerPoint-Präsentation im Listenfeld ausgewählt wird und auf die Schaltfläche ÖFFNEN PRÄSENTATION geklickt wird, dann wird die Prozedur aus *Listing 532* aufgerufen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Klasse     Form_PowerPoint
'=====================================================

Private Sub Befehl2_Click()
  'PowerPoint-Präsentation öffnen
  Dim objPpApp As New PowerPoint.Application
```

Listing 532: PowerPoint-Applikation erstellen und Präsentation öffnen

```
  objPpApp.Visible = True
  objPpApp.Presentations.Open Liste0.Value
End Sub
```

Listing 532: PowerPoint-Applikation erstellen und Präsentation öffnen (Forts.)

Erstellen Sie zunächst über die Deklaration `Dim objPpApp As New PowerPoint.Application` ein neues PowerPoint-Objekt. Danach haben Sie automatisch Zugriff auf alle Methoden und Eigenschaften, die dieses Objekt standardmäßig anbietet. Weisen Sie dann 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.

Access mit Textdateien

In Access haben Sie über Menüfunktionen standardmäßig die Möglichkeit, Tabellen als Textdateien zu speichern. Ebenso können Sie Textdateien in Access importieren. Selbstverständlich können Sie beide Aktionen auch per Prozedur durchführen.

414 Textdateien speichern

Den Vorgang des Speicherns einer Tabelle in eine Textdatei können Sie über eine Prozedur automatisieren. So speichern Sie im folgenden Beispiel die Tabelle ARTIKEL in eine Textdatei. Als Trennzeichen verwenden Sie dabei das Semikolon.

Beim Datentransfer aus *Listing 533* sollen nur die Datenfelder ARTIKELNAME, LIEFEREINHEIT, MINDESTBESTAND, LAGERBESTAND und BESTELLTE EINHEITEN in die Textdatei übertragen werden. Sie brauchen bei der folgenden Prozedur aus *Listing 533* die Tabelle ARTIKEL vorher nicht zu öffnen. Das Öffnen, Verarbeiten und Transferieren der Daten in die Textdatei wird automatisch vorgenommen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=======================================================

Sub TabelleAlsTextdateiSpeichern()
  Dim rst As New ADODB.Recordset
  Dim strZeile As String
  Dim intZ As Long

  On Error GoTo Fehler
  rst.Open "Artikel", CurrentProject.Connection
  Open "C:\Eigene Dateien\Artikel.csv" For Output As #1
```

Listing 533: Eine Tabelle in eine Textdatei schreiben – Variante 1

```
  intZ = 0

  Do Until rst.EOF
    strZeile = strZeile & rst!Artikelname & ";" & _
      rst!Liefereinheit & ";" & rst!Einzelpreis & ";" & _
      rst!Mindestbestand & ";" & rst!Lagerbestand & ";" & _
      rst!BestellteEinheiten
    rst.MoveNext
    Print #1, strZeile
    intZ = intZ + 1
    strZeile = ""
  Loop

  Close #1
  MsgBox "Transfer beendet! Es wurden " & intZ _
    & " Sätze übertragen!"
  rst.Close
Exit Sub

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

Listing 533: 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 Variable zwischenzuspeichern. In der folgenden Variable `intZ` sollen die übertragenen Datensätze gezählt werden. Der Wert soll später ausgegeben werden. Öffnen Sie daraufhin mithilfe der Methode `Open` die Tabelle ARTIKEL in der aktuell geöffneten Datenbank. Mit der Anweisung `Current Project.Connection` erhalten Sie eine Referenz auf die aktuell geöffnete Datenbank.

Öffnen Sie dann über die Methode `Open` die Textdatei, in die die Daten hineintransferiert 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 533* 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 die Prozedur bei mehreren Starts hintereinander die

Daten immer unten an die Textdatei anhängen, was zur Folge hätte, dass Sie die Datensätze mehrfach vorliegen hätten.

Das nächste optionale Argument `Access-Zugriff` legt die Operation fest, die mit der geöffneten Datei ausgeführt werden soll. Dabei setzen 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 von anderen Anwendungen mit der geöffneten Datei 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 diese mit einer anderen Dateinummer geöffnet werden kann. Wenn Sie dennoch versuchen, in eine bereits so geöffnete Textdatei zu schreiben, werden Sie daran gehindert und erhalten eine Fehlermeldung angezeigt.

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 (Bytes) 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 Überprüfung nehmen Sie über die Eigenschaft `EOF` (End of File) vor. Sie dürfen allerdings nicht vergessen, den Datenzeiger mithilfe der Methode `MoveNext` jeweils einen Datensatz weiterzusetzen, 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, die Sie beim Öffnen der Textdatei eingesetzt haben. Nur so wird gewährleistet, dass der »Ausdruck« auch in die gewünschte Textdatei kommt.

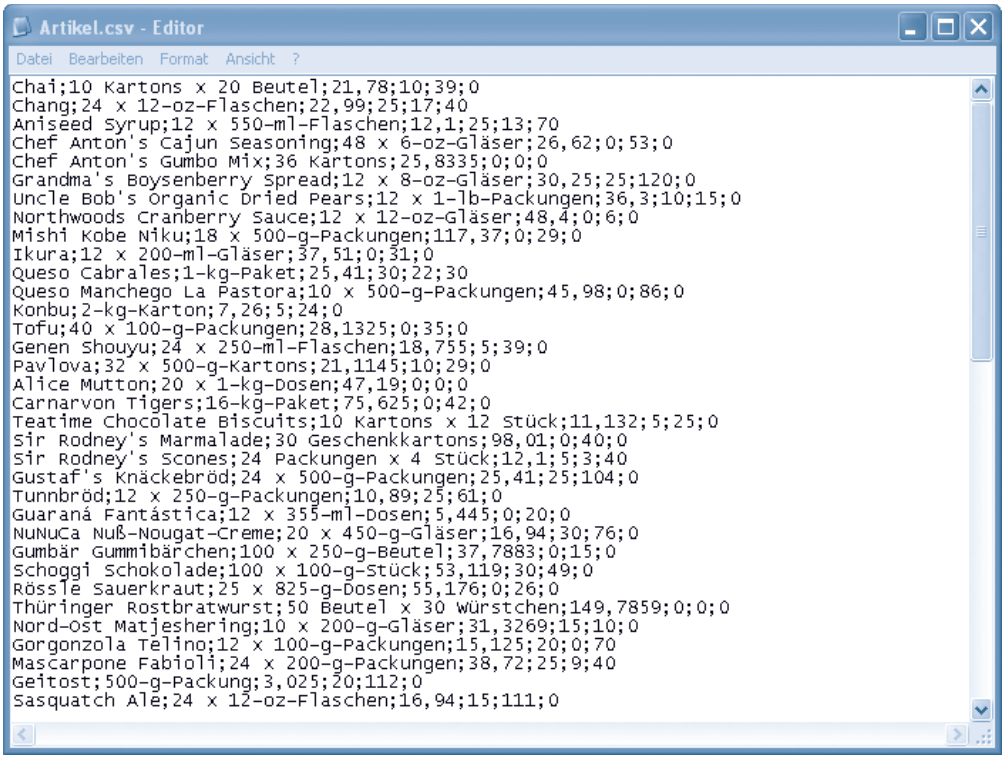


Abbildung 367: Die CSV-Textdatei wurde aus der Tabelle Artikel erzeugt.

415 Eine weitere Methode Textdateien zu speichern

Eine weitere Möglichkeit, um Access-Tabellen in Textdateien zu speichern, bietet das `DoCmd`-Objekt. So speichert die Prozedur aus *Listing 534* die Tabelle ARTIKEL im Verzeichnis C:\EIGENE DATEIEN unter dem Namen ARTIKEL.TXT.

```
Sub TabelleTransferieren()
  DoCmd.OutputTo acOutputTable, "Artikel", acFormatTXT, _
    "C:\Eigene Dateien\Artikel.txt", True
End Sub
```

Listing 534: Eine Tabelle in eine Textdatei schreiben – Variante 3

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.

416 Codes sichern

Die Methode OutputTo bietet Ihnen die Möglichkeit, Ihre Programmierung zu sichern. Die Prozedur aus *Listing 535* sichert das Modul mdlWord in der Textdatei CODE.TXT im Verzeichnis C:\EIGENE DATEIEN.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=====================================================

Sub ModulTransferieren()
  DoCmd.OutputTo acOutputModule, "mdlWord", _
    acFormatTXT, "C:\Eigene Dateien\Code.txt", True
End Sub
```

Listing 535: Codes in einer Textdatei speichern

Indem Sie die Konstante acOutputModule verwenden, erkennt Access, dass es dabei aus der Entwicklungsumgebung das Modul mdlWord exportieren soll. Mit der Konstante acFormatTXT legen Sie fest, dass dieser Transfer im Textformat stattfinden soll.

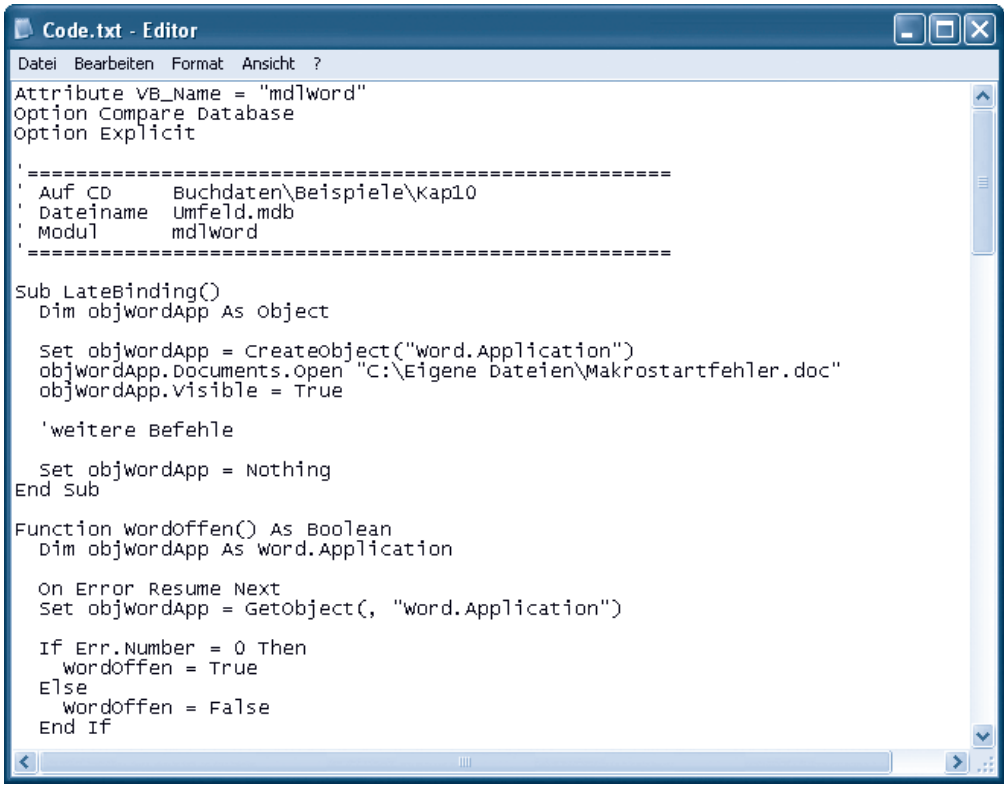


Abbildung 368: Der Code des Moduls wurde in eine Textdatei exportiert.

Haben Sie mehrere Module in Ihrer Datenbank untergebracht und möchten Sie diese alle sichern, dann setzen Sie die Prozedur aus *Listing 536* ein.

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

  Set dbs = Nothing
End Sub
```

Listing 536: Alle Prozeduren in mehreren Textdateien speichern

Im Auflistungsobjekt `AllModules` sind alle Module der Datenbank verzeichnet. Sie können damit ganz elegant ein Modul nach dem anderen innerhalb 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.

417 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 im *Kapitel Die Access-Objekte* dieses Buches nachlesen.

Eine alternative Möglichkeit ist, die Textdatei selbst über eine Prozedur einzulesen. Im folgenden Beispiel aus *Listing 537* werden aus der Textdatei ARTIKEL.CSV die Informationen ARTIKELNAME, LIEFEREINHEIT und EINZELPREIS in die Tabelle ARTIKEL2 eingelesen.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=======================================================

Sub TextdateiInTabelleEinlesen()
  Dim rst As New ADODB.Recordset
  Dim strZeile As String
  Dim intPos1 As Integer

  On Error GoTo Fehler
  rst.Open "Artikel2", CurrentProject.Connection, , adLockOptimistic
  Open "C:\Eigene Dateien\Artikel.csv" For Input As #1

  Do While Not EOF(1)
    rst.AddNew
    Line Input #1, strZeile
    rst!Artikelname = Left(strZeile, InStr(strZeile, ";") - 1)
    intPos1 = InStr(strZeile, ";") + 1
    strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
    rst!Liefereinheit = Left(strZeile, InStr(strZeile, ";") - 1)
    intPos1 = InStr(strZeile, ";") + 1
    strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
    rst!Einzelpreis = Left(strZeile, InStr(strZeile, ";") - 1)
    rst.Update
  Loop
  Close #1
  rst.Close
  Exit Sub

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

Listing 537: Teile einer Textdatei einlesen – Variante 1

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 der Prozedur die Datentabelle sowie die Textdatei über die Methode `Close`.

Abbildung 369: Die Daten wurden erfolgreich eingelesen.

Hinweis

Wenn Sie sich die Prozedur aus *Listing 537* noch einmal ansehen, dann sehen Sie, dass einige Zeilen darin redundant sind. Wie kann diese Prozedur umgeschrieben werden, damit sie 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 in *Listing 538* gemacht wird.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Umfeld.mdb
' Modul      mdlTXT
'=====================================================

Sub TextdateiInTabelleEinlesenSchneller()
  Dim rst As New ADODB.Recordset
  Dim strZeile As String
  Dim intZ As Integer
  Dim intPos1 As Integer
```

Listing 538: Teile einer Textdatei einlesen – Variante 2

```
  On Error GoTo Fehler
  rst.Open "Artikel3", CurrentProject.Connection, , adLockOptimistic
  Open "C:\Eigene Dateien\Artikel.csv" For Input As #1

  Do While Not EOF(1)
    Line Input #1, strZeile
    rst.AddNew
    For intZ = 1 To 3
      rst.Fields(intZ) = Left(strZeile, InStr(strZeile, ";") - 1)
      intPos1 = InStr(strZeile, ";") + 1
      strZeile = Mid(strZeile, intPos1, Len(strZeile) - InStr(strZeile, ";"))
    Next intZ
    rst.Update
  Loop

  Close #1
  rst.Close
  Exit Sub

Fehler:
  MsgBox Err.Number & " " & Err.Description
  Close #1
End Sub
```

Listing 538: Teile einer Textdatei einlesen – Variante 2 (Forts.)

Über die Zählvariable `intZ` können Sie in einer Schleife die Datenbankfelder ARTIKELNAME (1. Feld) bis EINZELPREIS (3. 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 in der Variablen `strZeile` von links nach rechts vor.

Symbol- und Menüleisten

Hinweis

Bei Access 2007 gibt es keine Menü- und Symbolleisten mehr. Diese wurden durch die sogenannte Multifunktionsleiste, im Englischen THE RIBBON, ersetzt. Die Multifunktionsleiste kann mithilfe der Auszeichnungssprache XML programmiert werden.

Der erste Teil des Kapitels richtet sich an Anwender der Access-Versionen 2000 bis 2003. Im zweiten Teil finden Sie eine Einführung in die Programmierung von Ribbons in Access 2007.

Symbol- und Menüleisten bis Access 2003

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.

418 Leisten identifizieren

Die folgende Prozedur in *Listing 539* schreibt alle Leisten von Access mit deren eindeutigem ID-Wert, Namen und Typ in den Direktbereich.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub LeistenIdentifizieren()
  Dim intZ As Integer
  Dim str As String

  For intZ = 1 To CommandBars.Count

  Select Case CommandBars(intZ).Type
    Case 0
      str = "Symbolleiste"
    Case 1
      str = "Menüleiste"
    Case 2
      str = "Kontextmenü"
```

Listing 539: Leisten über die Eigenschaft Type identifizieren

```
    Case Else
      str = "nicht ermittelbar"
  End Select

  Debug.Print intZ & " --> " & _
    CommandBars(intZ).Name & " --> " & str
  Next intZ
End Sub
```

Listing 539: Leisten über die Eigenschaft Type identifizieren (Forts.)

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 157*.

Index	Konstante	Befehlsleiste
0	msoBarTypeNormal	Symbolleiste
1	msoBarTypeMenuBar	Menüleiste
2	msoBarTypePopup	Kontextmenü

Tabelle 157: Die möglichen Leistentypen 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.

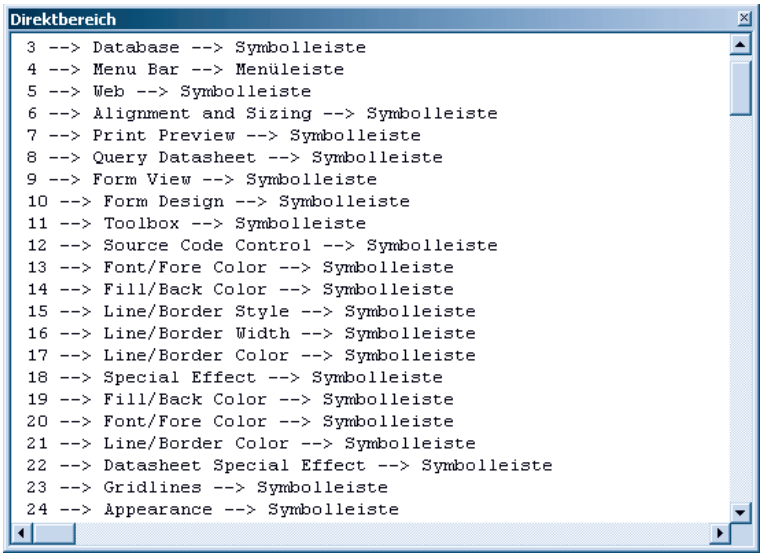


Abbildung 370: Alle Access-Leisten im Überblick

419 Symbolleiste ein- und ausblenden

In der folgenden Prozedur blenden Sie die Symbolleiste DATENBANK ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteDatenbankAnzeigen()
  Application.CommandBars("Database").Visible = True
End Sub
```

Listing 540: Symbolleiste einblenden – Variante 1

In der folgenden Prozedur aus *Listing 541* blenden Sie die Symbolleiste DATENBANK aus.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteDatenbankAusblenden()
  Application.CommandBars("Database").Visible = False
End Sub
```

Listing 541: Symbolleiste ausblenden – Variante 1

Mithilfe der Eigenschaft `Visible` können Sie die Symbolleiste 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 542* blenden Sie die Symbolleiste WEB ein.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteEinblendenAlternative()
  DoCmd.ShowToolbar "Web", acToolbarYes
End Sub
```

Listing 542: Symbolleiste einblenden – Variante 2

Um die Symbolleiste WEB wieder auszublenden, benutzen Sie dieselbe Methode, übergeben aber eine andere Konstante.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteAusblendenAlternative()
  DoCmd.ShowToolbar "Web", acToolbarNo
End Sub
```

Listing 543: Symbolleiste ausblenden – Variante 2

420 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 544*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

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 544: Alle Symbolleisten anzeigen lassen

Um fast alle Symbolleisten auszublenden, starten Sie die Prozedur aus *Listing 545*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub AlleSymbolleistenAusblenden()
  Dim intZ As Integer
```

Listing 545: Fast alle Symbolleisten ausblenden

```
  For intZ = 3 To Application.CommandBars.Count
    On Error Resume Next
    CommandBars(intZ).Visible = False
  Next intZ
End Sub
```

Listing 545: Fast alle Symbolleisten ausblenden (Forts.)

Lassen Sie dabei aber die Symbolleiste DATENBANK unberührt, indem Sie die Schleife erst mit dem Index `3` beginnen lassen.

421 Symbolleisten-IDs ermitteln

Jede einzelne Symbolschaltfläche in Access hat eine eindeutige ID, über die Sie die Symbolschaltfläche ansprechen können. Wie können Sie aber genau diese ID in Access ermitteln? In der Prozedur aus *Listing 546* werden alle Symbole der Symbolleiste DATENBANK mit ihrem Namen und ihrer eindeutigen ID in den Direktbereich geschrieben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolIDsAuflisten()
  Dim intZ As Integer
  Dim cb As CommandBar

  Set cb = Application.CommandBars("Database")

  For intZ = 1 To cb.Controls.Count
    Debug.Print cb.Controls(intZ).Caption & " --> " & _
      cb.Controls(intZ).Id
  Next intZ
End Sub
```

Listing 546: Alle Symbol-IDs auflisten

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.

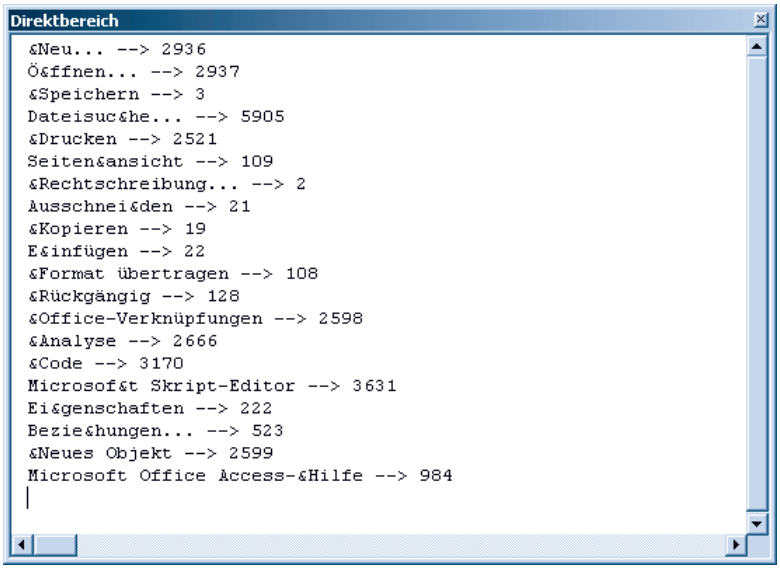


Abbildung 371: Alle IDs der Symbolleiste Datenbank werden dokumentiert.

422 Leisten und Befehle ermitteln

Im nächsten Listing schreiben Sie die Namen aller Leisten und ihrer dazugehörigen Befehle in den Direktbereich.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub CommandBarsAuflisten()
  Dim intZ As Integer
  Dim intSz As Integer
  Dim cbs As Object

  Set cbs = Application.CommandBars

  For intZ = 1 To cbs.Count
    Debug.Print "Das ist die Leiste: " & cbs(intZ).Name
    For intSz = 1 To cbs(intZ).Controls.Count
      Debug.Print cbs(intZ).Controls(intSz).Caption
    Next intSz
    Debug.Print vbLf
  Next intZ
End Sub
```

Listing 547: 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`.

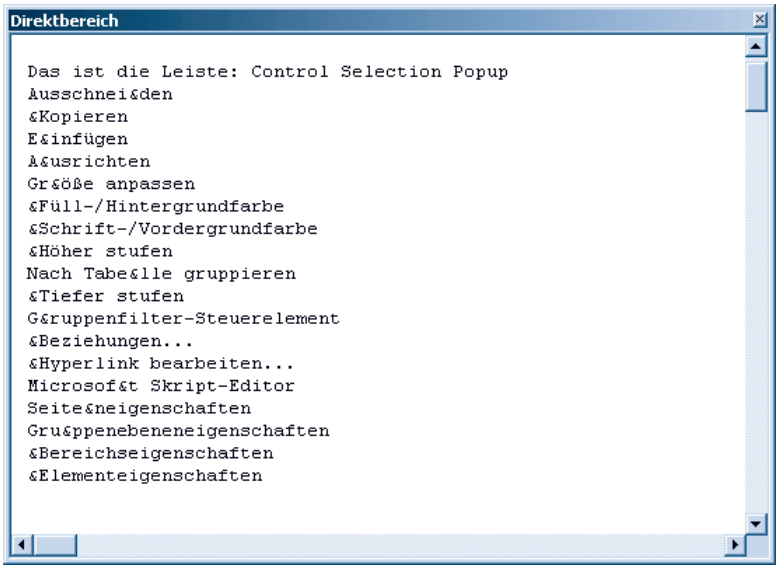


Abbildung 372: Die Namen sowie Symbole aller Symbolleisten dokumentieren

423 Symbolschaltflächen (de)aktivieren

Einzelne Symbole können Sie direkt ansprechen, indem Sie die genaue Anordnung in der Symbolleiste über die Eigenschaft `Controls` angeben.

Im Beispiel aus *Listing 548* wird in der Symbolleiste DATENBANK das Symbol NEU deaktiviert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub SymbolDeaktivieren()
  Dim cbc As CommandBarControl

  Application.CommandBars("Database").Controls(1).Enabled = False
End Sub
```

Listing 548: 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.

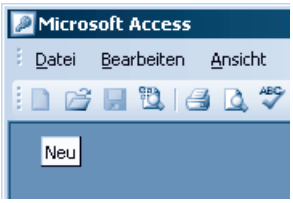


Abbildung 373: Das Symbol Neu ist deaktiviert worden.

Um diese Symbolschaltfläche wieder zu aktivieren, starten Sie die Prozedur aus *Listing 549*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolAktivieren()
  Dim cbc As CommandBarControl

  Application.CommandBars("Database").Controls(1).Enabled = True
End Sub
```

Listing 549: Das Symbol Neu wieder aktivieren

424 Neue Symbolleiste erstellen

Wenn Sie Ihre eigene Symbolleiste anlegen möchten, die die für Sie wichtigen 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 können Sie dem *Listing 550* entnehmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub NeueSymbolleisteAnlegen()
  Dim cb As CommandBar
```

Listing 550: Eine neue Symbolleiste anlegen

```
  On Error Resume Next
  Set cb = CommandBars.Add("Neue Symbolleiste")

  With cb
    .Visible = True
    .Top = 400
    .Left = 70
  End With
End Sub
```

Listing 550: Eine neue Symbolleiste anlegen (Forts.)

Deklarieren Sie im ersten Schritt eine Objektvariable vom Typ `CommandBar`. Mit der Methode `Add` fügen Sie eine neue Symbolleiste ein und weisen sie der Objektvariable 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 Prozedurfehler kommt. Die Methode `Add` wird in diesem Fall einfach ignoriert.

425 Symbolleiste löschen

Um eine Symbolleiste zu löschen, setzen Sie die Methode `Delete` aus *Listing 551* ein. Sehen Sie sich dazu wieder die Anweisung `On Error Resume Next` an, die einen Prozedurfehler verhindert, wenn versucht wird, die bereits gelöschte Symbolleiste beispielsweise erneut zu löschen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolleisteLöschen()
  On Error Resume Next
  Application.CommandBars("Neue Symbolleiste").Delete
End Sub
```

Listing 551: Symbolleiste löschen

426 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. Dieser ID ist eine eindeutige Funktion hinterlegt.

Neben der ID gibt es auch noch die sogenannte FaceId. Diese FaceId ist ebenso eindeutig und gibt das Aussehen des Symbols wieder.

Wie können Sie aber genau diese FaceId in Access ermitteln? In der folgenden Aufgabe aus *Listing 552* werden Sie die ersten 1.000 Symbole in eine eigene Symbolleiste einfügen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub SymbolFaceIDsErmitteln()
  Dim cb As CommandBar
  Dim cbc As CommandBarControl
  Dim intZ As Integer

  On Error Resume Next
  Set cb = Application.CommandBars.Add("Symbole", msoBarFloating)

  For intZ = 1 To 1000
    Set cbc = cb.Controls.Add(msoControlButton)
    With cbc
      .FaceId = intZ
      .TooltipText = intZ
    End With
  Next intZ

  cb.Visible = True
End Sub
```

Listing 552: Symbolleiste mit Symbolen bestücken

Deklarieren Sie im ersten Schritt zwei Objektvariable: Die Objektvariable cb repräsentiert später die neue Symbolleiste. Über die Objektvariable cbc 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 tausend 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.

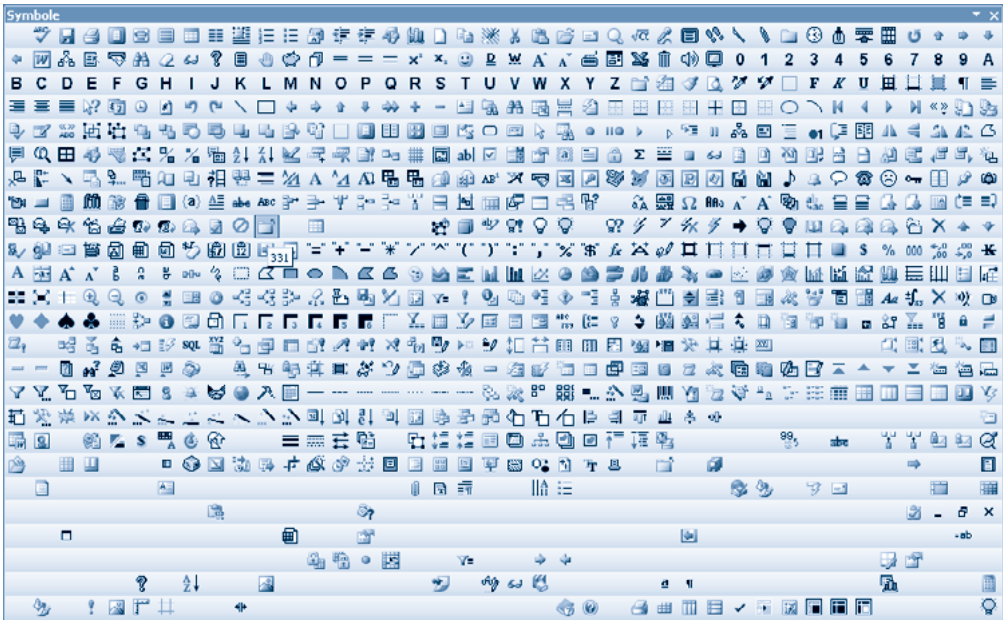


Abbildung 374: Die ersten 1.000 Symbole

427 Symbolschaltflächen-IDs ermitteln

Sie haben nun das Aussehen der einzelnen Symbole über die FaceId herausgefunden. Es fehlt Ihnen nun noch die ID der einzelnen Symbole, die die Funktion der Symbolschaltfläche festlegt.

Im folgenden Beispiel aus *Listing 553* wird die Symbolleiste DATENBANK untersucht. Alle IDs der Symbole dieser Schaltfläche werden in den Direktbereich geschrieben.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub IDsErmitteln()
  Dim cb As CommandBar
  Dim cbc As CommandBarControl

  On Error Resume Next
  Set cb = Application.CommandBars("Database")
```

Listing 553: Die eindeutigen IDs von Symbolen ermitteln

```
  For Each cbc In cb.Controls
    Debug.Print cbc.TooltipText & " --> " & cbc.Id
  Next cbc
End Sub
```

Listing 553: Die eindeutigen IDs von Symbolen ermitteln (Forts.)

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 die Symbolschaltfläche anklicken.

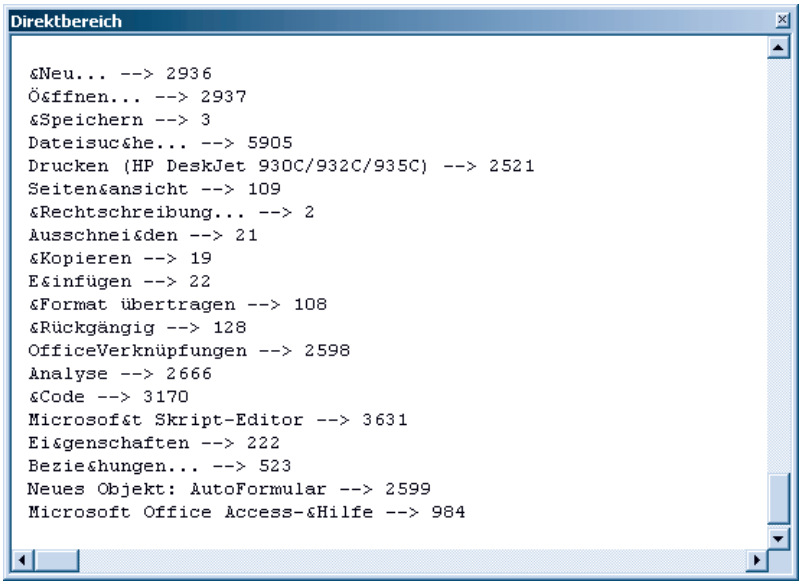


Abbildung 375: Die IDs der Symbole wurden dokumentiert.

428 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 eine eigene Prozedur zuweisen.

In der folgenden Prozedur aus *Listing 554* wird die Symbolleiste NEUE SYMBOLLEISTE angelegt und die gerade beschriebenen Funktionen integriert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub SymbolleisteEinfügenUndFüllen()
  Dim cb As CommandBar
  Dim cbc As CommandBarControl

  On Error GoTo Fehler
  Set cb = CommandBars.Add("Neue Symbolleiste")

  'Speichern-Symbol einfügen
  Set cbc = cb.Controls.Add(msoControlButton, 3)

  'Symbol Neu einfügen
  Set cbc = cb.Controls.Add(msoControlButton, 2936)

  'Symbol Öffnen einfügen
  Set cbc = cb.Controls.Add(msoControlButton, 2937)

  'Benutzerdefiniertes Symbol einfügen
  Set cbc = cb.Controls.Add(msoControlButton)
  With cbc
    .FaceId = 125
    .TooltipText = "Aktuelles Datum"
    .OnAction = "DatumAusgeben"
  End With

  Set cb = Nothing
  Set cbc = Nothing
  Exit Sub

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

Listing 554: Eine neue Symbolleiste anlegen und mit Symbolen bestücken

Geben Sie in der Anweisung `Set` im Zusammenspiel mit der Methode `Add` an, in welche Symbolleiste Sie die Symbole einfügen möchten, um die neue Symbolleiste anzulegen. Danach fügen Sie mit der Methode `Add` die einzelnen Symbole ein. Wenn Sie dabei auf bereits fertige Funktionen von Access zurückgreifen möchten, genügt es, wenn Sie bei der Methode `Add` die dazugehörige `ID` mit angeben, die Sie vorher bereits über die Prozedur aus *Listing 553* ermittelt haben. Damit wird das Aussehen sowie die dazugehörige QuickInfo, die die Beschreibung der Funktion enthält, automatisch mit angelegt.

Abbildung 376: 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, welche Prozedur beim Klick auf das Symbol ausgeführt werden soll. Sie müssen zu diesem Zweck noch die Prozedur erfassen. Diese Prozedur sehen Sie in *Listing 555*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub DatumAusgeben()
  MsgBox "Heute ist der " & Date
End Sub
```

Listing 555: Das aktuelle Datum ausgeben

429 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.

In der nächsten Prozedur aus *Listing 556* wird die Symbolleiste DATENBANK geschützt.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================
```

Listing 556: Symbolleisten schützen

```
Sub SymbolleisteSchützen()
  With Application.CommandBars("Database")
    .Protection = msoBarNoChangeVisible + msoBarNoCustomize
    .Visible = True
  End With
End Sub
```

Listing 556: Symbolleisten schützen (Forts.)

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 der *Tabelle 158* 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`	Kein Hinzufügen bzw. Löschen von Symbolen aus der Symbolleiste. Auch das Verschieben der Symbole ist nicht möglich.
`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 auf dem Bildschirm nicht frei bewegt werden.
`msoBarNoResize`	Die Symbolleiste kann in ihrer Form nicht verändert werden.

Tabelle 158: Die Möglichkeiten zum Schutz von Symbolleisten

Heben Sie den Schutz wieder auf, indem Sie die folgende Prozedur aus *Listing 557* starten.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================
```

Listing 557: Symbolleistenschutz wieder aufheben

```
Sub SymbolleistenschutzAufheben()
  With Application.CommandBars("Database")
    .Protection = False
    .Visible = True
  End With
End Sub
```

Listing 557: Symbolleistenschutz wieder aufheben (Forts.)

Setzen Sie die Eigenschaft `Protection` auf den Wert `False`, um den Schutz der Symbolleiste wieder aufzuheben.

430 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 sie 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 die Prozedur aus *Listing 558*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub DeaktivierenSymbole()
  Dim cb As CommandBar

  Set cb = CommandBars("Neue Symbolleiste")
  cb.Visible = True
  With cb
    .Controls(1).Enabled = False
    .Controls(2).Enabled = False
  End With
End Sub
```

Listing 558: 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 deaktivie-

ren möchten, und deaktivieren es schließlich, indem Sie die Eigenschaft `Enabled` auf den Wert `False` setzen.

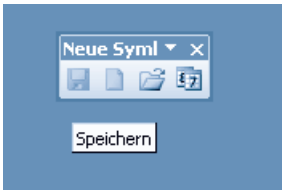


Abbildung 377: Die ersten beiden Symbole wurden deaktiviert.

Um diese beiden Symbole wieder zu aktivieren, setzen Sie die Eigenschaft `Enabled` für diese Symbole auf `True`.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub AktivierenSymbole()
  Dim cb As CommandBar

  Set cb = CommandBars("Neue Symbolleiste")
  cb.Visible = True
  With cb
    .Controls(1).Enabled = True
    .Controls(2).Enabled = True
  End With
End Sub
```

Listing 559: Symbole wieder aktivieren

431 Adaptive Menüs ausschalten

Bei der Ermittlung der korrekten Einfügeposition kann Ihnen aber die seit der Access-Version 2000 bestehende Funktion der 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 in den Menüs immer weiter 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 die Prozedur aus *Listing 560* vornehmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub AdaptiveMenüsAusschalten()
  Application.CommandBars.AdaptiveMenus = False
End Sub
```

Listing 560: Adaptive Menüs ausschalten

Setzen Sie die Eigenschaft `AdaptiveMenus` auf den Wert `False`, um die personalisierten Menüs abzuschalten. Dadurch werden die Menüs in Access wie gewohnt 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 von Office zur Verfügung.

432 Neues Menü einfügen

In der nächsten Aufgabe soll genau vor dem HILFE-Menü ein zusätzliches 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 561* entnehmen.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub NeuesMenüEinfügen()
  Dim intz As Integer
  Dim intPosHilfe As Integer
  Dim cbc As CommandBarControl
```

Listing 561: Neues Menü einfügen

```
  On Error GoTo Fehler
  intz = Application.CommandBars("Menu Bar").Controls.Count
  intPosHilfe = Application.CommandBars("Menu Bar").Controls(intz).Index

  Set cbc = Application.CommandBars("Menu Bar"). _
  Controls.Add(Type:=msoControlPopup, Before:=intPosHilfe, Temporary:=True)
  cbc.Caption = "&Eigene Funktionen"

  Set cbc = Nothing
  Exit Sub

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

Listing 561: Neues Menü einfügen (Forts.)

Definieren Sie im ersten Schritt zwei `Integer`-Variable, die zum einen die Anzahl der Menüs, 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 dabei 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 Drop-down-Feld ein.
`msoControlComboBox`	Fügt ebenfalls ein Drop-down-Feld ein.
`msoControlPopup`	Fügt ein Drop-down-Menü ein.

Tabelle 159: Alle möglichen Konstanten für Steuerelemente in Menüleisten

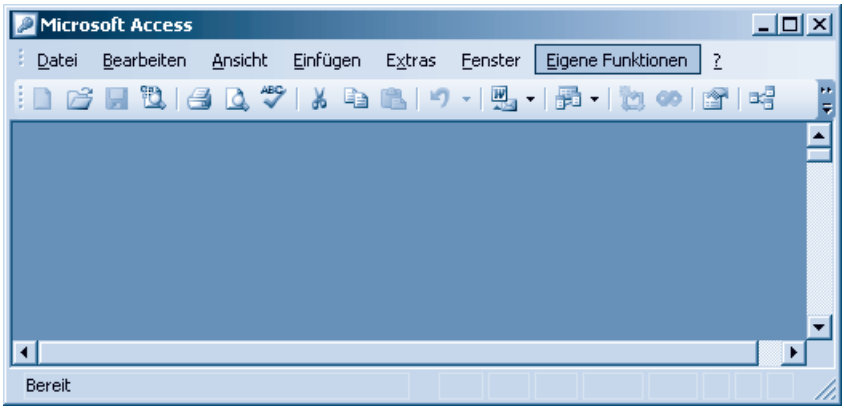


Abbildung 378: Das neue Menü wurde eingefügt.

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.

433 Menüleiste zurücksetzen

Haben Sie ein wenig mit Ihrer Menüleiste herumexperimentiert und möchten Sie diese ganz schnell wieder auf ihren ursprünglichen Zustand zurücksetzen, dann starten Sie die Prozedur aus *Listing 562*.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Sub MenüLeisteZurücksetzen()
  On Error Resume Next
  Application.CommandBars("Menu Bar").Reset
End Sub
```

Listing 562: Menüleiste zurücksetzen

Die Methode `Reset` setzt die angegebene integrierte Befehlsleiste auf die Standardkonfiguration der Steuerelemente zurück.

434 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 ebenfalls weitere Menüs bzw. Menüunterpunkte in die Menüleiste von Access. Im 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 die Prozedur aus *Listing 563* an.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub MenüLöschen()
  On Error Resume Next
  Application.CommandBars("Menu Bar").Controls("Eigene Funktionen").Delete
End Sub
```

Listing 563: Menü löschen

Wenden Sie die Methode `Delete` an, um das neu eingefügte Menü wieder zu löschen.

435 Menübefehle einfügen

Das gerade eingefügte Menü enthält noch keine Menübefehle. Diese fügen Sie jetzt ein. Erweitern Sie die vorher bereits erfasste Prozedur aus *Listing 561* wie folgt:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub NeuesMenüEinfügen2()
  Dim intZ As Integer
  Dim intPosHilfe As Integer
  Dim cbc As CommandBarControl
  Dim cbc2 As CommandBarControl

  On Error GoTo Fehler
  intZ = Application.CommandBars("Menu Bar").Controls.Count
  intPosHilfe = Application.CommandBars("Menu Bar").Controls(intZ).Index
```

Listing 564: Neues Menü und Menübefehle einfügen

```
  Set cbc = Application.CommandBars("Menu Bar"). _
  Controls.Add(Type:=msoControlPopup, Before:=intPosHilfe, Temporary:=True)
  cbc.Caption = "&Eigene Funktionen"

  Set cbc2 = cbc.Controls.Add(Type:=msoControlButton)

  With cbc2
    .Caption = "Speicherort der Datei anzeigen"
    .Style = msoButtonIconAndCaption
    .OnAction = "Verzeichniszurückgeben"
    .FaceId = 23
  End With

  Set cbc2 = cbc.Controls.Add(Type:=msoControlButton)

  With cbc2
    .Caption = "Wo&chentag, Datum und Uhrzeit anzeigen"
    .Style = msoButtonCaption
    .OnAction = "DatumUndUhrzeit"
    .BeginGroup = True
  End With

  Set cbc2 = Nothing
  Set cbc = Nothing
  Exit Sub

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

Listing 564: 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 festzulegen. Ist das Menü einmal aktiviert, können Sie durch die Eingabe des unterstrichenen Buchstabens innerhalb des Menübefehls die dahinter liegende Funktion bzw. Prozedur starten.

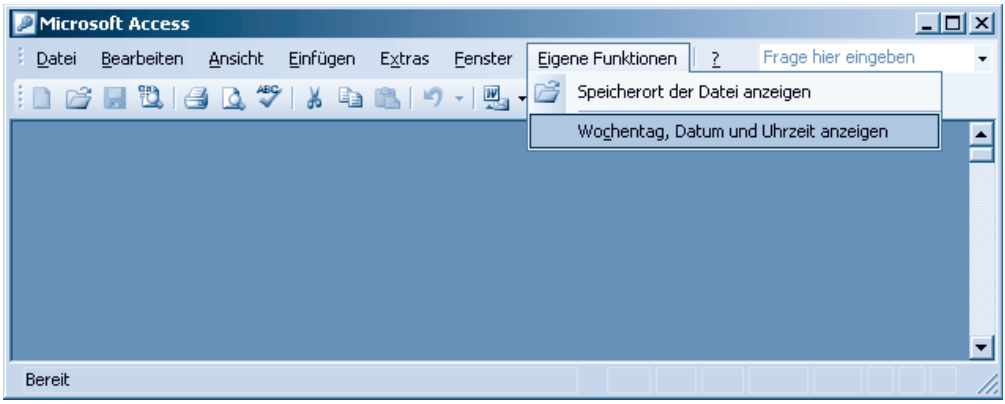


Abbildung 379: Das neue Menü wurde angelegt und mit Menübefehlen ausgestattet.

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.

436 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.

437 Menü auslesen

Um speziell die Menüleiste von Access auszulesen, setzen Sie den Code aus *Listing 565* ein.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub MenüNamenAusgeben()
  Dim cb As CommandBar
  Dim cbc As CommandBarControl

  Set cb = Application.CommandBars("Menu Bar")
```

Listing 565: Menüs auslesen

```
  For Each cbc In cb.Controls
    Debug.Print cbc.Caption
  Next
End Sub
```

Listing 565: Menüs auslesen (Forts.)

Deklarieren Sie zuerst zwei Objektvariable. Eine davon vom Typ `CommandBar` und die zweite vom Typ `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.

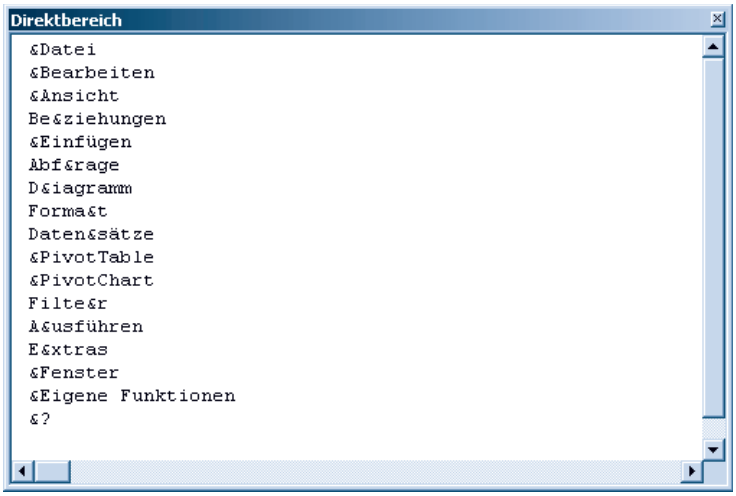


Abbildung 380: Die Menünamen wurden dokumentiert.

438 Menübefehle auslesen

Möchten Sie mehr über ein bestimmtes Menü erfahren, hilft Ihnen die Prozedur aus *Listing 566*. Sie schreibt alle Menübefehle des ersten Menüs in den Direktbereich.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub MenüsAuslesen()
  Dim cbc As CommandBarControl
  Dim cbc2 As CommandBarControl
```

Listing 566: Menübefehle eines Menüs auslesen

```
  Set cbc = Application.CommandBars("Menu Bar").Controls(1)

  For Each cbc2 In cbc.Controls
    Debug.Print cbc2.Caption
  Next cbc2
End Sub
```

Listing 566: Menübefehle eines Menüs auslesen (Forts.)

Indem Sie die Eigenschaft `Controls` auf den Wert `1` setzen, bestimmen Sie, dass das Menü DATEI gemeint ist.

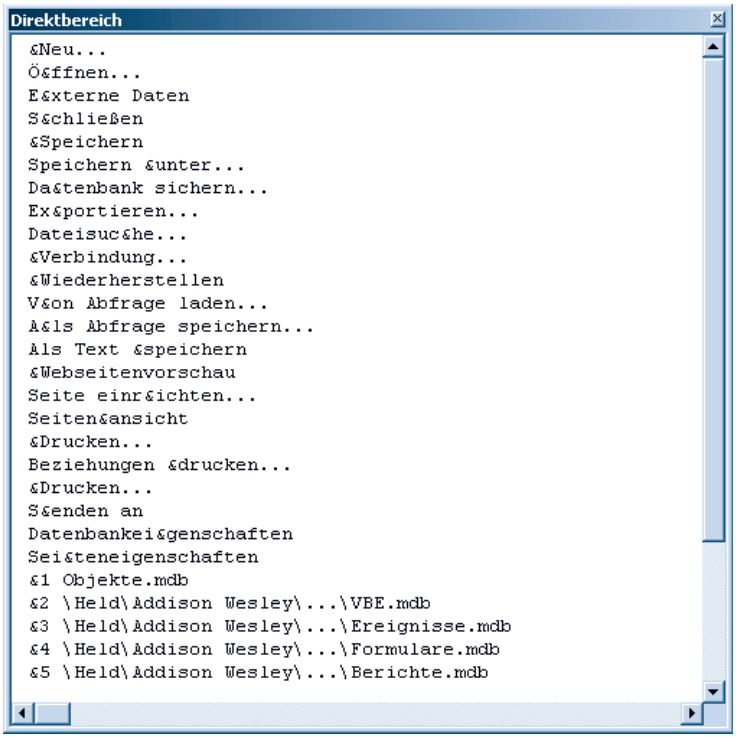


Abbildung 381: Alle Befehle des ersten Menüs wurden dokumentiert.

439 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 567* werden alle Menübefehle im Menü EINFÜGEN deaktiviert.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub AlleMenübefehleInLeisteDeaktivieren()
  Dim cbc As CommandBarControl
  Dim cbc2 As CommandBarControl

  Set cbc = Application.CommandBars("Menu Bar").Controls(5)

  For Each cbc2 In cbc.Controls
    cbc2.Enabled = False
  Next cbc2
End Sub
```

Listing 567: Alle Menübefehle deaktivieren

Setzen Sie die Eigenschaft `Enabled` auf den Wert `False`, um die einzelnen Menübefehle zu deaktivieren.

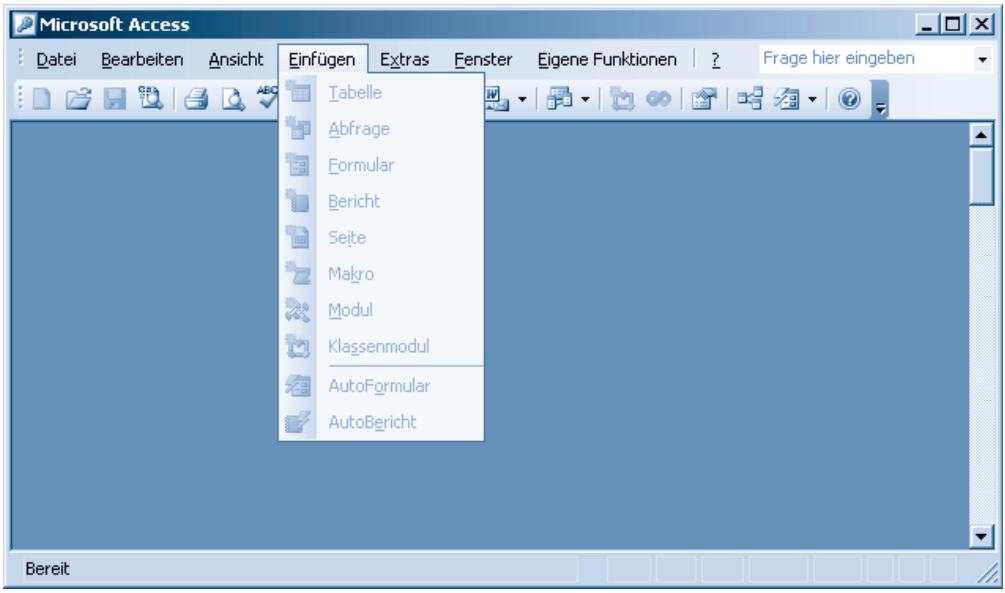


Abbildung 382: Alle Menübefehle im Menü Einfügen wurden deaktiviert.

Um die Menübefehle wieder zu aktivieren, starten Sie die Prozedur aus *Listing 568*.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub AlleMenübefehleInLeisteAktivieren()
  Dim cbc As CommandBarControl
  Dim cbc2 As CommandBarControl

  Set cbc = Application.CommandBars("Menu Bar").Controls(5)

  For Each cbc2 In cbc.Controls
    cbc2.Enabled = True
  Next cbc2
End Sub
```

Listing 568: Alle Menübefehle eines Menüs wieder aktivieren

440 Menübefehle (de)aktivieren

Natürlich können Sie auch gleich ganz oben im Menü den Menüpunkt deaktivieren. Dadurch kann das entsprechende Menü nicht mehr heruntergeklappt werden. Der Code für diese Anweisung entnehmen Sie dem *Listing 569*:

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Sub MenüDeaktivieren()
  'Menü Einfügen deaktivieren
  Application.CommandBars("Menu Bar").Controls(5).Enabled = False
End Sub
```

Listing 569: Ganzes Menü deaktivieren

Setzen Sie die Eigenschaft `Enabled` auf den Wert `True`, um das Menü wieder zur Verfügung zu stellen.

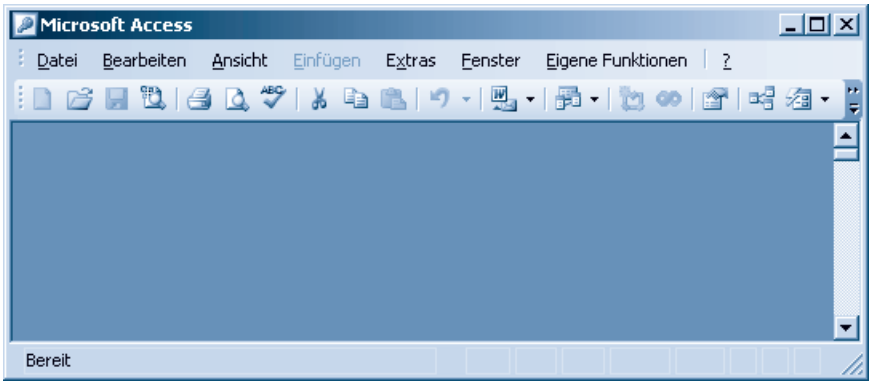


Abbildung 383: Ein ganzes Menü wurde deaktiviert.

Die Multifunktionsleiste programmieren

Neu ab Access 2007 ist die sogenannte Multifunktionsleiste, im Englischen THE RIBBON genannt. Die Multifunktionsleiste kann nicht direkt über VBA programmiert werden, sondern nur über den Umweg von XML-Markups. VBA wird hier lediglich zum Einsatz gebracht, um die auf XML basierenden Konfigurationen bei Bedarf dynamisch, also zur Laufzeit, in die Datenbank einzuladen bzw. um auch auf Aktionen wie beispielsweise Mausklicks auf Schaltflächen zu reagieren. Diese neue Technik ist nur für Access 2007 einsetzbar.

Auf den folgenden Seiten wird beschrieben, wie Sie ein XML-Markup in eine Datenbank bekommen. In diesem XML-Markup steht alles, was das Aussehen der Multifunktionsleiste ausmacht. Das XML-Markup beschreibt also ganz genau, wie die neue Oberfläche in Access 2007 aussehen soll.

Folgende Arbeitsschritte sind notwendig, um eine neue Menüleiste in Access einzubinden:

- Erstellung eines XML-Markups nach einem bestimmten Muster.
- Anlage einer Access-Tabelle mit bestimmten Feldinhalten.
- Erfassen des XML-Markups in dieser Tabelle.
- Einlesen dieser Daten in Access, mit oder ohne VBA.
- Einstellen der gewünschten Menüleiste in den Access-Optionen.
- Speichern und Neustart der Datenbank.

441 Programmierung über ein XML-Markup

Am folgenden Beispiel soll demonstriert werden, wie Sie eine zusätzliche Registerseite mit dem Namen MEINE REGISTERSEITE anlegen und mit einer Schaltfläche belegen können. Der Rahmen für ein solches Markup sieht wie in *Abbildung 384* dargestellt aus.

Die Erstellung eines XML-Markups kann mit einem normalen Text-Editor erfolgen, wenn Sie keinen XML-Editor zur Verfügung haben. Wenn Sie eine XML-Datei nur ansehen möchten, können Sie dazu den Microsoft Internet Explorer verwenden. Dieser zeigt Ihnen eine XML-Datei übersichtlich formatiert an.

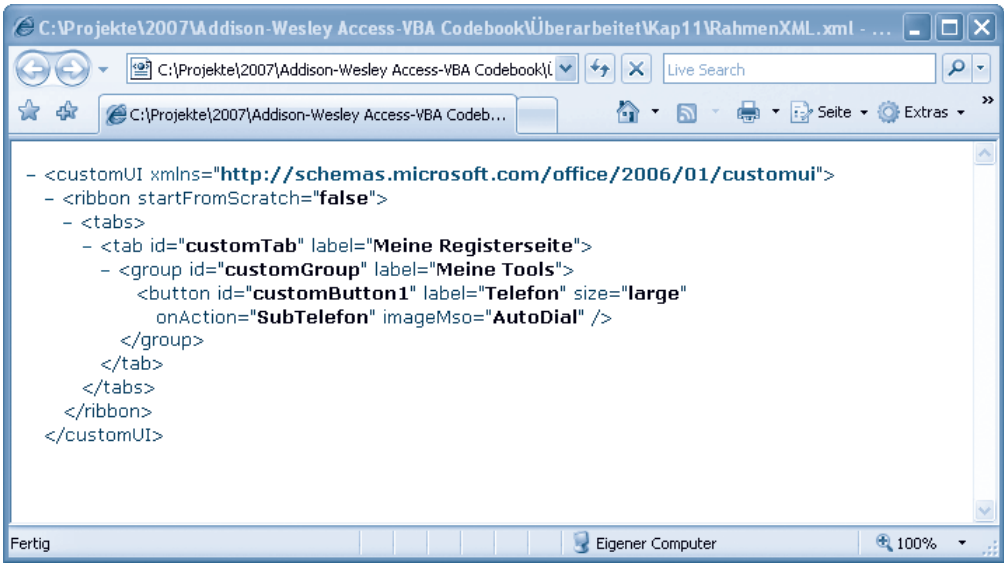


Abbildung 384: Der Grundrahmen des XML-Markups

Ein XML-Markup für die Menüleisten-Programmierung hat als oberstes Element ein `customUI`-Element und beginnt mit der Zeile

```
<customUI xmlns=
"http://schemas.microsoft.com/office/2006/01/customui">
```

Diese Zeile definiert, dass das XML-Markup eine Beschreibung für das Microsoft Office Custom User Interface enthält und die folgenden XML-Tags zum Namensraum desselben gehören.

Das darauf folgende Element sieht wie folgt aus:

```
<ribbon startFromScratch="true">
```

Die Beschreibungen für die Multifunktionsleiste befinden sich alle unterhalb dieses `ribbon`-Elements. Wenn Sie das Attribut `startFromScratch` wie in der Abbildung gezeigt auf den Wert `False` setzen, dann werden die vier Standard-Registerseiten

START, ERSTELLEN, EXTERNE DATEN und DATENBANKTOOLS in Access angezeigt. Soll dies nicht geschehen, dann setzen Sie dieses Attribut auf den Wert `True`.

Über XML-Elemente mit den Namen `Tab` definieren Sie Registerseiten. In unserem Beispiel definieren Sie genau eine Registerseite.

```
<tab id="customTab" label="Meine Registerseite">
```

Über das Attribut `id="customTab"` geben Sie an, dass es sich um eine anwenderspezifische Registerseite handelt, über das `Attribut label="Meine Registerseite"` geben Sie an, mit welchem Namen die Registerseite in der Multifunktionsleiste angezeigt werden soll. Das im `Tab`-Element enthaltene nächste Element lautet:

```
<group id="customGroup" label="Meine Tools">
```

Damit definieren Sie auf der Registerseite eine Gruppe die den Namen `"Meine Tools"` tragen wird. Das folgende Element

```
<button id="customButton1" label="Telefon" size="large"
onAction="SubTelefon" imageMso="AutoDial" />
```

definiert eine Schaltfläche. Über die Attributzuweisung `label="Telefon"` bestimmen Sie, dass die Schaltfläche den Namen TELEFON tragen wird, über `size="large"`, dass ein großes Symbol angezeigt wird und über `imageMso="AutoDial"` welches Bild die Schaltfläche tragen soll. Ganz wichtig ist die Zuweisung `onAction="SubTelefon"`. Damit wird festgelegt, dass bei Betätigung der Schaltfläche die VBA-Prozedur `SubTelefon` aufgerufen wird.

Speichern Sie dieses XML-Markup unter dem Namen RAHMENXML.XLM.

442 Anlage der Tabelle USysRibbons

In diesem Abschnitt wird gezeigt, wie das soeben erstellte XML-Markup über einen automatischen Mechanismus in Access geladen werden kann, um die von Ihnen getätigten Konfigurationen umzusetzen. In einem folgenden Abschnitt wird gezeigt, wie Sie dieses Laden auch über VBA-Code erledigen können, um so mehr Flexibilität zu erhalten.

Wenn Sie das Laden des XML-Markups von Access, ohne den Einsatz einer VBA-Prozedur erledigen wollen, starten Sie Microsoft Access 2007 und legen eine leere Tabelle mit dem Namen USYSRIBBONS an. Orientieren Sie sich an dem folgenden Aufbau. Achten Sie auf die Bezeichnungen der Tabellenfelder, inklusive der Groß- und Kleinschreibung.

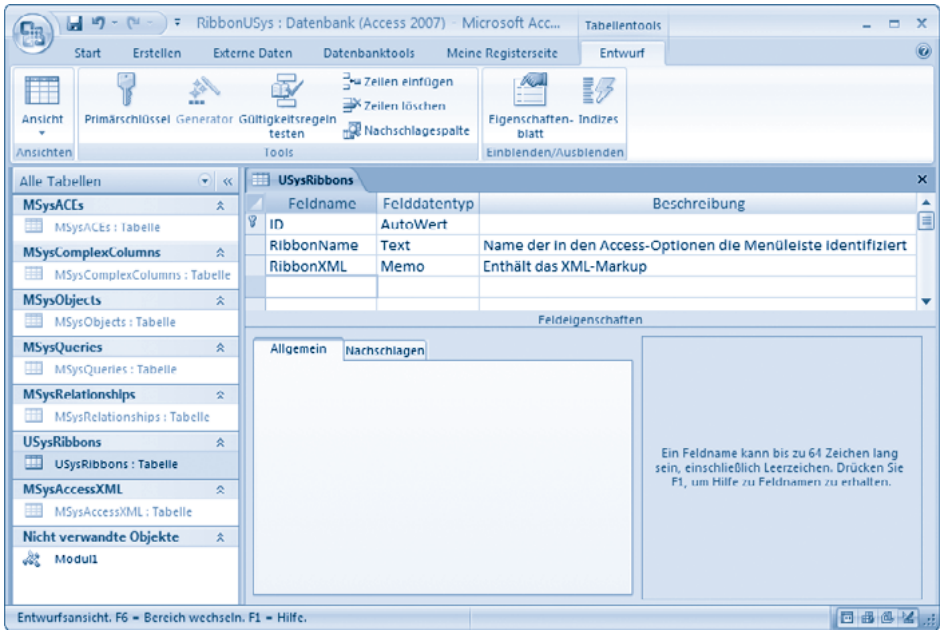


Abbildung 385: Die Tabelle USysRibbons

1. Öffnen Sie die vorher abgespeicherte XML-Datei beispielsweise über den Text-Editor und kopieren den Inhalt der Datei in die Zwischenablage.
2. Öffnen Sie die Tabelle USYSRIBBONS und legen den ersten Satz wie folgt an.

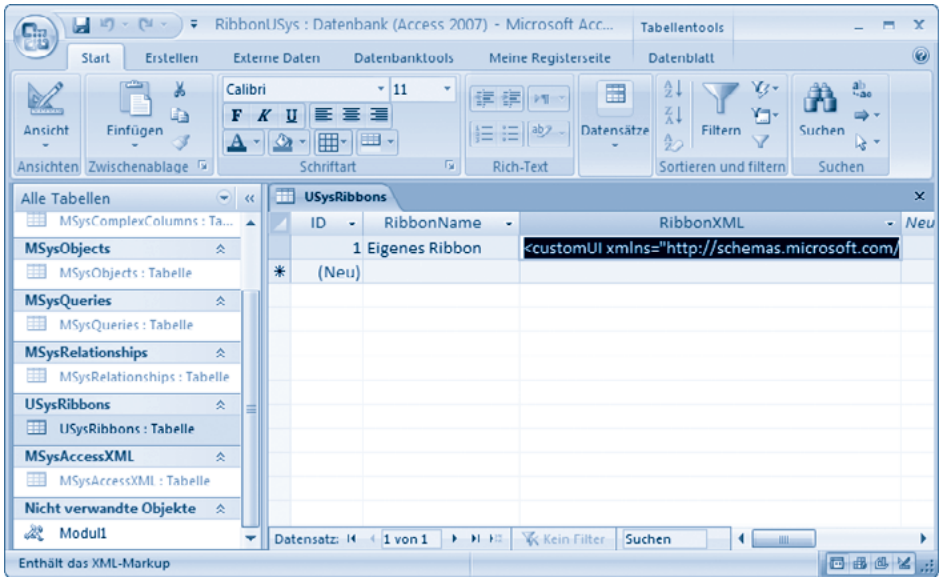


Abbildung 386: Das XML-Markup ins Feld RibbonXML einfügen

3. Vergeben Sie im Feld RIBBONNAME einen Namen nach eigener Wahl, z. B. EIGENES RIBBON.
4. Setzen Sie den Mauszeiger in das Feld RIBBONXML und drücken Sie die Tastenkombination Strg + V, um das Skript im Memofeld einzufügen.
5. Speichern Sie die Tabelle.

Hinweis

Bitte beachten Sie, dass die Tabelle USYSRIBBONS von Access automatisch versteckt wird. Um diese Datei in der Navigationsleiste anzuzeigen, müssen Sie folgendermaßen vorgehen:

Klicken Sie mit der rechten Maustaste auf die obere Leiste der Navigationsleiste, und wählen Sie im angezeigten Kontextmenü den Menüpunkt NAVIGATIONSOPTIONEN. Im Dialog NAVIGATIONSOPTIONEN aktivieren Sie dann die Option SYSTEMOBJEKTE ANZEIGEN.

443 Ereignis-Prozedur erstellen

Sie erinnern sich: Sie hatten innerhalb Ihres XML-Markups im Element-Tag `button` über die Attributzuweisung `onAction = "SubTelefon"` die Prozedur SUBTELEFON als Ereignisprozedur für den Schaltflächenklick definiert. Diese Prozedur müssen Sie noch erstellen. Erstellen Sie daher im VBA-Editor in einem Modul folgende Prozedur. Beachten Sie, dass Sie als Argument die Objektvariable `IRibbonControl` mitgeben.

```
'=====================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=====================================================

Public Sub SubTelefon(control As IRibbonControl)
  MsgBox "Prozedur 'SubTelefon' aufgerufen!"
End Sub
```

Listing 570: Die Ereignisprozedur für den Schaltflächenklick

444 Neue Menüleiste auswählen

Als letzten Schritt müssen Sie nun nur noch in den Access-Optionen angeben, dass die von Ihnen definierte Registerseite verwendet werden soll.

1. Wenn Sie seit Erstellung oder Modifikation der Tabelle USYSRIBBON die Datenbank noch nicht neu gestartet haben, dann tun Sie dies jetzt.
2. Klicken Sie anschließend auf die runde Office-Schaltfläche am linken, oberen Rand.
3. Klicken Sie auf die Schaltfläche ACCESS-OPTIONEN.
4. Aktivieren Sie die Rubrik AKTUELLE DATENBANK.

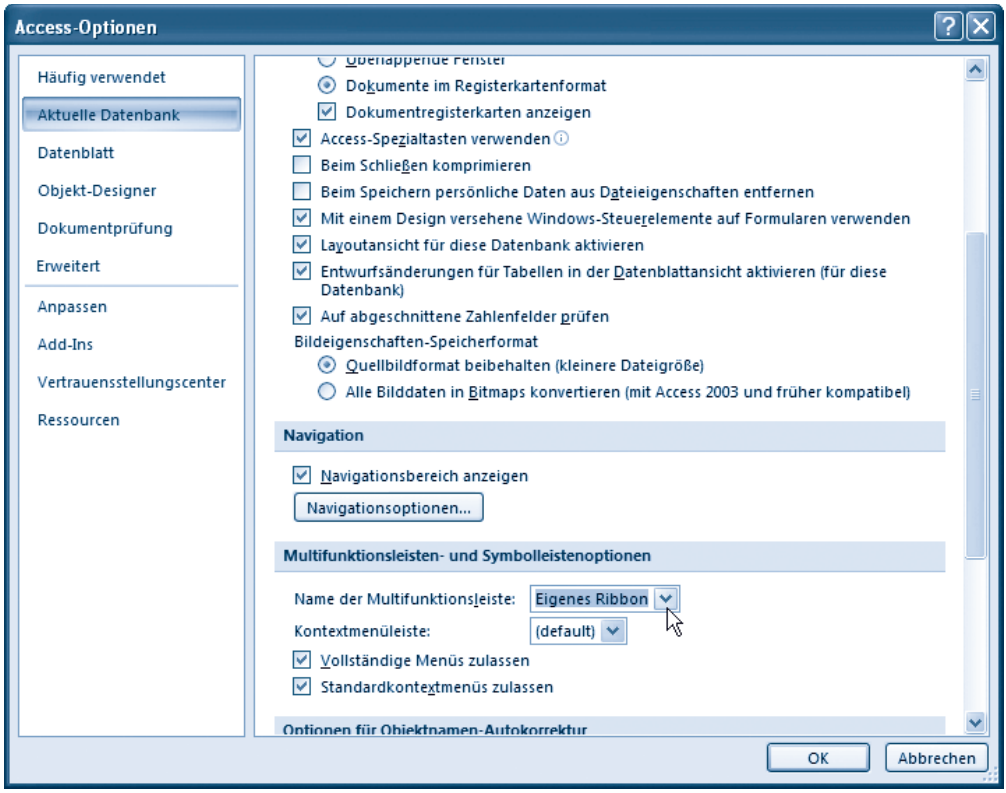


Abbildung 387: Die gewünschte Multifunktionsleiste angeben

5. In der Gruppe MULTIFUNKTIONSLEISTEN- UND SYMBOLLEISTENOPTIONEN wählen Sie aus dem Drop-down NAME DER FUNKTIONSLEISTE die nun angebotene Leiste EIGENES RIBBON.
6. Bestätigen Sie mit OK.

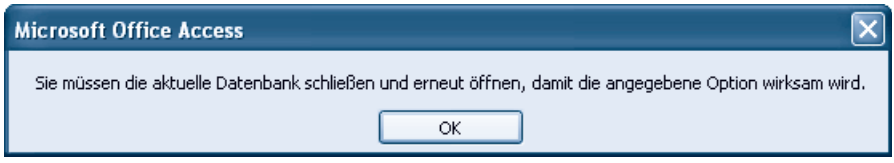


Abbildung 388: Bestätigung der Meldung

7. Bestätigen Sie die Rückfrage mit OK.
8. Schließen Sie die Datenbank und öffnen diese erneut.

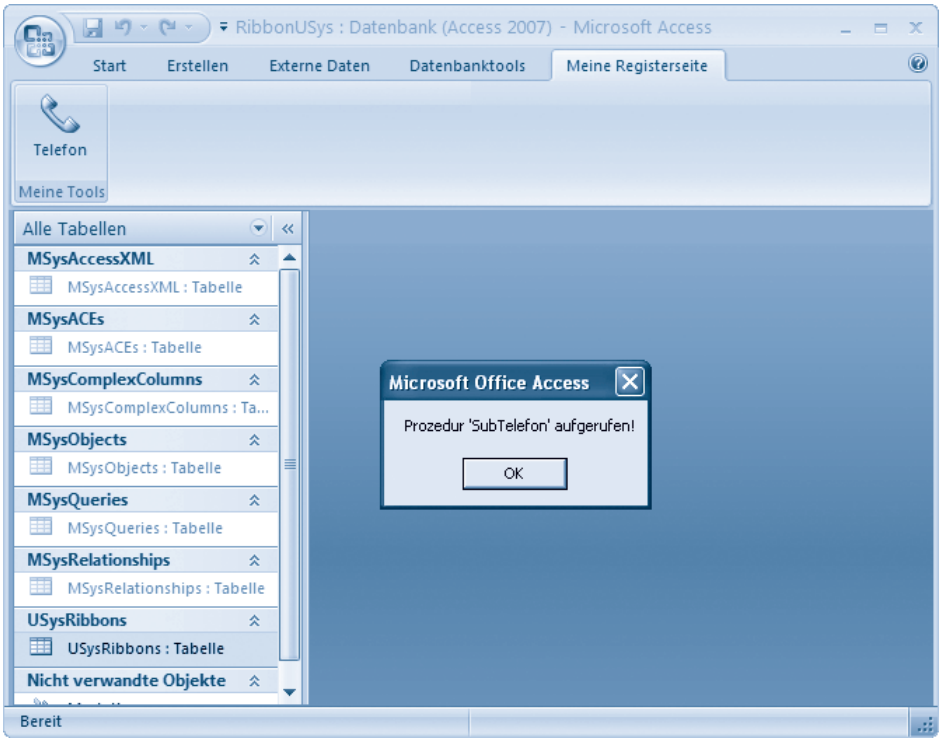


Abbildung 389: Die neue Registerseite mit funktionierender Schaltfläche.

Nach dem erneuten Start der Datenbank wird die neue Registerseite automatisch angezeigt. Ein Klick auf die Schaltfläche TELEFON in der Gruppe MEINE TOOLS ruft die Prozedur SUBTELEFON auf.

445 Dynamisches Laden ohne USysRibbon

Als Alternative zum automatischen Laden über die Tabelle USYSRIBBON können Sie dieses Laden auch über VBA-Code erledigen. Damit haben Sie die Möglichkeit, zur Laufzeit über unterschiedliche XML-Markups verschiedene Konfigurationen laden zu können. Sie können das XML-Markup dann in einer beliebigen Tabelle halten, in einer externen XML-Datei oder direkt im VBA-Code als definierte String-Konstante. Wählen Sie für dieses Beispiel die Variante 1 und benennen Sie die Tabelle USYSRIBBON um in MEINERIBBONS.

Um die Tabelle dynamisch über VBA-Code laden zu können, wird noch eine bestimmte Bibliothek benötigt. Wählen Sie hierzu in der Entwicklungsumgebung von Access aus dem Menü EXTRAS den Befehl VERWEISE. Aktivieren Sie dort die Bibliothek MICROSOFT OFFICE 12.0 OBJECT LIBRARY und bestätigen Sie mit OK.

Erfassen Sie nun die folgende Funktion in einem Standardmodul.

```
'=======================================================
' Auf CD     Buchdaten\Beispiele
' Dateiname  Leisten.mdb
' Modul      MdlCmd
'=======================================================

Function LoadRibbons()
  Dim i As Integer
  Dim db As DAO.Database
  Dim rst As DAO.Recordset

  On Error GoTo Fehlerbehandlung
  Set db = Application.CurrentDb

  For i = 0 To (db.TableDefs.Count - 1)
    If db.TableDefs(i).Name = "MeineRibbons" Then

      Set rst = CurrentDb.OpenRecordset _
        (db.TableDefs(i).Name)
      rst.MoveFirst

      While Not rst.EOF
        Application.LoadCustomUI rst("RibbonName") _
          .Value, rst("RibbonXML").Value
        rst.MoveNext
      Wend

      rst.Close
      Set rst = Nothing
    End If
  Next i

  db.Close
  Set db = Nothing
  Exit Function

Fehlerbehandlung:
  Select Case Err.Number
    Case 32609
      MsgBox "Das Ribbon wurde schon geladen!", vbInformation
    Case Else
      MsgBox Err.Number & " " & Err.Description
  End Select

End Function
```

Listing 571: Menüleistenkonfiguration dynamisch laden

Setzen Sie die Methode `LoadCustomUI` ein, um die Multifunktionsleisten (es könnten auch mehrere sein) in Access bekannt zu machen. Dabei übergeben Sie den Namen des Ribbons sowie das XML-Markup aus den Feldern RIBBONNAME und RIBBONXML an die Datenbank.

In diesem Beispiel soll der Einfachheit halber nun die Registerseite auch gleich beim Programmstart geladen werden. Erstellen Sie daher ein Makro mit dem Namen AUTOEXEC und rufen Sie über dieses die Funktion LOADRIBBONS auf.

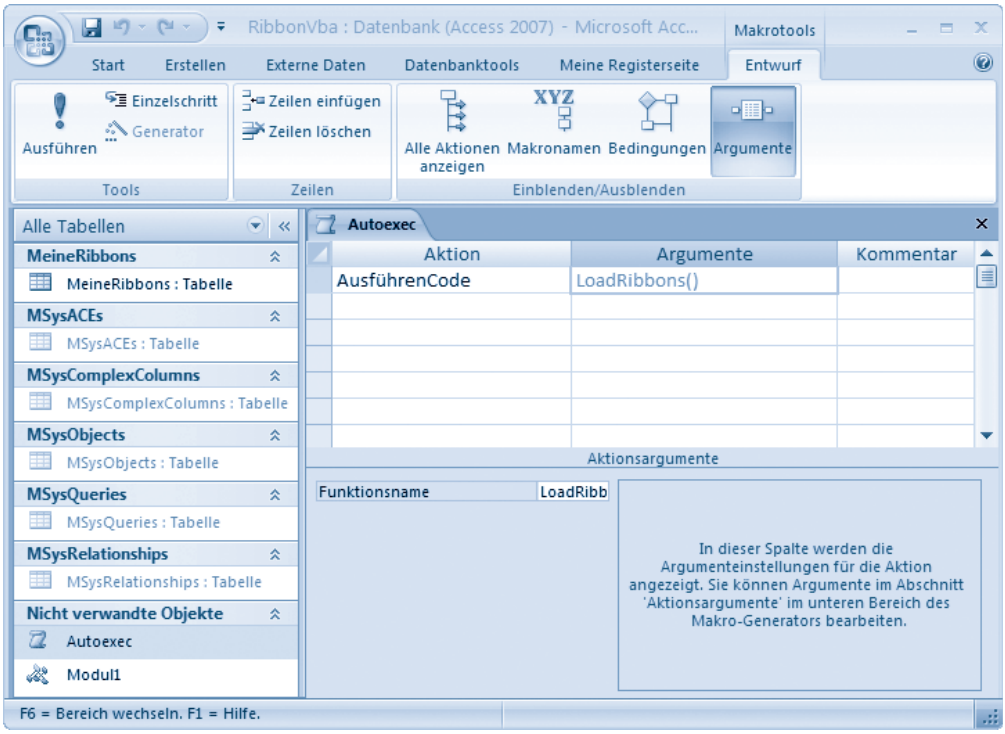
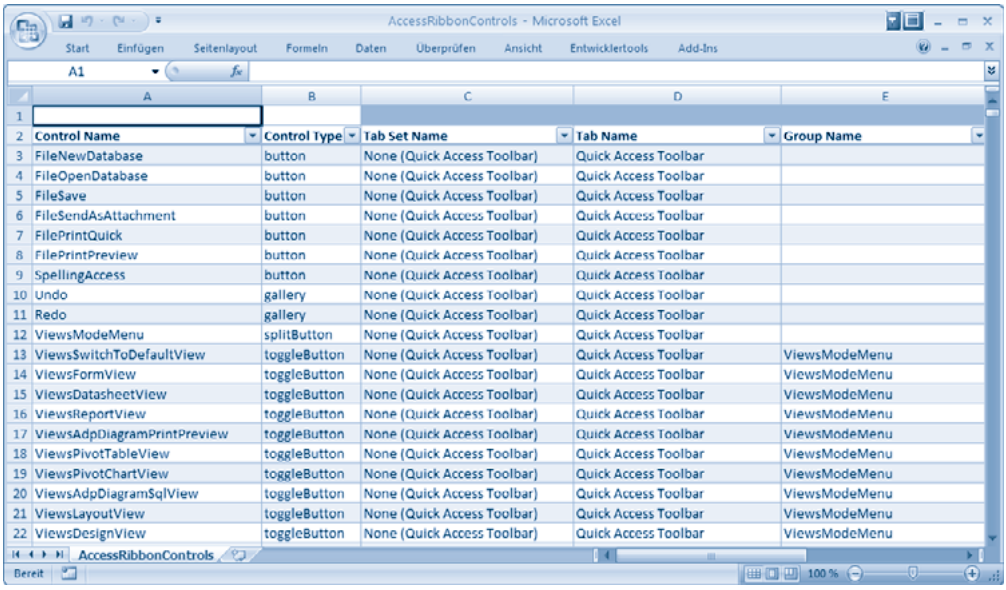


Abbildung 390: Die Funktion LoadRibbons beim Programmstart über ein Makro aufrufen

Weitere wichtige Quellen und Hilfen

Damit Sie genau wissen, wie die Elemente für die Ribbons heißen, können Sie sich eine Dokumentation (2007OFFICECONTROLIDSEXCEL2007.EXE) mit den Namen der Elemente, die Sie in die Ribbons einfügen möchten, von der Microsoft-Homepage herunterladen.

In der Datei ACCESSRIBBONCONTROLS.XLSX finden Sie dann eine Auflistung aller Elemente.



AccessRibbonControls - Microsoft Excel

Start | Einfügen | Seitenlayout | Formeln | Daten | Überprüfen | Ansicht | Entwicklertools | Add-Ins

A1

	A	B	C	D	E
1					
2	Control Name	Control Type	Tab Set Name	Tab Name	Group Name
3	FileNewDatabase	button	None (Quick Access Toolbar)	Quick Access Toolbar	
4	FileOpenDatabase	button	None (Quick Access Toolbar)	Quick Access Toolbar	
5	FileSave	button	None (Quick Access Toolbar)	Quick Access Toolbar	
6	FileSendAsAttachment	button	None (Quick Access Toolbar)	Quick Access Toolbar	
7	FilePrintQuick	button	None (Quick Access Toolbar)	Quick Access Toolbar	
8	FilePrintPreview	button	None (Quick Access Toolbar)	Quick Access Toolbar	
9	SpellingAccess	button	None (Quick Access Toolbar)	Quick Access Toolbar	
10	Undo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar	
11	Redo	gallery	None (Quick Access Toolbar)	Quick Access Toolbar	
12	ViewsModeMenu	splitButton	None (Quick Access Toolbar)	Quick Access Toolbar	
13	ViewsSwitchToDefaultView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
14	ViewsFormView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
15	ViewsDatasheetView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
16	ViewsReportView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
17	ViewsAdpDiagramPrintPreview	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
18	ViewsPivotTableView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
19	ViewsPivotChartView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
20	ViewsAdpDiagramSqlView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
21	ViewsLayoutView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu
22	ViewsDesignView	toggleButton	None (Quick Access Toolbar)	Quick Access Toolbar	ViewsModeMenu

AccessRibbonControls

Bereit

100 %

Abbildung 391: Die Auflistung aller verfügbaren Ribbon-Controls

Eine weitere Informationsquelle bietet die Datei OFFICE2007ICONSGALLERY.EXE, die Sie ebenfalls bei Microsoft herunterladen können. Sie enthält die eindeutigen Bezeichnungen für Bildsymbole. Anders als in den Vorversionen wurden als Bezeichnungen keine eindeutigen Nummern, sondern sprechende Namen wie FILTER AUTODIAL, HAPPYFACE usw. verwendet. Nach dem Download entpacken Sie die Datei und installieren die dahintersteckende Excel-Arbeitsmappe. Öffnen Sie die Datei OFFICE2007ICONSGALLERY.XLSM in Excel.

1. Starten Sie Microsoft Excel.
2. Klicken Sie die Schaltfläche OFFICE.
3. Klicken Sie danach auf die Schaltfläche EXCEL-OPTIONEN.
4. Aktivieren Sie die Rubrik HÄUFIG VERWENDET.
5. Aktivieren Sie das Kontrollkästchen ENTWICKLERREGISTERKARTE IN DER MULTIFUNKTIONSLEISTE ANZEIGEN.
6. Bestätigen Sie mit OK.
7. Wechseln Sie auf das Ribbon ENTWICKLERTOOLS und klappen exemplarisch das Drop-down GALLERY 1 herunter.

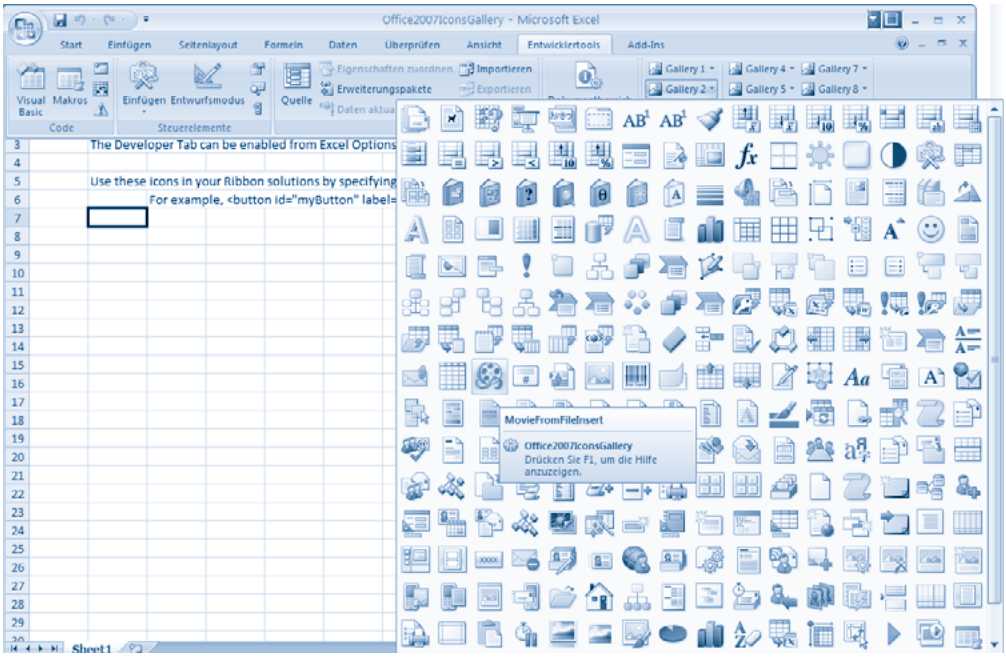
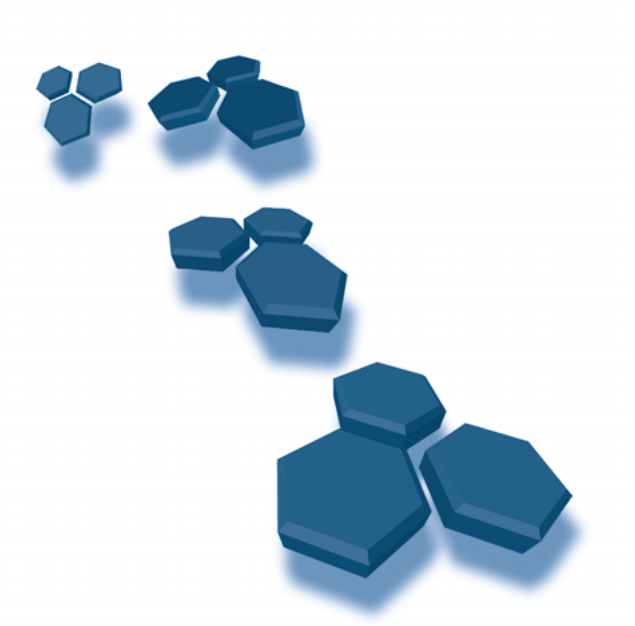


Abbildung 392: Die Auflistung der verfügbaren Bildelemente

Wenn Sie mit der Maus über die einzelnen Symbole streichen, dann sehen Sie in der QuickInfo den Namen des Bildelements. Diesen Namen geben Sie im XML-Markup beim Schlüsselwort `imageMso` an. Diese Symbole können Sie in allen Office-Programmen verwenden.

Teil II Anhang



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.
WeekDay	Gibt den Wochentag aus einem Datumswert zurück (1–7).

Tabelle 160: Die Funktionen der Kategorie Datum und Zeit

Funktion	Kurzbeschreibung
WeekDayName	Gibt eine Zeichenfolge zurück, die den festgelegten Wochentag angibt.
Year	Extrahiert die Jahrsinformation aus einem Datumswert.

Tabelle 160: 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 Bytes 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.
Space	Gibt eine Zeichenfolge vom Typ Variant (String) zurück, die aus einer angegebenen Anzahl von Leerzeichen besteht.

Tabelle 161: Die Textfunktionen

Funktion	Kurzbeschreibung
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 161: 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 162: 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 Bytes 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 oder 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 sequenziellen 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 Bytes 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.

Tabelle 162: Die Funktionen und Anweisungen der Kategorie Dateifunktionen und -anweisungen (Forts.)

Funktion/ Anweisung	Kurzbeschreibung
Print (A)	Schreibt Daten, die für die Ausgabe formatiert sind, in eine sequenzielle Datei.
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, gibt sie einen Wert vom Typ Variant (Double) zurück, der die Task-ID des Programms darstellt. Andernfalls wird null zurückgegeben.
Write (A)	Schreibt Daten in eine sequenzielle Datei.

Tabelle 162: 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 bei konstanten Raten und festem Zinssatz.
IRR	Gibt den internen Ertragssatz für eine Folge regelmäßiger Ein- und Auszahlungen zurück.
Log	Liefert den Logarithmus einer Zahl.

Tabelle 163: Die Funktionen der Kategorie Mathematik

Funktion	Kurzbeschreibung
MIRR	Liefert den modifizierten internen Ertragssatz für eine Folge regelmäßiger, mit unterschiedlichen Zinssätzen ausgestatteter Ein- und Auszahlungen.
NPer	Errechnet die Anzahl der Zeiträume für eine Annuität bei konstanter Zahlung und festem Zinssatz.
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 Kapitalanteil 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 Wertes 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 163: 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.

Tabelle 164: Die Prüffunktionen von Access

Funktion	Beschreibung
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.
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 164: 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 165: Typumwandlungsfunktionen

Farbfunktionen

Für den Einsatz von Farben können Sie in Access die Funktionen `RGB` oder `QBColor` einsetzen.

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	Rot-Wert	Grün-Wert	Blau-Wert
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 166: 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 167: 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 Feldes einer Tabelle/Abfrage, der einem bestimmten Kriterium entspricht.
DMax	Ermittelt den größten Wert eines Feldes in einer Tabelle/Abfrage.
DMin	Ermittelt den kleinsten Wert eines Feldes 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 168: Die Suchmethoden von Access

Übersicht der Konstanten

In diesem Teil des Anhangs können Sie eine Auswahl der wichtigsten Konstanten von Access sowie Konstanten allgemein einsetzbarer Funktionen finden.

Die 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	Rot-Wert	Grün-Wert	Blau-Wert
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

Die 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 Tabellen-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

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand

Konstante	Beschreibung	Wert
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
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 Datenbank-Eigenschaften 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

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
acCmdDocMaximize	Maximiert das Datenbankfenster	15
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 Formularkopf und -fuß an bzw. blendet sie 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-Hilfe-Fenster 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 Formular-Assistenten	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
acCmdPageSetup	Zeigt den Dialog SEITE EINRICHTEN an	32
acCmdPaste	Fügt Inhalt aus der Zwischenablage ein	191

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
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 Eigenschaftenfenster 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
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

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
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
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

Tabelle 170: Die wichtigsten Konstanten der Methode RunCommand (Forts.)

Konstante	Beschreibung	Wert
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
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 können bestimmte Konstanten eingesetzt werden, 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.

Tabelle 171: Die Konstanten der Funktion Dir

Konstante	Wert	Beschreibung
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 (Forts.)

Die Shell-Konstanten

Mithilfe der Funktion Shell können Sie jedes ausführbares 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 es erhält den Fokus.
vbNormalFocus	1	Das Fenster hat den Fokus, und die ursprüngliche Größe und Position wird 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 den dazugehörigen Konstanten können Sie Texte automatisch anpassen. Sie haben dabei unter anderem die Möglichkeit, Texte in Groß- bzw. Kleinbuchstaben umzuwandeln.

Entnehmen Sie aus 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 Wortes 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 in der folgenden Tabelle nachsehen 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`)

Tabelle 174: Die Konstanten der Funktion VarType

Konstante	Wert	Beschreibung
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 (Forts.)

File Input-/Output-Konstanten

Mithilfe dieser Konstanten können Sie den Zugriff auf eine Datei festlegen. 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 diese 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-/Output-Konstanten werden bevorzugt in der Verarbeitung von Textdateien angewendet.

Entnehmen Sie die einzelnen Möglichkeiten des Dateizugriffs aus 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 Datei.

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 FormatDateTime verwendet.

```
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 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 geklickt.
vbCancel oder 2	Die Schaltfläche Abbrechen wurde geklickt.
vbAbort oder 3	Die Schaltfläche Abbruch wurde geklickt.
vbRetry oder 4	Die Schaltfläche Wiederholen wurde geklickt.
vbIgnore oder 5	Die Schaltfläche Ignorieren wurde geklickt.
vbYes oder 6	Die Schaltfläche Ja wurde geklickt.
vbNo oder 7	Die Schaltfläche Nein wurde geklickt.

Tabelle 177: Rückgabewerte für Schaltflächen

Stichwortverzeichnis

A

Abfragen
  auflisten 185
  ausführen 356
  erstellen (ADO) 380
  erstellen (ADOX) 383
  erstellen (DAO) 381
  programmieren 353
  starten 354
Abs 102
Abschreibung
  errechnen 104, 105
AbsolutePosition 559
Absolutwert
  ausgeben 102
ACCDE-Format
  erstellen 586
AccessObject 183
Access-Tabelle
  transferieren 685
  übertragen 681
Access-Version
  feststellen 60, 188
Access-Verzeichnis
  ermitteln 145
ActiveConnection 337, 384
ActualWork 669
Adaptive Menüs
  ausschalten 747
AdaptiveMenus 748
Add 472, 619, 674, 683, 695, 740, 743, 749, 752
ADD COLUMN 373
ADD CONSTRAINT 373
AddFromFile 613, 638, 639
AddFromGuid 610
AddIns 607
AddItem 436, 444, 447, 473, 484, 705, 720
AddNew 329, 334, 533, 545, 665, 672, 712, 727
AfterUpdate 452
Aktionsschaltfläche
  beschriften 403
Aktivierreihenfolge
  anpassen 419
Aktualisierungsabfrage
  ausführen 356
Aktuellen Datensatz
  kopieren 575
Aktuelles Laufwerk
  wechseln 75
Aktuelles Verzeichnis
  ermitteln 75, 143
  wechseln 74
AllDataAccessPages 521
AllModules 183, 726
AllowEdits 543
AllowMultiSelect 403
AllReports 506
AllTables 184
ALTER TABLE 373
AND 303
Änderungen
  verhindern 553
Änderungsdatum
  ermitteln 37
Anfügeabfrage
  ausführen 358
Animation 228
Animationseffekt
  bestimmen 228
Anschluss
  auslesen 192
Ansicht
  aktualisieren 376
Ansparungsbetrag
  errechnen 110
Anwendernamen
  abfragen 190
  ermitteln 126
Anwendung
  finden 130

Anwendungspfad
ermitteln 188
Anzeigeposition
festlegen 739
API-Funktionen
einsetzen 123
Append 341, 344, 346, 348, 384, 599, 602, 603
Application 186
Applikation
beenden 190
ApplyFilter 193
ApplyTheme 519, 523
Arabische Zahlen
umwandeln 179
Arbeitsgruppe
anlegen (DAO) 599
entfernen (ADO) 601
entfernen (DAO) 600
AS SELECT 381
ASC 47, 74, 277, 476
Asc 172
AtEndOfStream 247
Attachments 672, 674
Attributes 340
Aufgaben
abfragen 667
Auswahl
aufheben 425
Auswahlabfrage
ausführen 368
AVG 371
Avg 290

B
BackColor 420, 424, 503, 571, 708
Balloon 225
Balloon-Objekt
anzeigen 229
BalloonType 227
Beep 151
BeforeDelConfirm 556
Benutzer
anlegen (ADO) 602
anlegen (DAO) 601
auslesen (DAO) 604
entfernen (DAO) 603
Berechtigungen
zuweisen (ADO) 605
Berichte
auflisten 505
drucken 496
exportieren 500
formatieren 501
kopieren 497
öffnen 493
programmieren 491
schließen 495
umbenennen 499
Berichtselemente
einfügen 513
identifizieren 506
Berichtsfilter
setzen 494
Berichtsteil
ausblenden 510
einblenden 510
Berichtstypen 491
BETWEEN 393
Bibliotheken
auslesen 224
einbinden 611
Bibliotheks-Objektnamen
auslesen 613
Bilder
umbenennen 81
Bildschirmaktualisierung
ausschalten 201
einschalten 201
Bildschirmauflösung
ermitteln 129
Birthday 661
Blinkende Schaltfläche
erstellen 461
Body 666, 668, 672, 674
Bold 701
Bookmark 324
Bremsweg
ermitteln 164

Buchstaben
entfernen 47
zählen 176
BusinessAddressCity 661
BusinessAddressPostalCode 661
BusinessAddressState 661
BusinessAddressStreet 661
BusinessFaxNumber 661
BusinessTelephoneNumber 661
Button 229
ButtonName 403

C
Caption 541, 569, 701, 735, 737, 752
Case Else 49
Catalog 335, 337, 603
Categories 666, 669
CDate 25, 118, 181, 448
CDbl 117
CD-ROM-Laufwerk
bedienen 128
ermitteln 124
öffnen 128
schließen 128
Cell 684
Cells 485
Change 432, 447
CharLower 148
CharUpper 148
ChDir 74, 166
ChDrive 75
Choose 49
Chr 50, 113
Clear 704
ClearContents 485
CLng 119
Clone 321
Close 89, 94, 201, 247, 261, 268, 276, 303, 315, 328, 331, 418, 477, 495, 545, 552, 555, 624, 672, 687
CloseClipboard 149
CloseCurrentDatabase 190
CodeModul 607
CodeModule 639
CodePanes 607
Codes
sichern 725
Column 337, 339, 445, 707
ColumnCount 700
Columns 340
ColumnWidth 700
CommandBar 735
CommandBarControl 752
CommandBars 607, 731, 737
Complete 669
Computername
ermitteln 127
Connect 350
Connection 261, 298, 595
Control 192
Controls 412, 737, 755
ControlTipText 428, 542, 580
ControlType 413, 427, 507, 542
CopyObject 196, 497
COUNT 371
Count 290, 340, 656, 672, 676, 732, 737
CountOfDeclarationLines 635
CountOfLines 627, 635, 637, 641
CREATE INDEX 377
CREATE TABLE 375
CREATE VIEW 381
CreateDataAccessPage 519
CreateDirectory 135
CreateEventProc 640
CreateField 343, 344, 348
CreateFolder 237
CreateGroup 599
CreateItem 660, 674
CreateObject 485, 570, 657, 660, 679, 683, 694, 712, 719
CreateQueryDef 382, 390
CreateReport 511, 516
CreateReportControl 513
CreateTableDef 344
CreateTextfile 248
CreateUser 602
CSV-Datei 217
CurDir 75
CurrentData 185
CurrentDB 530

CurrentDb 38, 117, 267, 344, 382
CurrentProject 184, 186, 521, 683
CurrentUser 93, 190
CursorLocation 279
Cursorposition
  ermitteln 156

D
Database 38, 267
Date 26, 433, 447
DateAdd 27
DateCreated 184, 236
DateDiff 31
Dateiansicht
  festlegen 404
Dateiattribute
  setzen 97
Dateidialogfeld
  anzeigen 404
Dateien
  auflisten 76, 242
  kopieren 80, 231
  löschen 79, 234
  suchen 40, 152
  umbenennen 84
  verschieben 232
  zählen 169
Datei-Endung
  prüfen 63
Dateiexistenz
  prüfen 85, 235
Dateifilter
  festlegen 403
Dateiinfo
  abfragen 236
Dateiinformationen
  auslesen 153
Dateiname
  ermitteln 54
Dateisystem
  ermitteln 246
Dateityp
  identifizieren 146
DateLastAccessed 236
DateLastModified 236
Daten
  ausgeben (DAO) 267
  gruppieren (ADO) 294
Datenbank
  öffnen 168
  öffnen (ADO) 260, 594
  öffnen (DAO) 594
  öffnen mit Access 2007(ADO) 261
  schließen 190
  schützen 590
  überprüfen 166
  verschlüsseln 587
Datenbankabfragen
  starten 713
Datenbankinhalte
  transferieren 212
Datenbankkennwort
  ändern (DAO) 597
Datenbankname
  ermitteln 38, 186
Datenbankpfad
  ermitteln 187
Datenbankstatus
  überprüfen 167
Datenfeld
  überprüfen 115
Datensätze
  aktivieren 206
  auswählen 254
  clonen (ADO) 321
  clonen (DAO) 323
  einlesen (DAO) 270
  extrahieren 264
  filtern (ADO) 311
  filtern (DAO) 313
  finden 252
  finden (ADO) 296
  finden (DAO) 300
  hinzufügen (ADO) 329
  hinzufügen (DAO) 333
  in Array lesen (ADO) 268
  löschen (ADO) 326
  löschen (DAO) 327
  sortieren (ADO) 277
  sortieren (DAO) 279

suchen 779
summieren 320, 779
verändern (ADO) 304
verändern (DAO) 306
zählen 779
zählen (ADO) 315
zählen (DAO) 318
Datensatzlöschung
verhindern 555
Datensatzmarkierer
ausblenden 423
einblenden 423
Datenträgerdateien
schließen 95
Datenzugriffsseiten
anpassen 523
anzeigen 522
auslesen 520
erstellen 517
DatePart 29
DateSerial 32, 182
DateValue 34
Datum
zerlegen 35
Datumsabfrage
durchführen 394
Datumsangaben
formatieren 41
Datumsdifferenzen
errechnen 31
Datumstabelle
öffnen 37
Datumswerte
erkennen 34, 118
umsetzen 34
umwandeln 118
DAvg 292
Day 35, 464
DayLength 464
DCount 292, 316
DDB 104
Debug.Print 28, 732
DefinedSize 340
Definition
Aktionsabfragen 353
Auswahlabfragen 353
Kreuztabellenabfragen 353
Parameterabfragen 353
SQL-Abfragen 354
DELETE 361
Delete 326, 327, 347, 603, 604, 739, 751
DeleteControl 628
DeleteFile 234
DeleteFolder 237
DeleteLines 625, 627
DeleteObject 197, 399
DESC 277, 476
Description 196
DeviceName 192
DFirst 292
Diagramm-Assistent 478
Diagramme
einfügen 475, 478
Dialoge
anzeigen 405
Diashow
programmieren 677
Dir 40, 76, 82, 94, 167, 437, 473, 484
Direktbereich
editieren 732
Direktfenster
editieren 28
Dirty 554
Display 650
DLast 292, 546
DLookUp 292, 555, 577
DMax 292, 322, 559
DMin 292
Do Until 40, 56, 268, 272, 328
Do While 314
DoCmd 193, 353, 492
Dokumenteigenschaften
auslesen 180
DoMenuItem 459
Doppelte Eingaben
verhindern 576
DriveLetter 244
DriveType 244
DROP COLUMN 373

DROP CONSTRAINT 373
DROP INDEX 378
DROP TABLE 379
Drucker
  auflisten 191
Druckername
  ermitteln 192
DstDev 292
DStDevP 292
DSum 292, 321
DueDate 669
Dump
  ziehen (ADO) 262
Duration 667
Durchschnittsverbrauch
  errechnen 161
DVar 292
DVarP 292

E
Early-Binding 647
Echo 201
Editor
  aufrufen 99
Eigene Berichte
  erstellen 511
Eigene Löschabfrage
  einsetzen 557
Eingabedialog
  anzeigen 400
Eingaben
  erzwingen 401
  limitieren 543
  prüfen 114
  vorwegnehmen 577
Eingabe-Prüfung
  vornehmen 549
Email1Address 661
E-Mail-Adressen
  auslesen 655
E-Mail-Anhänge
  speichern 675
E-Mails
  importieren 670
  mit Anhang versenden 654
  verschicken 651
Enabled 429, 738, 756, 757
End 666
End With 752
Endtermine
  errechnen 27
Environ 86, 433
EOF 268, 276, 305, 597, 692
Err 39, 168
Ersten Datensatz
  ermitteln 779
EuroConvert 308
Excel-Prozedur
  starten 716
Excel-Status
  ermitteln 693
Execute 271, 353
Exit Function 172
Exit Sub 83, 550, 639
Exklusivmodus 590
Export 624, 642, 643
ExportXML 259
Externes Programm
  aufrufen 131
  starten 98

F
FaceId 740, 753
Fahrleistung
  berechnen 163
FastLaserPrinting 497
Fehler
  abfangen 80
  ignorieren 739
Fehlerbeschreibung
  abrufen 39
Fehlerhafte Verweise
  ermitteln 616
Fehlernummer
  abrufen 39
  ausgeben 196
Felder
  aktivieren 205
  ausblenden 539
  ausgeben (ADO) 265

einblenden 539
fokussieren 528
hinzufügen 373
löschen 373
vorbelegen 546
Feldlöschung
verhindern 578
Fenstergröße
festlegen 211
maximieren 211
minimieren 211
wiederherstellen 212
Fields 268, 276, 307, 314, 331, 334, 533
FileAttr 96
FileCopy 80
FileDateTime 78
FileDialog 401, 404
FileDialogSelectedItems 404
FileExists 235
FileLen 85
FileName 474
FileSystem 231, 246
FileSystemObject 181, 247
FileTimeToLocalFileTime 153
FileTimeToSystemTime 153
FillColor 503
FillStyle 503
Filter 312
entfernen 195
löschen 403
setzen 193
Filters 403
Find 296, 555, 559, 629, 632, 692
FindExecutable 146
FindFirst 300, 301, 328, 664
FindFirstFile 153
FindLast 300
FindNext 300, 328
FindPrevious 300
FindRecord 252
FindWindow 130
FIRST 371
First 290
FirstDay 464
FirstName 657, 661
Fix 106
FolderExists 238
FontBold 502, 503, 538
FontItalic 503
FontName 503
FontSize 503
FontUnderline 503
For Each Next 186, 193, 244, 335, 413, 542, 611, 614, 735, 754
For Next 74, 105
ForeColor 424, 502, 503, 701
Form_Activate 557
Form_AfterInsert 544
Form_AfterUpdate 554
Form_BeforeInsert 545
Form_BeforeUpdate 548, 549
Form_Click 562
Form_Close 530
Form_Current 538
Form_DblClick 561
Form_Delete 555
Form_Load 534
Form_MouseDown 563
Form_MouseUp 563
Form_Open 525
Form_Resize 559
Form_Unload 535
FormatDateTime 41
Formularcodes
importieren 640
sichern 643
Formulare
auslesen 412
erstellen 406
löschen 399, 629
öffnen 206, 416
schließen 418
Formular-Ereignisprozedur
einstellen 640
Formularer-Eignisse
löschen 627
Formularhintergrund
einstellen 420
Fortschrittsbalken
anpassen 469

programmieren 467
FreeFile 94
FreeSpace 246
FROM 284, 363, 381
FullName 187, 225, 521
FullPath 615
Funktion
beenden 172
FV 110

G
Ganzzahligen Wert
ermitteln 106
Gesamtkapazität
ermitteln 246
GetAttr 96
GetClipboardData 149
GetCommandLine 145
GetComputerName 128
GetCurrentDirectory 143
GetCursorPos 156
GetDefaultFolder 649, 656, 668, 672
GetDeviceCaps 129
GetDriveType 125
GetExitCodeProcess 132
GetKeyboardState 151
GetLogicalDriveStrings 125
GetNameSpace 649
GetObject 649, 672, 679, 680, 692, 694, 718
GetRows 268, 270
GetShortPathName 147
GetString 265, 266
GetSystemDirectory 141
GetTempPath 142
GetUserName 126, 527
GetVersionEx 138
GotFocus 570
GoTo 692
GoToControl 205
GoToRecord 206, 254, 539, 559
Großbuchstaben
umwandeln 58
Group 599
GROUP BY 294
Groups 599, 604

H
HasLegend 477
HasModul 645
HasTitle 477
HAVING 295
Heading 226
Hintergrund
anpassen 423
Hour 42

I
Icon 227
ID 735, 750
Idealgewicht
bestimmen 162
If 166
Index
definieren 377
entfernen 378
Industriezeit
errechnen 159
Initialen
erstellen 171
InitialFileName 403
Initialize 690
InitialView 404
INNER JOIN 364
InnerText 523
Input 89
InputBox 72, 400
INSERT INTO 358
InsertLines 637, 641
InStr 52, 56, 175
InStrRev 54
Int 106
Integer 47
Internet
aufrufen 568
Internet Explorer
aufrufen 101
InternetGetConnectedState 155
Internetverbindung
überprüfen 155
INTO 368
IsArray 115

IsBroken 617
IsLoaded 185, 418, 531
IsNull 551, 578, 674
IsNumeric 114
IsObject 116
ItemData 438
ItemsSelected 441, 442

J
Join 56

K
Kalender
  erstellen 32
Kalender-Steuerelement
  programmieren 463
Kalenderwoche
  ermitteln 29
Kennwort
  festlegen 585
KeyDown 572
KeyUp 572
Kill 79, 83, 94, 624
Kleinbuchstaben
  umwandeln 58
Kombinationsfeld
  editieren 450
  füllen 446, 449
  programmieren 446
Kontakt
  hinzufügen 665
  speichern 661
Kontaktdaten
  importieren 661
  übertragen 658
Kontextmenü
  deaktivieren 565
Kontrollkästchen
  auswerten 453
  initialisieren 454
  programmieren 453
Kundenabfrage
  durchführen 385
Kurze Pfadnamen
  ermitteln 147

L
Labels 227
Lagerbestand
  erhöhen 562
  ermitteln 287
Lagerbestandsdurchschnitt
  errechnen 291
Lagerhüter
  ermitteln 289
LAST 371
Last 290
LastName 657, 661
Late-Binding 647
Laufwerke
  auslesen 243
Laufwerksbuchstaben
  abfragen 244
Laufwerkseigenschaften
  auslesen 245
Laufwerksname
  abfragen 246
LCase 58
Leerzeichen
  auffüllen 66
  entfernen 72
Left 56, 61, 432, 739
Leisten
  identifizieren 731
Len 63, 176
Lesezeichen
  setzen (ADO) 324
Letzte Änderung
  festhalten 554
Letzten Datensatz
  einstellen 557
Letzten Satz
  auslesen 538
Lieferantenabfrage
  durchführen 386
Liefertermin
  errechnen 27
LIKE 361
Like 176
LimitToList 451
Line Input 89, 91, 727

ListCount 439, 441, 535, 705
Listenfeld
  auslesen (mehrspaltig) 444
  füllen 436, 534
  füllen (mehrspaltig) 443
  programmieren 435
Listenfeldauswahl
  verarbeiten 438
Listenfeldeintrag
  abwählen 442
  löschen 439, 441
  markieren 442
  zählen 439
ListIndex 438, 440, 445, 690, 707
ListRows 447
Location 666
Locked 429
LockType 596
Löschabfrage
  ausführen 361
Löschkriterium
  festlegen 361
Löschrückfrage
  einholen 399
LostFocus 570

M

Mailadressen
  überprüfen 52
MainWindow 618
Markierung
  festlegen (Textfeld) 425
MAX 371
Max 290, 468, 488
Maximalwert
  ermitteln 779
Maximize 211
mciSendString 128
MDE-Format
  erstellen 586
Media Player-Steuerelement
  einsetzen 472
Mehrfachindex
  hinzufügen 373
  löschen 373
Menü
  aktivieren 755
  auslesen 753
  deaktivieren 755
  einfügen 748
  löschen 751
Menübefehle
  aktivieren 757
  ausführen 208
  auslesen 754
  deaktivieren 757
  einfügen 751
  gruppieren 753
Menüleiste
  zurücksetzen 750
Merhfachauswahl
  zulassen 403
Mid 56, 62, 125, 176
MIN 371
Min 290, 468, 488
Minimalwerte
  ermitteln 779
  ermitteln (ADO) 286
  ermitteln (DAO) 293
Minimize 211
Minuten 42
  umrechnen 159
MkDir 78, 134
Mod 505
Mode 227
Modulcheck
  durchführen 622
Module
  auflisten 183
  drucken 623
  einfügen 619
  löschen 621
Monatsletzten
  ermitteln 181
Monatsnamen
  ermitteln 43
Month 35, 44, 160, 182, 447, 464
MonthLength 464
MonthName 43, 447
MouseMove 566

MoveFile 232
MoveFirst 548
MoveNext 272, 280, 305, 314, 327, 660, 684
MoveSize 211
MsgBox 397, 452, 549
Multifunktionsleiste 731
  programmieren 758, 760
Muss-Felder
  definieren 551

N
Nachkommastellen
  entfernen 119
Name 38, 55, 84, 185, 186, 193, 335, 412, 506, 512, 614, 732
Navigate 470, 570
NavigationButtons 423
Navigationsschaltflächen
  ausblenden 421
Neuanlage
  verhindern 553
New 476
NewCurrentDatabase 344
NewPassword 598
NewRecord 541, 543, 545, 553
NextDay 464
NextMonth 465
NextWeek 465
NextYear 465
NoMatch 302, 328, 664
Normalzeit
  umwandeln 160
Nothing 268, 340, 382, 665
NotInList 451
Now 95, 248, 555
Number 196
NumericScale 340
Numerische Zeichen
  zählen 175

O
Object 657
Objekte
  drucken 200
  exportieren 198
  kopieren 196
  löschen 197
  umbenennen 195
  versenden 201
Objektstatus
  erkennen 168
Office-Assistenten
  einsetzen 226
OldValue 426, 552
On Error 80, 405
On Error GoTo 196
On Error Resume Next 739
OnAction 744
Open 88, 91, 92, 168, 261, 263, 471, 533, 552, 596, 683, 687, 721, 722, 727
OpenClipboard 149
OpenDataAccessPage 522
OpenDatabase 598
OpenForm 206, 416, 628
OpenProcess 132
OpenQuery 353, 354, 454
OpenRecordset 268, 276, 314, 674
OpenReport 405, 493
OpenTable 117, 194, 251
OpenText 624
OpenTextFile 247
Optionsfelder
  programmieren 455
OR 303
ORDER BY 266, 277, 325
Ordner
  auflisten 240
Ordnerexistenz
  prüfen 238
Ordnergröße
  ermitteln 239
Outlook-Postfach
  ansteuern 470
Outlook-Status
  prüfen 648
OutPutTo 645
OutputTo 198, 257, 500, 685, 725

P
Parameterabfragen
erstellen (ADOX) 387
erstellen (DAO) 390
PARAMETERS 388
ParentCatalog 340
Path 187, 689
Pfadname
ermitteln 54
Picture 439
Piepton
erzeugen 151
Port 192
Positionen
auslesen 172
Posteingang-Ordner
anzeigen 650
PowerPoint-Dateien
suchen 719
PowerPoint-Status
abfragen 718
Precision 340
Preisabfrage
durchführen 392
PreviousDay 465
PreviousMonth 465
PreviousWeek 465
PreviousYear 465
Print 92, 552, 723
Printers 191
PrintOut 200, 496, 624
ProcCountLines 625, 629
ProcOfLine 635
ProcStartLine 625, 626
Properties 340, 345, 595
Protection 745
Provider 261, 595
Prozedur-Check
durchführen 626
Prozeduren
auflisten 634
beenden 83
einfügen 637
exportieren 642
löschen 624
Punkte
entfernen 117
Put 93, 94

Q
QBColor 503, 565, 571
Quellcode
schützen 585
QueryDef 381
QuickInfo
anzeigen 579
definieren 542
hinzufügen 428
Quit 190, 485, 705

R
Randomize 112
Rate 109
ReceivedTime 672
Rechnungsfälligkeiten
überwachen 537
RecordCount 315, 318, 530, 539, 543
RecordSelectors 423
Recordset 267, 272
RecordsetClone 539, 543, 548, 575
RecordSource 512, 516, 539, 548
Reference 224, 615
References 615, 617
RefreshDataBaseWindow 376
RefreshLink 352
Reihenfolge
Ereignisse 581
RelatedColumn 340
ReminderMinutesBeforeStart 667, 669
ReminderPlaySound 667, 669
ReminderSet 667, 669
ReminderSoundFile 669
Remove 613, 614, 621
RemoveDirectory 135
RemoveItem 440, 442
Rename 195, 499
Replace 65, 633
ReplaceLine 634
Report 512
Report_Close 530

Report_Open 525
Rept 717
RequiredAttendees 666
Reset 95, 750
Restore 212
RGB 420, 424, 503, 701
Ribbon 731
Right 64
RmDir 79
Römische Zahlen
umwandeln 177
RootFolder 246
Round 107, 246
RowSource 450, 451, 452
RowSourceType 450
RTrim 171
Run 687, 716
RunCommand 208, 405, 460, 494
RunSQL 210, 353, 356

S

Sammel-E-Mails
versenden 673
Save 661, 674
SaveAsFile 676
Schaltfläche
anpassen 564
einfügen 459
programmieren 457
Schaltjahr
überprüfen 160
Schließen
verhindern 535
Schriftfarbe
anpassen 423
SearchTreeForFile 152
Second 42
SELECT 265, 272, 284
SELECT * 368
Select Case 62, 74, 172, 181, 338, 510, 550
SELECT DISTINCT 273
SELECT INTO 363
SELECT TOP 5 287
Selected 442, 535
SelectedItems 404
Selection 683
SelLenght 425
SelStart 425
SenderName 672
SendObject 201, 651, 654
Set 193, 340, 476, 632, 644, 680, 743
SetAttr 97
SetCurrentDirectoryA 145
SetData 477
SetFocus 426, 529, 550, 618
SetPermissions 605
SetSelection 630
SetWarnings 357
Sgn 103
SHBrowseForFolder 136
Shell 98, 133
ShellAbout 144
ShellExecute 131, 568, 653
SHGetPathFromIDList 136
Show 229, 404, 690
ShowAllRecords 195
ShowDateSelectors 464
ShowToolbar 209, 733
Sicherheitscheck
durchführen 529
Size 239
SizeToFit 516
Sleep 134, 468, 678
Slider-Steuerelement
einsetzen 486
SLN 105
sndPlaySound32 150
Sonderzeichen
entfernen 174
Sort 279, 476
SortOrder 340
Soundkarte
überprüfen 149
Sounds
ausgeben 150
Space 66
SpecialEffect 568, 708
Speicherrückfrage
einholen 548

Spezialeffekt
einsetzen 566
Spinbutton-Steuerelement
einsetzen 488
Split 57
SpreadSheet-Steuerelement
einsetzen 483
SQL-Abfragen
ausführen 210
SQL-Statements
absetzen (ADO) 271
Stammverzeichnis
abfragen 246
Standardabweichung
ermitteln 779
Standarddialoge
anzeigen 401
Standardmeldungen
unterdrücken 357
Standardverzeichnis
festlegen 145
Start 666
StartDate 669
STDEV 371
StDev 290
Step 33
Steuerelement_BeforeUpdate 576
Steuerelement_Enter 579
Steuerelement_Exit 580
Steuerelemente
löschen 627
Steuerelementtyp
identifizieren 413
Str 67, 121
StrComp 68
StrConv 59
String 70
StrReverse 69
Stückzahl
errechnen 157
Stunden
umrechnen 158
Stundengeschwindigkeit
errechnen 161
Subject 666, 668, 672, 674
SUM 371
Sum 290
Switch 71
Symbolbeschriftung
ermitteln 735
Symbolleisten
ausblenden 209, 733
einblenden 209, 733
erstellen 738
löschen 739
schützen 744
Symbolleisten-IDs
ermitteln 735
Symbolschaltflächen 739, 741, 742, 746
aktivieren 737
deaktivieren 737
Symbolschaltflächenaussehen
festlegen 739
Symboltyp
festlegen 227
SysCmd 188
Systemdatum
abrufen 26

T

TabDef 348
Tabellen
aktualisieren 350
auflisten 184
erstellen 341, 375
konvertieren 257
löschen 347, 379
öffnen 117, 251, 263, 268
programmieren 251
Steuerelemente für Formulare 410
transferieren 215
verknüpfen 256, 345
verlinken (DAO) 349
zusammenfassen (ADO) 281
Tabellenänderungsabfrage
ausführen 373
Tabellencheck
vornehmen 116
Tabelleneigenschaften
abfragen 339

auslesen (ADOX) 336
Tabellenerstellungsabfrage
durchführen 363
Tabellenfeld
anhängen (DAO) 347
Tabellennamen
ermitteln 335
Tabellenstrukturen
auslesen (ADOX) 337
Table 337
TableDefs 117, 350
Tables 340
Tasten
abfangen 151
erkennen 572
Tastenkombinationen
einrichten 573
Temporäres Verzeichnis
ermitteln 142
Termine
abfragen 665
Textdatei
auslesen 91
einlesen 247
erstellen 95
importieren 726
lesen 88, 89
öffnen 88
schließen 89
schreiben 92, 248
speichern 721
Texte
ersetzen 631
exportieren 216
importieren 216
konvertieren 148
vervollständigen 70
zerlegen 57, 62
Textfelder
ausblenden 430
begrenzen 432
einblenden 430
hervorheben 570
identifizieren 192
initialisieren 434
leeren 709
programmieren 423
prüfen 435
schützen 429
sperren 429
überwachen 432
Textlänge
feststellen 63
Textstream 247
Time 26, 433
TimerInterval 531
TimeValue 34
Title 404
Today 465
ToolTipText 740
TOP 286, 293
Top 739
TotalSize 246
TransferDatabase 212, 257
TransferSpreadsheet 215, 696
TransferText 216
TreeView 472
TreeView-Steuerelement
einsetzen 471
Trim 72
Type 335, 338, 340, 444, 617, 622, 636, 732
TypeName 193, 434, 710
TypeText 683, 692

U

UBound 58, 116
UCase 58, 59, 541
Uhrzeit
abrufen 26
Umgebungsvariablen
auslesen 86
Umlaute
tauschen 66
Unikatliste
erstellen (ADO) 273
erstellen (DAO) 275
UNION 281, 368
UNIQUE 377
UnRead 672
UPDATE 356

Update 304, 322, 545, 555, 672, 728
URLs
  auslesen 678
Userform
  einfügen 699
Users 603, 604

V

Val 120
Value 455, 464, 468, 487, 489, 542
Value List 450
VAR 371
Variable Feldanpassung
  vornehmen 559
Variablen Titel
  einsetzen 541
Varianz
  feststellen 779
VBComponent 607
VBE
  aufrufen 618
VBE-Bibliothek
  einbinden 608
VBE-Programmierung 607
VB-Komponenten
  identifizieren 617
VBProjects 607
Vergleichsabfrage
  durchführen 364
Verknüpfte Tabellen
  abfragen (ADO) 282
  abfragen (DAO) 285
Verknüpfungsadresse
  anpassen 351
Version 60, 189
Verweise
  auslesen 615
Verzeichnis
  anlegen 78, 237
  auflisten 76
  erstellen 134
  löschen 79, 135, 237
  überprüfen 165
Verzeichnisbaum
  anzeigen 136
Visible 430, 510, 527, 540, 561, 618, 683, 700, 705, 733, 739
VolumeName 246
Vorzeichen
  auslesen 103

W

Währungsumrechnungen
  durchführen (ADO) 308
  durchführen (DAO) 310
Warnmeldung
  anzeigen 397
WaveOutGetNumDevs 149
Webbrowser
  programmieren 469
WeekDay 45
WeekDayName 45
WHERE 272, 284, 286, 361, 381
Window 618
Windows 607
Windows-Explorer
  öffnen 100
Windows-Info-Dialog
  anzeigen 144
Windows-Systemverzeichnis
  ermitteln 141
Windows-Version
  ermitteln 138
WindowWidth 560
With 752
Wochennummer
  ausgeben 30
Wochentag
  ermitteln 45
Word-Dokument
  öffnen 687
  schließen 687
Word-Prozedur
  starten 687
Word-Status
  abfragen 680
WorksheetFunction 179
Write 95

Y

Year 35, 182, 464

Z

Zahl
  entfernen 48
  extrahieren 120
  runden 107
  umwandeln 50
Zahlungsart
  definieren 49
Zeichen
  auslesen 90
  ersetzen 65
Zeichenfolgen
  spiegeln 69
  umwandeln 25
  vergleichen 51, 68
Zeile
  lesen 89
Zeit
  zerlegen 42
Zeitwerte
  erkennen 34
  umsetzen 34
Zinssatz
  errechnen 109
Zoom-Darstellung 482
Zufallsbuchstaben
  erzeugen 113
Zufallszahlen
  erzeugen 112
Zugangsverwaltung
  aufsetzen 525
Zugriff
  dokumentieren 532
Zugriffsmodus
  feststellen 96
Zuletzt geänderten Datensatz
  aufrufen 558
Zuordnungen
  treffen 71
Zwischenablage
  löschen 149

informit.de, Partner von Addison-Wesley, bietet aktuelles Fachwissen rund um die Uhr.

www.informit.de

In Zusammenarbeit mit den Top-Autoren von Addison-Wesley, absoluten Spezialisten ihres Fachgebiets, bieten wir Ihnen ständig hochinteressante, brandaktuelle deutsch- und englischsprachige Bücher, Softwareprodukte, Video-Trainings sowie eBooks.

wenn Sie mehr wissen wollen …

www.informit.de

THE SIGN OF EXCELLENCE

Das Excel-VBA-Codebook liefert Tipps, Tricks und Lösungen zu allen Problemen der täglichen Programmierarbeit mit den Excel-Versionen 2000 bis 2007. Alle diese „Rezepte“ sind in Kategorien sortiert und somit sehr leicht auffindbar. Die CD zum Buch enthält nochmals alle vorgestellten Codes, sodass diese sehr schnell in eigene Projekte eingefügt werden können.

Melanie Breden; Michael Schwimmer
ISBN 978-3-8273-2471-9
99.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
ADDISON-WESLEY

THE SIGN OF EXCELLENCE

Excel für Profis - Ihr Einstieg in die VBA-Programmierung für Excel 2007. Das Buch gibt einen Überblick über die unter Excel verfügbaren Objektbibliotheken und hilft Ihnen, auch fortgeschrittene Aufgaben mühelos zu meistern. Sie lernen die erforderlichen Werkzeuge kennen und erhalten das notwendige Know-how für den erfolgreichen Einsatz. Ein eigenes Kapitel ist den Visual Tools for Office gewidmet, mit denen sich z.B. das Anpassen der Multifunktionsleiste, die Programmierung von individuellen Aufgabenbereichen oder die Abfrage von Web Services realisieren lassen.

Michael Kofler; Ralf Nebelo
ISBN 978-3-8273-2567-9
49.95 EUR [D]

www.addison-wesley.de

[The Sign of Excellence]
ADDISON-WESLEY