

DAS BUCH

VIEL WISSEN

Manfred Hoffbauer
Christoph Spielmann

Access XP

Programmierung

Version 2002



Das ganze Buch
sowie eine
Referenz auf CD!

Manfred Hoffbauer/Christoph Spielmann

Access XP Version 2002

Programmierung –

Das Buch



Fast alle Hard- und Software-Bezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

Der Verlag hat alle Sorgfalt walten lassen, um vollständige und akkurate Informationen in diesem Buch bzw. Programm und anderen evtl. beiliegenden Informationsträgern zu publizieren. SYBEX-Verlag GmbH, Düsseldorf, übernimmt weder die Garantie noch die juristische Verantwortung oder irgendeine Haftung für die Nutzung dieser Informationen, für deren Wirtschaftlichkeit oder fehlerfreie Funktion für einen bestimmten Zweck. Ferner kann der Verlag für Schäden, die auf eine Fehlfunktion von Programmen, Schaltplänen o.Ä. zurückzuführen sind, nicht haftbar gemacht werden, auch nicht für die Verletzung von Patent- und anderen Rechten Dritter, die daraus resultiert.

Projektmanagement: Simone Schneider

Endkontrolle: Mathias Kaiser Redaktionsbüro, Düsseldorf

Satz: reemers publishing services gmbh, Krefeld

Umschlaggestaltung: Guido Krüsselsberg, Düsseldorf

Farbproduktionen: More Colour, Meerbusch

Belichtung, Druck und buchbinderische Verarbeitung: Bercker, Grafischer Betrieb, Kevelaer

ISBN 3-8155-0167-9

1. Auflage 2001

Dieses Buch ist keine Original-Dokumentation zur Software der Firma Microsoft. Sollte Ihnen dieses Buch dennoch anstelle der Original-Dokumentation zusammen mit Disketten verkauft worden sein, welche die entsprechende Microsoft-Software enthalten, so handelt es sich um Raubkopien der Software. Benachrichtigen Sie in diesem Fall umgehend Microsoft GmbH, Edisonstr. 1, 85716 Unterschleißheim. Auch die Benutzung einer Raubkopie kann strafbar sein. Der Verlag und Microsoft GmbH

Alle Rechte vorbehalten. Kein Teil des Werks darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder in einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Printed in Germany

Copyright © 2001 by SYBEX-Verlag GmbH, Düsseldorf



Vorwort		XVII
Hinweise zum Aufbau des Buches		XIX
Einführung in die Makroprogrammierung	Kapitel 1 —○	1
Schnellstart VBA	Kapitel 2 —○	31
Die Programmiersprache VBA	Kapitel 3 —○	53
Programmieren mit Objekten	Kapitel 4 —○	193
Datenzugriff mit VBA	Kapitel 5 —○	225
Formulare programmieren	Kapitel 6 —○	307
Steuerelemente programmieren	Kapitel 7 —○	371
Berichte programmieren	Kapitel 8 —○	417
Objektorientierte Programmierung	Kapitel 9 —○	443
Design von Datenbankobjekten	Kapitel 10 —○	469
Vorgehensweise bei der Anwendungsentwicklung	Kapitel 11 —○	541
Zugriff auf das Sicherheitssystem	Kapitel 12 —○	603
OLE und DDE	Kapitel 13 —○	627
Den Funktionsumfang von Access erweitern	Kapitel 14 —○	689
Index		737



Vorwort		XVII
Hinweise zum Aufbau des Buches		XIX
Grundlagen		XX
Programmierpraxis		XX
Anwendungsentwicklung		XXI
Spezielle Programmiertechniken		XXI
Sicherheitshinweis		XXI
Einführung in die Makroprogrammierung	Kapitel 1	1
Das erste Makro		3
Übersicht über alle Aktionen		9
Bedingungen		12
Gruppenmakros		17
Editieren von Makros		20
Fehlersuche und Fehlermeldungen		22
Ereignisse		23
Der Makro-Generator		25
Übersicht aller Ereignisse		26
Ereigniseigenschaften von Formularen		27
Schnellstart VBA	Kapitel 2	31
Prozeduren		32
Aufbau einer Prozedur		34
Anzeigen einer Meldung		36
Erweitern der Prozedur		37
Das <i>Fokuserhalt</i> -Ereignis		42





Ausführen von Aktionen	44
Angabe der Aktionsargumente	46
Werte für Parameter ermitteln	47
<i>If...Then...Else</i>	50
Ausführung mehrerer Anweisungen	51
 Die Programmiersprache VBA	 Kapitel 3
 Das Modulfenster	 55
Die VBA-Entwicklungsumgebung	55
Das Bearbeiten von Prozeduren im Modulfenster	56
Arbeiten mit dem Modulfenster	58
 Module	 67
 Prozeduren	 69
Erstellen einer Prozedur	70
Aufbau von <i>Sub</i> -Prozeduren	71
Aufruf einer <i>Sub</i> -Prozedur	72
Strukturierte Programmierung	74
Geltungsbereiche von Prozeduren	75
 Ausdrücke	 78
Numerische Ausdrücke	79
Logische Ausdrücke	86
Zeichenketten	89
Datums- und Zeitausdrücke	100
 Variablen	 108
Deklaration	109
Numerische Datentypen	113
Der Datentyp <i>Boolean</i>	115
Der Datentyp <i>Date</i>	116
Der Datentyp <i>Variant</i>	116
Typumwandlung	120
Geltungsbereich und Lebensdauer von Variablen	125
Benutzerdefinierte Datentypen	129
Arrays	130



Konstanten	136
Parametervariablen	137
Übergabe mehrerer Parameter	139
Optionale Parameter	139
Parameter-Arrays	140
Übergabe von Arrays	142
<i>Function</i> -Prozeduren	143
<i>Call by reference</i> und <i>Call by value</i>	145
Kontrollstrukturen	149
<i>If...Then...Else</i>	149
Fehlersuche und Fehlerbehandlung	160
Unerwartete Laufzeitfehler	165
Fehlersuche	165
Der Unterbrechungsmodus	166
Das Direktfenster	169
Das Lokal-Fenster	171
Arbeiten mit Lesezeichen	175
Kompilieren des Programmcodes	175
Compilerdirektiven	176
Datenbankverweise	179
Herstellen eines Verweises	179
Zugriff auf eine referenzierte Datenbank	182
Bearbeiten einer referenzierten Datenbank	184
Ereignisorientierte Programmierung	185
Standardereignisse	187
Ereignisprozeduren mit Parametern	188
Verknüpfen von Funktionen mit Ereignissen	191





Programmieren mit Objekten	Kapitel 4	193
Was ist ein Objekt?		194
Die Access-Objekthierarchie		195
Ausführen von Makroaktionen (<i>DoCmd</i> -Objekt)		198
Bezugnahme auf Formulare (<i>Form</i> -Objekte)		198
Bezugnahme auf Berichte (<i>Report</i> -Objekte)		203
Bezugnahme auf Steuerelemente (<i>Control</i> -Objekte)		204
Das <i>Screen</i> -Objekt		206
Das <i>CurrentProject</i> -Objekt		207
Das <i>CurrentData</i> -Objekt		208
Zugriff auf Eigenschaften und Methoden		208
Angaben für Eigenschaften		210
Methoden		211
Die <i>With</i> -Anweisung		212
Objektvariablen		213
Spezielle Objektdatentypen		215
Auflistungen		217
Die <i>For Each...Next</i> -Schleife		219
Der Objektkatalog		220
Objekte begutachten		221
Elemente suchen		223
Datenzugriff mit VBA	Kapitel 5	225
Schnellzugriff auf Daten		226
Datenzugriff mit den DAO-Objekten		229
<i>Database</i> -Objekte		231
<i>Recordset</i> -Objekte		233
Fehler beim Datenzugriff		262
Datenzugriff mit ADO		263
Das Objektmodell		264
Auf Daten zugreifen		265
Zugriff auf Datensätze		270
Fehler beim Datenzugriff		283





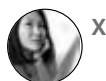
Einführung in SQL	284
Auswahlabfragen definieren	286
Tabellenerstellungsabfragen definieren	300
Löschabfragen definieren	301
Anfügeabfragen definieren	302
Aktualisierungsabfragen definieren	303
Kreuztabellenabfragen definieren	303
Übersicht der Abfragetypen	304
Das Datumsformat	305
 Formulare programmieren	 Kapitel 6 — 307
Formulare öffnen	308
Ansicht und Datenmodus festlegen	310
Filter und Bedingungen	311
Kontrollieren des Öffnens von Formularen	313
Schließen von Formularen	315
Kontrollieren des Schließens von Formularen	317
Suchen nach Datensätzen	318
Positionieren auf Datensätze	320
Suchen nach weiteren Datensätzen	322
Die Prozedur <i>FormularDatensatzAuswählen</i>	325
Navigieren in Formularen	327
Kontrolle der Datensatznavigation	330
Neueingabe von Datensätzen	332
Das <i>VorEingabe</i> -Ereignis	333
Das <i>NachEingabe</i> -Ereignis	334
Löschen von Datensätzen	334
Löschen über Menübefehle	335
Deaktivieren der Warnmeldung	336
Datensatz aus der Datenquelle löschen	337
Kontrolle des Löschvorgangs	340
Kontrolle der Dateneingabe	343
<i>VorAktualisierung</i> - und <i>NachAktualisierung</i> -Ereignisse	343





Filtern und Sortieren	344
Arbeiten mit formularbasierten Filtern	347
Filter speichern	350
Dialogfenster	353
Öffnen von Dialogfenstern	354
Informationen an Dialogfenster übergeben	358
Positionierung, Größenänderung und Aktivierung von Formularen	361
Kontrolle der Größenänderung	363
Aktivierung und Deaktivierung von Formularen	365
RollUp-Formulare	366
Zeitgeberprogrammierung	367
Abfangen von Formularfehlern	368

Steuerelemente programmieren	Kapitel 7	371
Textfelder		372
Überwachen der Dateneingabe		373
Kontrolle des Fokus		380
Optionsfelder, Kontrollkästchen und Umschaltflächen		381
Ereignisse von Kontrollkästchen		383
Umschaltflächen		384
Optionsfelder		384
Befehlsschaltflächen		388
Listen- und Kombinationsfelder		389
Auswahl von Listeneinträgen		390
Zugriff auf Spalten		392
Listenfelder mit Wertelisten füllen		394
Listenfelder mit einer Callback-Funktion füllen		397
Listenfelder mit Mehrfachauswahl		401
Kombinationsfelder		403





Seitenumbrüche	406
Register	408
Register nachträglich einsetzen	410
Hyperlinks	411
Hyperlinks einer Tabelle	411
Hyperlinks mit Bezeichnungsfeldern, Befehlsschaltflächen und <i>Bild</i> -Steuerelementen	413
Berichte programmieren	Kapitel 8 — 417
Berichte öffnen, drucken und schließen	418
Drucken einzelner Datensätze	420
Beeinflussung des Layouts	421
Datensatzwechsel	424
Rechnen in Berichten	425
Zeichnen auf Berichten	428
Auswahl der Linienart	430
Zeichnen von Kreisen und Punkten	432
Positionieren des Grafikcursors	435
Textausgabe	436
Anpassen des Koordinatensystems	439
Objektorientierte Programmierung	Kapitel 9 — 443
Klassen in der Praxis	445
Einsatz von <i>Property</i> -Prozeduren	447
Einsatz des Klassenmoduls	449
Eigenschaften ohne Member-Variablen	451
Definition von Methoden	452
Initialisierung und Terminierung	454
Einsatz des <i>Collection</i> -Objekts	456
Formulare als Objektklassen	459
Methoden und Eigenschaften in Formularmodulen	459
Anlegen von Formularinstanzen	461



**Design von Datenbankobjekten****Kapitel 10 — 469**

Datenbanken designen	471
Datenbank erzeugen	473
Tabellen designen	473
Tabellen erzeugen	474
Felder erzeugen	478
Tabellen und Felder löschen	484
Indizes erzeugen	485
Indizes löschen	488
Beziehungen definieren	489
Beziehung löschen	492
Design von Abfragen	493
Felder einer Abfrage ermitteln	494
Parameterabfragen handhaben	495
Tabellen und Indizes erzeugen	497
Tabellen verändern	502
Indizes erzeugen	504
Tabellen und Indizes löschen	505
Ermitteln eines Abfragetyps	507
Abfragen ausführen	508
Formulare und Berichte designen	510
Formulare erzeugen	510
Steuerelemente erstellen	512
Optionsgruppen manipulieren	515
Steuerelemente löschen	521
Berichte erzeugen	521
Gruppierungen aktivieren	522
Berichtssteuerelement erzeugen	526
Entwicklungstool	526
Zugriff auf Module	531
Module auflisten	532
Module erzeugen	535
Programmcode in ein Modul einfügen	537
Programmcode verändern	539



Vorgehensweise bei der Anwendungsentwicklung	Kapitel 11	541
Der Entwicklungszyklus		542
Relationale Datenbanksysteme		545
Das Entity-Relationship-Modell		546
Richtlinien für Namenskonventionen		548
Besonderheiten beim Entwurf der Struktur		552
Tabellenstrukturen		553
Spezielle Beziehungstypen		557
Gesamtstruktur		559
Replikation von Datenbanken		562
Anlegen einer Replikatgruppe		563
Synchronisieren von Replikaten		564
Sicherheitskopien erzeugen		566
Replikation mit VBA		566
Formulare und Berichte		570
Planen der Formulare und Berichte		570
Erstellen der entsprechenden Abfragen		571
Anlegen der Formulare und Berichte		571
Einrichten der Benutzeroberfläche		572
Verwalten von Symbolleisten		573
Selbstdefinierte Symbolleisten		576
Anlegen neuer Menüleisten		583
Einsatz von Symbol- und Menüleisten		585
Kontextmenüs		587
Der Übersichtsmanager		589
Zugriff auf Optionen		589
Datenbanken verwalten		592
Datenbanken reparieren und komprimieren		592
Datenpflege während der Laufzeit		593
Zusätzliche Befehlszeilenoptionen		598
Benutzerdefinierte Hilfe		599
Die Struktur des Hilfesystems		600
Implementierung in Access		601





Das Sicherheitssystem	602
Zugriff auf das Sicherheitssystem	Kapitel 12 — 603
Zugriff auf Benutzer und Benutzergruppen	605
Anlegen neuer Benutzer und Benutzergruppen	608
Aktivierung des Anmeldeverfahrens	611
Anlegen neuer Benutzergruppen	612
Benutzer und Benutzergruppen löschen	612
Zuordnung von Benutzern und Gruppen	613
<i>Container- und Document-Objekte</i>	616
Erteilen von Zugriffsberechtigungen	618
Eigentümerrechte	621
Sichern einer Datenbank	622
OLE und DDE	Kapitel 13 — 627
OLE – Verknüpfen und Einbetten	628
Einfügen von Objekten	630
Verknüpfen von Objekten	633
Die Vor-Ort-Aktivierung	635
Objektfelder	636
Arbeiten mit gebundenen Objektfeldern	638
Gebundene Objektfelder von VBA aus steuern	640
Automation	654
Automationsobjekte erstellen	655
Verweise zu OLE-Servern	660
Automationsobjekte aus Dateien erstellen	663
Access per Automation steuern	665
ActiveX-Steuerelemente	667
ActiveX-Steuerelemente einfügen	669
Eigenschaften von ActiveX-Steuerelementen	670
Ereigniseigenschaften von ActiveX-Steuerelementen	672
Gebundene ActiveX-Steuerelemente	672





DDE	673
Access als DDE-Client	674
Access als DDE-Server	681
Den Funktionsumfang von Access erweitern	Kapitel 14 — 689
Die Windows-Komponenten	690
Was ist das Windows-API?	691
Das Windows-API unter Access nutzen	694
Tipps zur Erleichterung der Programmierung	694
Nützliche API-Funktionen finden	695
Verwendung der Funktion erlernen	695
API-Funktion deklarieren	697
API-Funktionen einsetzen	698
Änderungen sichern	698
Anwendung testen	699
Unterschiede zwischen Windows-Versionen	699
Ein Beispiel	699
Parameterübergabe an API-Funktionen	700
Referenz- und Werteparameter	701
String-Parameter	702
Typfreie Parameter	706
Typsichere API-Funktionen	707
Benutzerdefinierte Datentypen (Strukturen) und Arrays	709
Zusammenfassung	710
Das API unter Access nutzen	710
Fenstermanipulationen	711
SubClasses eines Fensters	719
Grafikprogrammierung	723
Bildlaufleisten	727
Mausereignisse und Tastatur	730
Windows Registry	732
Index	737





Vorwort





Das erfolgreichste relationale Datenbanksystem für das Betriebssystem Windows, Microsoft Access, ist zusätzlich zur Verwaltung von Daten bestens für die Programmierung von Datenbankanwendungen geeignet. Mit dem vorliegenden Buch erlernen Sie die Programmierung mit Makros und VBA schnell und effektiv. Das Buch berücksichtigt dabei alle Neuerungen der aktuellen Version XP von Access.

Nachdem Sie sich im ersten Teil des Buches die Grundlagen der Programmierung erarbeitet haben, erfahren Sie im zweiten Teil alles Notwendige über die Programmierung von Tabellen, Abfragen, Formularen und Berichten.

Aber auch bei wachsenden Ansprüchen lässt Sie dieses Buch nicht im Stich. Ein eigenes Kapitel beschreibt, wie Sie am besten bei der Entwicklung professioneller Anwendungen vorgehen. Weitere Kapitel erläutern die Handhabung des Sicherheitssystems von Access aus der Perspektive des Programmierers sowie die Techniken zur Entwicklung von Anwendungen, die auf Microsoft Office XP basieren. Hierzu benötigen Sie genaue Kenntnisse über OLE und DDE.

Neben Tom Jordan und Bruno Jennrich haben Martin Hoffmann und Stefan Ditrach entscheidend zum Gelingen dieses Buches beigetragen. Mit der tatkräftigen Unterstützung von Stefan Truckenmüller haben wir alle Neuerungen von Access XP in das Buch eingearbeitet.

Damit wir Ihr Access-XP-Programmierbuch auch in Zukunft kontinuierlich verbessern können, teilen Sie uns bitte Ihre Meinung mit. Wenden Sie sich bitte per Mail an manfred.hoffbauer@fullaccess.de, senden Sie ein Fax an 02 11/1 71 93 58.

Manfred Hoffbauer und Christoph Spielmann im April 2001





Hinweise zum Aufbau des Buches





Das Access-XP-Programmierbuch führt Sie schrittweise in die Programmierung mit VBA ein. Um von diesem Buch zu profitieren, sollten Sie sich mit der Benutzeroberfläche von Access gut auskennen. Sie sollten bereits intensiv mit Tabellen, Abfragen, Formularen und Berichten gearbeitet und diese vor allem schon entworfen haben. Wenn das nicht der Fall ist, dann sollten Sie über die Anschaffung eines geeigneten Buches wie beispielsweise des Access-XP-Buches nachdenken, das ebenfalls im Sybex-Verlag erschienen ist.

Das Access-XP-Programmierbuch ist in vier Teile gegliedert, die im Folgenden kurz vorgestellt werden.

Grundlagen

Für Einsteiger in die Welt der Programmierung sind zunächst die ersten beiden Teile interessant. Sie sind didaktisch und problemorientiert aufgebaut und führen anhand zahlreicher Beispiele in die Welt der Programmierung mit VBA ein.

Auch wenn Sie bereits mit anderen Programmiersprachen vertraut sind, sollten Sie die *Kapitel 1 Einführung in die Makroprogrammierung* und *Kapitel 2 Schnellstart VBA* lesen. Bezüglich *Kapitel 3 Die Programmiersprache VBA* genügt es dann, wenn Sie dieses grob überfliegen, um die Syntax von VBA kennen zu lernen. *Kapitel 4 Programmieren mit Objekten* ist hingegen geradezu unentbehrlich, denn ohne das Wissen über die Programmierung mit Objekten dürfte Ihnen der Umstieg auf VBA sehr schwer fallen.

Programmierpraxis

Nachdem Sie sich die Grundlagen der Programmiersprache VBA erarbeitet haben, geht es im zweiten Teil des Buches an die Praxis. Die Kapitel 5 bis 9 spiegeln aus der Sicht von VBA wider, was Sie bereits von der Benutzeroberfläche her kennen: die Arbeit mit Tabellen, Abfragen, Formularen und Berichten. In den Kapiteln 5 bis 8 dreht sich dabei alles um die Handhabung der Datenbankobjekte. In *Kapitel 9 Objektorientierte Programmierung* erfahren Sie dann, wie Sie durch den Einsatz von Klassenmodulen eigene Objekte selbst definieren.

Mit dem Entwurf von Datenbankobjekten mithilfe von VBA beschäftigt sich *Kapitel 10 Design von Datenbankobjekten*. So lassen sich beispielsweise während der Laufzeit neue Schaltflächen in Formularen anordnen oder Steuerelemente farbiger hinterlegen.





Anwendungsentwicklung

Wenn Sie die ersten beiden Teile dieses Buches durchgearbeitet und fleißig geübt haben, kennen Sie sich mit der Programmierung von Access schon recht gut aus. Dann ist der richtige Zeitpunkt gekommen, um sich mit der Gestaltung von Datenbankanwendungen auseinander zu setzen. Das hierfür notwendige Wissen vermitteln die *Kapitel 11 Vorgehensweise bei der Anwendungsentwicklung* und *Kapitel 12 Zugriff auf das Sicherheitssystem* des Buches.

Spezielle Programmiertechniken

Kapitel 13 OLE und DDE und *Kapitel 14 Den Funktionsumfang von Access erweitern* beschreiben echte Profi-Techniken. Hier erfahren Sie alles über die Programmierung von OLE und DDE. Außerdem erfahren Sie einiges über die Windows-Interns und den Einsatz von API-Funktionen mit Access.

Sicherheitshinweis

Wir haben alle in diesem Buch beschriebenen Verfahren und Techniken sowie die auf der CD enthaltene Software nach bestem Ermessen geprüft, getestet und ständig verbessert. Dennoch sollten Sie sich darüber im Klaren sein, dass wir natürlich keinen Beta-Test im Stil der großen Softwarehäuser durchführen können.

Wir übernehmen daher keine Gewährleistung für die Richtigkeit der im Buch beschriebenen Verfahren und Techniken oder der Software auf der beiliegenden CD. Für Verbesserungsvorschläge sind wir jedoch jederzeit dankbar. Bei Problemen oder Fragen erreichen Sie uns über folgende Anschrift:

TriniDat Software Entwicklung GmbH
Herr Manfred Hoffbauer und Herr Christoph Spielmann
Postfach 10 50 43
40041 Düsseldorf

Telefon: 02 11/1 71 93 56

Telefax: 02 11/1 71 93 58

Mail: buch@trinidadat.de

Web: <http://www.trinidadat.de>

Die dem Buch beiliegende CD lässt sich in kürzeren Abständen aktualisieren als das Buch selbst. Die Datei *Liesmich.txt* enthält aktuelle Ergänzungen, die zum Zeitpunkt der Drucklegung des Buches noch nicht berücksichtigt werden konnten.

Außerdem finden Sie im Internetangebot der Autoren aktuelle Informationen zu Access XP und diesem Buch. Schauen Sie unter folgender Adresse nach:

<http://www.fullaccess.de>





Einführung in die Makroprogrammierung



Das erste Makro	3
Editieren von Makros	20
Fehlersuche und Fehlermeldungen	22
Ereignisse	23



1



Access entwickelt sich immer mehr zu dem führenden Datenbankentwicklungswerkzeug für Windows. Mit den flexiblen und leicht zu handhabenden Datenbankobjekten wie Tabellen, Abfragen, Formularen und Berichten lassen sich anspruchsvolle Datenbanken fast im Handumdrehen entwickeln. Für den letzten Schliff sind diese vier Elemente allerdings oft nicht ausreichend. So ist es auf die Dauer etwas unkomfortabel, ein Formular immer vom Datenbankfenster aus zu öffnen. Dieses erweist sich zwar bei der Entwicklung als sehr effektiv; für den reinen Anwender gibt es allerdings eine Vielzahl besserer Lösungsmöglichkeiten.

Um häufig wiederkehrende Aktionen zu automatisieren, ist eine automatische Steuerung erforderlich. In Access kann diese Steuerung mithilfe von Makros erledigt werden.

Makros stellen ein sehr einfaches und leicht zu erlernendes Werkzeug dar, das sich nicht hinter der ausgewachsenen Programmiersprache VBA (*Visual Basic for Applications*) verstecken muss. Auch Anfänger können nach sehr kurzer Einarbeitungszeit schon die ersten Makros erstellen.

Makros vs. VBA



Ein Nachteil darf allerdings nicht verschwiegen werden: Das Anlegen neuer Makros ist zwar schnell erledigt; genauso schnell leidet allerdings auch die Übersichtlichkeit. Wenn sich einmal 20 bis 30 Makros im Datenbankfenster tummeln (für die Sie auch noch vernünftige Namen vergeben müssen), verlieren Sie schnell den Überblick. Mit VBA wirkt sich dieses Problem weit weniger stark aus, da die zu einem Formular gehörenden Funktionen beispielsweise zusammen mit dem Formular und nicht im Datenbankfenster abgelegt werden.

Als Fazit lässt sich Folgendes festhalten: Makros bieten einen sehr leichten und schnellen Einstieg in die Programmierung von Access. Bei größeren Projekten oder kniffligen Aufgaben stoßen sie aber schnell an ihre Grenzen. VBA ist zwar etwas schwerer zu erlernen, erweist sich aber in kurzer Zeit als komfortabler und angenehmer als Makros. Wenn Sie VBA einmal kennen, werden Sie in Zukunft wahrscheinlich kaum mehr ein Makro anlegen wollen.

Im Folgenden lernen Sie die Techniken zum Erstellen eines Makros zusammen mit ihrem praktischen Einsatz kennen. Darüber hinaus werden die grundlegenden Philosophien der Programmierung von Access erläutert. Sie haben damit das notwendige Rüstzeug, um im *Kapitel 2 Schnellstart VBA* mit sehr geringem Aufwand auf VBA umzusteigen. Dieses Kapitel vermittelt Ihnen das notwendige Wissen, um Makros schnell und unproblematisch in VBA umsetzen zu können.

Auch wenn Sie völlig auf die Programmierung von Makros verzichten und direkt in VBA einsteigen möchten, sollten Sie dieses Kapitel durcharbeiten, da Sie hier neben Grundbegriffen auch das Prinzip der ereignisorientierten Programmierung kennen lernen.



Das erste Makro

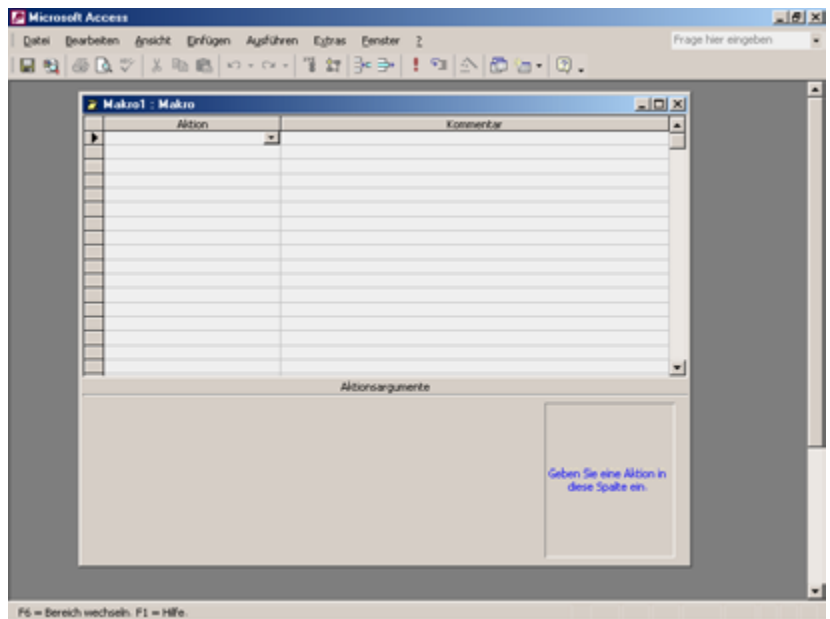
Ein Makro ist nichts anderes als eine Liste von Aktionen, die Access nacheinander ausführt. Eine Aktion ist hierbei ein bestimmter Befehl wie beispielsweise das Öffnen oder Schließen eines Formulars oder die Ausgabe eines Signaltons.

Wie für alle Objekttypen stellt Access auch für Makros eine eigene Kategorie im Datenbankfenster zur Verfügung. Um ein neues Makro anzulegen, wechseln Sie in die Kategorie *Makros* und klicken anschließend auf die Schaltfläche *Neu*. Access öffnet daraufhin das so genannte *Makrofenster*.



Abb. 1.1

Das Makrofenster von
Access



Das Makrofenster ist eine Tabelle, in der mehrere Aktionen untereinander angeordnet werden können. Die Tabelle besteht zu diesem Zweck aus den zwei Spalten *Aktion* und *Kommentar*, wobei die einzelnen Aktionen in der ersten Spalte angegeben werden. Zur Verbesserung der Übersichtlichkeit, können Sie zu jeder Aktion einen beliebigen Text als Kommentar eingeben.

Das Makrofenster

Eine Zeile des Makrofensters, die jeweils aus einer Aktion und einem Kommentar besteht, wird auch als *Aktionszeile* bezeichnet.



Direkt nachdem Access das Makrofenster geöffnet hat, blinkt in der ersten Zeile der Spalte *Aktion* die Einfügemarke. Gleichzeitig wird am rechten Spaltenrand die Schaltfläche eines Kombinationsfeldes angezeigt. Wenn Sie nun auf diese Schaltfläche klicken, öffnet Access eine Liste mit allen 53 Aktionen, die zur Programmierung von Makros verwendet werden können. Aus dieser Liste können Sie nun beispielsweise die Aktion mit dem Namen *Meldung* auswählen, die im Folgenden näher erläutert wird.

Sobald Sie die *Meldung*-Aktion ausgewählt haben, zeigt Access im rechten unteren Bereich des Makrofensters einen blauen Text an, der kurz die Aufgabe dieser Aktion beschreibt. Wie Sie dem Text entnehmen können, bringt sie ein Meldungs Fenster auf den Bildschirm.



Abb. 1.2

Ein Meldungs Fenster

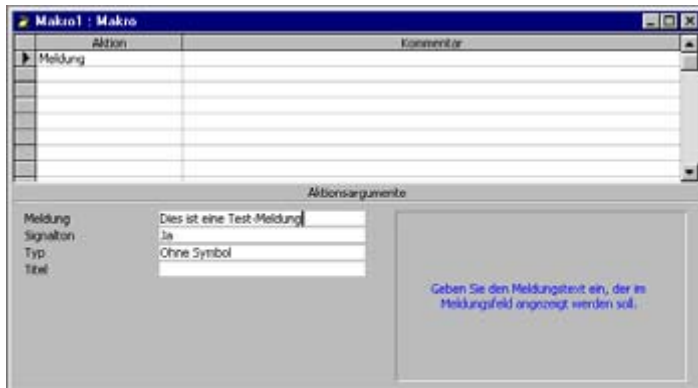
Ein Meldungs Fenster ist dafür vorgesehen, dem Anwender eine bestimmte Information mitzuteilen. Zu diesem Zweck kann es mit einem beliebigen Text versehen werden. Wie aber teilen Sie der *Meldung*-Aktion mit, welcher Text angezeigt werden soll?

Eingabe der
Meldung

Zur Eingabe diese Textes befindet sich im unteren Teil des Makrofensters ein Textfeld mit der Beschriftung *Meldung*. Zusammen mit den anderen drei Eingabefeldern stellen diese die so genannten *Aktionsargumente* der *Meldung*-Aktion dar. Geben Sie in das Eingabefeld *Meldung* einfach einmal den Text "Dies ist eine Test-Meldung" ein.

Abb. 1.3

Die Aktionsargumente
der *Meldung*-Aktion

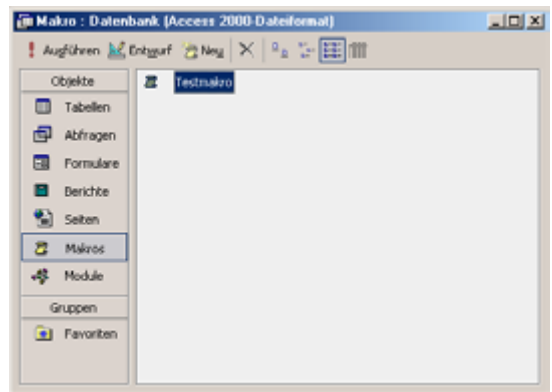


Ihr erstes Makro ist damit bereits fast fertig. Was nun noch fehlt, ist ein passender Name. Schließen Sie das Makro, indem Sie entweder den Menüpunkt *Datei > Schließen* auswählen oder auf die *Schließen*-Schaltfläche (X) im Titel des Fensters klicken. Access öffnet daraufhin ein Meldungsfenster, in dem Sie entscheiden können, ob Sie das Makro dauerhaft speichern möchten. Klicken Sie hier auf *Ja*. Anschließend erscheint ein Dialogfenster, in dem Sie den Namen des Makros eingeben können. Geben Sie Ihrem Makro den Namen *Testmakro* und bestätigen Sie mit der *OK*-Schaltfläche. Im Datenbankfenster wird das neue Makro nun mit seinem Namen aufgelistet.

Speichern des Makros

Abb. 1.4

Das Makro *Testmakro* im Datenbankfenster



Was Sie bis jetzt erledigt haben, war lediglich die Programmierung des Testmakros. Damit die Meldung nun auch auf Ihrem Bildschirm erscheint, muss das Makro gestartet bzw. ausgeführt werden. Auch hierzu gibt es mehrere Möglichkeiten. Die einfachste besteht darin, dass Sie den Namen des Makros im Datenbankfenster doppelt mit der Maus anklicken. Nachdem Sie dies ausgeführt haben, erscheint auf Ihrem Bildschirm das Meldungsfenster mit dem vorher definierten Text. Wenn Sie anschließend die *OK*-Schaltfläche des Meldungsfensters betätigen, verschwindet das Fenster vom Bildschirm, und das Makro ist beendet.

Eine andere Möglichkeit zum Start des Makros stellt der Menüpunkt *Extras > Makro > Makro ausführen* zur Verfügung. Dieser Menüpunkt aktiviert ein Dialogfenster, in dem Sie ein bestimmtes Makro über ein Kombinationsfeld auswählen können. Ein Klick auf *OK* startet das ausgewählte Makro.

Abb. 1.5

Das Dialogfenster *Makro ausführen*



Bearbeiten des Makros

Um Veränderungen an dem Makro vorzunehmen, öffnen Sie es wieder in der Entwurfsansicht. Dazu markieren Sie es im Datenbankfenster und klicken auf *Entwurf*. Alternativ können Sie auch bei gedrückter **Strg**-Taste doppelt auf den Makronamen klicken oder den Menüpunkt *Entwurf* des Kontextmenüs auswählen. Access öffnet daraufhin wieder das Makrofenster.

Aktionsargumente der Meldung-Aktion

Im unteren Teil des Makrofensters finden Sie unterhalb des *Meldung*-Aktionsarguments außerdem noch drei weitere Aktionsargumente mit den Namen *Signalton*, *Typ* und *Titel*. Die folgende Tabelle enthält eine Erklärung aller vier Argumente:

Die Aktionsargumente der *Meldung*-Aktion

Aktionsargument	Beschreibung
<i>Meldung</i>	Ein Text, der innerhalb des Meldungsfensters angezeigt wird.
<i>Signalton</i>	Dieses Argument bestimmt, ob kurz vor dem Anzeigen ein Signalton ertönt, der den Anwender auf das Meldungsfenster aufmerksam macht. Geben Sie hier entweder <i>Ja</i> oder <i>Nein</i> an.
<i>Typ</i>	Wählt ein Symbol aus, das zusätzlich zum Text in dem Meldungsfenster angezeigt wird. Die möglichen Einstellungen entnehmen Sie der folgenden <i>Tabelle 1.2</i> .
<i>Titel</i>	Legt den Text fest, der im Titel des Meldungsfensters erscheint. Wenn Sie hier nichts angeben, erscheint der Text "Microsoft Access".

Tabelle 1.1

Typ

Über das dritte Argument mit dem Namen *Typ* kann ein Symbol bestimmt werden, das Access im Meldungsfenster links vom Text anzeigt. Die folgende Abbildung zeigt, wie so etwas aussehen kann.

Abb. 1.6

Ein Meldungsfenster mit einem Symbol



Einige Aktionsargumente – zu denen auch das *Typ*-Aktionsargument gehört – erlauben die Wahl zwischen verschiedenen vordefinierten Einstellungen. Wenn Sie die Einfügemarke in das Eingabefeld des *Typ*-Arguments positionieren, erscheint am rechten Rand die Schaltfläche eines Kombinationsfeldes. Ein Klick auf diese Schaltfläche aktiviert eine Liste mit den folgenden vier Einstellungen:

Mögliche Einstellungen des Aktionsarguments *Typ*

Einstellung	Symbol
Ohne Symbol	Es wird kein Symbol angezeigt.
Kritisch	
Warnmeldung (?)	
Warnmeldung (!)	
Information	

Tabelle 1.2

Durch Auswahl einer dieser vier Einstellungen können Sie das Symbol festlegen, das bei der Ausführung der Aktion im Meldungsfenster angezeigt wird.

Grundsätzlich haben Sie mithilfe der vier Aktionsargumente die Möglichkeit, das Meldungsfenster nach Ihren individuellen Vorstellungen zu konfigurieren. Experimentieren Sie einfach einmal etwas mit den Einstellungen herum, und begutachten Sie das Ergebnis bei der Ausführung des Makros!

In dem jetzigen Stadium des Testmakros wird nur eine einzelne Aktion, nämlich die *Meldung*-Aktion ausgeführt. Makros erlauben jedoch auch die schrittweise Ausführung mehrerer hintereinander folgender Aktionen. Um dies einmal auszuprobieren, öffnen Sie das Testmakro wieder zur Bearbeitung in der Entwurfsansicht. Platzieren Sie die Einfügemarke in die zweite Aktionszeile direkt unter die *Meldung*-Aktion. Mittels der Makroliste wählen Sie nun erneut die *Meldung*-Aktion aus, so dass anschließend zwei *Meldung*-Aktionen direkt untereinander angeordnet sind.

Im unteren Bereich des Makrofensters werden wieder die vier Aktionsargumente der *Meldung*-Aktion angezeigt. Diese beziehen sich nun aber nicht auf die erste, sondern die zweite *Meldung*-Aktion, so dass Sie hier erneut vier individuelle Einstellungen vornehmen können. Zum Test geben Sie unter dem Aktionsargument *Meldung* den Text "Dies ist die zweite Meldung" ein.

Ausführung mehrerer Aktionen





Um nun nachträglich noch den Text der ersten Meldung anzupassen, klicken Sie einfach auf den Namen der ersten Aktion. Access positioniert dadurch die Einfügemarke in die erste Aktionszeile und zeigt im unteren Bereich des Makrofensters die Aktionsargumente dieser ersten *Meldung*-Aktion an. Durch das Anklicken einer Aktionszeile machen Sie diese zur aktiven bzw. aktuellen Aktionszeile. Die aktive Aktionszeile erkennen Sie daran, dass Access links neben der Zeile einen kleinen schwarzen Pfeil anzeigt. Die Aktionsargumente im unteren Bereich des Makrofensters beziehen sich stets auf die Aktion der aktiven Aktionszeile.



Hinweis

Das Verschieben der Einfügemarke und damit das Festlegen der aktiven Aktionszeile können Sie auch über die Pfeiltasten Ihrer Tastatur bewerkstelligen.

Zur Änderung des Meldungstextes ersetzen Sie den Inhalt des *Meldung*-Aktionsarguments durch den Text "Dies ist die erste Meldung". Sie besitzen nun zwei Meldungsaktionen, wobei die erste den Text "Dies ist die erste Meldung" und die zweite den Text "Dies ist die zweite Meldung" anzeigt.



Zum Test des Makros klicken Sie auf das *Ausführen*-Symbol der Symbolleiste. Bevor Access das Makro ausführen kann, werden Sie über eine Meldung allerdings noch zum Speichern aufgefordert. Dies ist notwendig, da Access grundsätzlich nur gespeicherte Makros ausführen kann. Nachdem Sie diese Meldung mit *OK* bestätigt haben, erscheint das erste Meldungsfenster mit dem Text "Dies ist die erste Meldung". Sobald Sie die Meldung mit *OK* quittiert haben, zeigt Access die zweite Meldung mit dem Text "Dies ist die zweite Meldung" an. Wird diese ebenfalls per *OK* bestätigt, ist die Ausführung des Makros beendet.

An dem Testmakro werden nun zwei Dinge deutlich: Access führt die Aktionen eines Makros nacheinander von oben nach unten aus. Es kommt also wesentlich auf die Reihenfolge an, in der Sie Ihre Aktionen im Makrofenster anordnen. Der andere Punkt ist, dass Access die Ausführung eines Makros unterbricht, bis die betreffende Aktion vollständig erledigt ist. Access wartet also mit der Ausführung der zweiten Aktion so lange, bis Sie das erste Meldungsfenster mit *OK* schließen und die erste Aktion damit abgeschlossen ist.

Die *ÖffnenFormular*-
Aktion

Eine der wichtigsten Aktionen ist die Aktion *ÖffnenFormular*. Wie der Name schon vermuten lässt, kann mithilfe dieser Aktion ein beliebiges Formular geöffnet werden. Um das folgende Beispiel nachzuvollziehen, sollten Sie die Beispieldatenbank *Kapitel1.mdb* öffnen. Diese Datenbank enthält ein Formular mit dem Namen *Adressen*, das auf der gleichnamigen Tabelle basiert.

Über die Schaltfläche *Entwurf* des Datenbankfensters legen Sie ein neues Makro an. In der ersten Aktionszeile wählen Sie jetzt allerdings nicht die Aktion *Meldung*, sondern die Aktion *ÖffnenFormular* aus, woraufhin Access im unteren Bereich des Makrofensters wieder die Aktionsargumente anzeigt. Da die *ÖffnenFormular*-Aktion eine völlig andere Aufgabe hat als die *Meldung*-Aktion, besitzt sie dementsprechend auch andere Aktionsargumente.



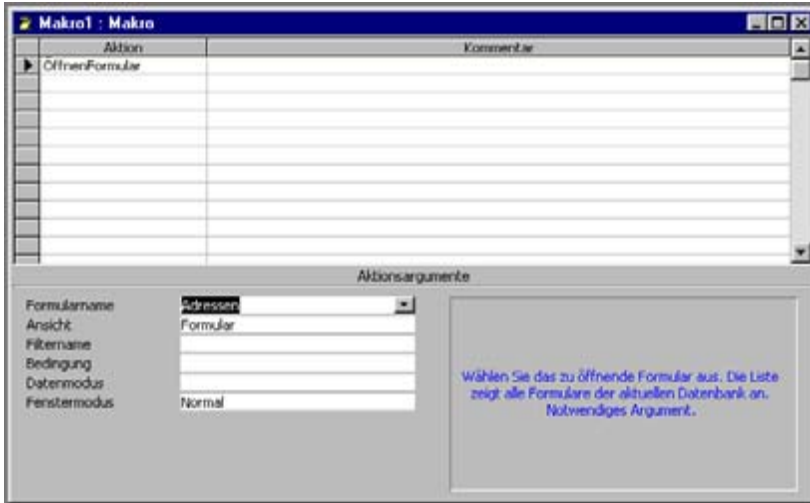


Abb. 1.7

Die *ÖffnenFormular*-Aktion

Von besonderem Interesse sind die ersten beiden Aktionsargumente mit den Namen *Formularname* und *Ansicht*. Über das erste Argument geben Sie den Namen des Formulars an, das Access öffnen soll. Sobald Sie die Einfügemarke in das Eingabefeld dieser Aktion positionieren, erscheint am rechten Rand eine Schaltfläche, über die Sie eine Liste aller Formulare aktivieren, die in der aktuellen Datenbank enthalten sind. Wählen Sie hier das gewünschte Formular – in diesem Beispiel also *Adressen* – aus. Das zweite Aktionsargument bestimmt, in welcher Ansicht das Formular geöffnet werden soll. Hier haben Sie – ebenfalls über die Liste eines Kombinationsfeldes – die Auswahl zwischen den Ansichten *Formular*, *Entwurf*, *Seitenansicht* und *Datenblatt*. Um das Formular *Adressen* in der Formularansicht zu öffnen, wählen Sie die Einstellung *Formular* aus.

Das fertige Makro speichern Sie unter dem Namen *Adressenformular öffnen*. Dies erledigen Sie entweder über den Menüpunkt *Datei > Speichern* oder über das Symbol *Speichern* der Symbolleiste. Wenn Sie das Makro nun ausführen, öffnet Access selbstständig das Adressenformular.



Beachten Sie, dass in der Beispieldatenbank *Kapitel1.mdb* bereits ein Makro mit dem Namen *Adressenformular öffnen* vorhanden ist. Zum Speichern Ihres Makros müssen Sie ihm entweder einen anderen Namen geben oder das vorhandene Makro vorher löschen.



Hinweis

Übersicht über alle Aktionen

Neben den beiden vorgestellten Aktionen *Meldung* und *ÖffnenFormular* stellt Access noch weitere 51 Aktionen zur Verfügung, die jeweils eine spezielle Aufgabe haben. Wenn Sie beispielsweise kein Formular, sondern einen Bericht öffnen (oder ausdrucken) möchten, verwenden Sie anstelle der *ÖffnenFormular*-Aktion die *ÖffnenBericht*-Aktion.





Die folgende Tabelle enthält eine Übersicht über alle 53 Aktionen, die Access zur Verfügung stellt. Den praktischen Einsatz eines Großteils dieser Aktionen werden Sie im Zusammenhang mit der VBA-Programmierung in den *Kapitel 6 Formulare programmieren Kapitel 7 Steuerelemente programmieren* und *Kapitel 8 Berichte programmieren* kennen lernen. Eine ausführlichere Erklärung finden Sie in der auf der CD zum Buch enthaltenen Referenz.

Übersicht über alle Aktionen



Aktion	Erklärung
<i>AbbrechenEreignis</i>	Bricht das Ereignis ab, das zum Start des Makros führte.
<i>AktualisierenDaten</i>	Aktualisiert die Daten eines Steuerelements, die auf Tabellen oder Abfragen basieren.
<i>AktualisierenObjekt</i>	Aktualisiert ein Datenbankobjekt.
<i>AnwendenFilter</i>	Filtert die Datensätze eines Formulars oder Berichts.
<i>AnzeigenAlleDatensätze</i>	Entfernt alle Filter des aktiven Formulars.
<i>AusführenAnwendung</i>	Startet eine DOS- oder Windows-Anwendung.
<i>AusführenBefehl</i>	Führt einen Menübefehl aus. Diese Aktion ersetzt zusammen mit den neuen Menü- und Symbolleistenfunktionen die in Vorgängerversionen verwendete Aktion <i>AusführenMenübefehl</i> .
<i>AusführenCode</i>	Führt eine VBA-Function-Prozedur aus.
<i>AusführenMakro</i>	Führt ein Makro aus.
<i>AusführenSQL</i>	Führt eine SQL-Aktionsabfrage aus.
<i>Ausgabeln</i>	Exportiert ein Objekt in das HTML-, ASP-, IIS-, Rich-Text-, Excel- oder Textformat.
<i>AuswählenObjekt</i>	Wählt ein Datenbankobjekt aus.
<i>Drucken</i>	Druckt Datenblätter, Formulare oder Berichte aus.
<i>Echo</i>	Bestimmt, ob Access den Bildschirm während der Makroausführung aktualisiert.
<i>EinblendenSymbolleiste</i>	Aktiviert oder deaktiviert eine Access-Symbolleiste.



Übersicht über alle Aktionen (Forts.)

Aktion	Erklärung
<i>GeheZuDatensatz</i>	Macht einen bestimmten Datensatz zum aktuellen Datensatz.
<i>GeheZuSeite</i>	Bewegt den Fokus des aktiven Formulars auf das erste Feld einer bestimmten Seite.
<i>GeheZuSteuerelement</i>	Gibt einem bestimmten Feld oder Steuerelement den Fokus.
<i>HinzufügenMenü</i>	Erstellt einen neuen Hauptmenüpunkt für ein benutzerdefiniertes Menü.
<i>KopierenObjekt</i>	Kopiert ein beliebiges Objekt des Datenbankfensters.
<i>LöschenObjekt</i>	Löscht ein Objekt des Datenbankfensters.
<i>Maximieren</i>	Bringt das aktive Fenster auf Vollbildgröße.
<i>Meldung</i>	Zeigt ein Dialogfenster mit einem Text an.
<i>Minimieren</i>	Reduziert das aktive Fenster auf Symbolgröße.
<i>ÖffnenAbfrage</i>	Öffnet eine Abfrage.
<i>ÖffnenBericht</i>	Öffnet einen Bericht.
<i>ÖffnenDatenzugriffsseite</i>	Öffnet eine Datenzugriffsseite in der Browse- oder der Entwurfsansicht.
<i>ÖffnenDiagramm</i>	Öffnet ein Schema.
<i>ÖffnenFormular</i>	Öffnet ein Formular.
<i>ÖffnenGespeicherteProzedur</i>	Öffnet eine gespeicherte Prozedur.
<i>ÖffnenModul</i>	Öffnet ein VBA-Modul.
<i>ÖffnenTabelle</i>	Öffnet eine Tabelle.
<i>Positionieren</i>	Verschiebt die Position oder ändert die Größe des aktiven Fensters.
<i>Sanduhr</i>	Zeigt eine Sanduhr anstelle des Mauszeigers an.
<i>Schließen</i>	Schließt ein Fenster.
<i>SendenObjekt</i>	Verschickt ein Objekt über das Netzwerk.
<i>SetzenMenüelement</i>	Deaktiviert einen Menüpunkt oder markiert ihn mit einem Häkchen.



Übersicht über alle Aktionen (Forts.)

Aktion	Erklärung
<i>SetzenWert</i>	Verändert den Wert eines Feldes, eines Steuerelements oder einer Eigenschaft.
<i>Signalton</i>	Erzeugt einen kurzen Ton über den PC-Lautsprecher.
<i>Speichern</i>	Speichert ein Objekt.
<i>StopAlleMakros</i>	Beendet alle momentan ausgeführten Makros.
<i>StopMakro</i>	Beendet das Makro.
<i>SuchenDatensatz</i>	Sucht einen Datensatz.
<i>SuchenWeiter</i>	Wiederholt die letzte Suche nach Datensätzen.
<i>Tastaturbefehle</i>	Simuliert das Drücken von Tasten.
<i>TransferArbeitsblatt</i>	Tauscht Daten mit einer Tabellenkalkulation aus.
<i>TransferDatenbank</i>	Tauscht Daten zwischen Datenbanken aus.
<i>TransferText</i>	Tauscht Daten mit einem Microsoft SQL Server ab Version 7.0 aus.
<i>TransferText</i>	Im- oder exportiert Textdateien.
<i>UmbenennenObjekt</i>	Benennt das ausgewählte Datenbankobjekt um.
<i>Verlassen</i>	Beendet Access.
<i>Warnmeldungen</i>	Unterdrückt die Ausgabe von Systemmeldungen.
<i>Wiederherstellen</i>	Bringt das aktive Fenster auf seine ursprüngliche Größe.

Tabelle 1.3

Bedingungen



Oft ist es erforderlich, dass eine bestimmte Aktion oder mehrere Aktionen nur dann ausgeführt werden, wenn eine bestimmte Bedingung erfüllt ist. Zu diesem Zweck kann im Makrofenster eine weitere Spalte mit der Spaltenüberschrift *Bedingung* aktiviert werden. Dies bewerkstelligen Sie über den Menüpunkt *Ansicht > Bedingungen* oder das gleichnamige Symbol in der Symbolleiste.



Access versteht unter einer Bedingung einen Ausdruck, der entweder *Ja* oder *Nein* zum Ergebnis hat. Wenn die Bedingung *Ja* zum Ergebnis hat, wird die Aktion ausgeführt; ist das Ergebnis dagegen *Nein*, wird sie nicht ausgeführt. Neben *Ja* und *Nein* kennt Access noch die Begriffe *Wahr* und *Falsch*, die aber die gleiche Bedeutung haben.

Ja und Nein

Bei *Ja* und *Nein* bzw. *Wahr* und *Falsch* handelt es sich um logische Ausdrücke, die zwei unterschiedliche Zustände widerspiegeln. Solche Ausdrücke werden auch als *boolesche Ausdrücke* bezeichnet. Intern interpretiert Access den Zustand *Nein* bzw. *Falsch* mit dem Wert 0. *Ja* und *Wahr* werden dagegen mit einem beliebigen anderen Wert (vorzugsweise ‑1) interpretiert.

Eine einfache Bedingung könnte beispielsweise folgendermaßen aussehen:

5 = 6

Vergleichsoperatoren

Wenn Sie den Ausdruck betrachten, werden Sie zu dem Ergebnis *Falsch* kommen, wogegen die folgende Bedingung

5 = 5

Ihre Zustimmung finden dürfte. Nichts anderes erledigt auch Access: Der Ausdruck wird komplett ausgewertet und erhält schließlich die Bewertung *Wahr* oder *Falsch*. In Abhängigkeit von diesem Ergebnis wird danach eine Aktion ausgeführt oder nicht. Die folgende Tabelle enthält noch eine Reihe weiterer Beispiele für Bedingungen:

Funktionen verschiedener Vergleichsoperatoren

Bedingung	Ergebnis	Bedingung	Ergebnis
5 = 5	Wahr	"Hallo" = "Hallo"	Wahr
5 = 6	Falsch	"Hallo" = "hallo"	Wahr
5 = "Hallo"	Fehler!	"B" > "A"	Wahr
4 > 6	Falsch	"Müller" > "Meier"	Wahr
3 < 4	Wahr	"Maier" < "Mayer"	Wahr
4 >= 4	Wahr	3 > 5 Und 1 > 0	Falsch

Tabelle 1.4



Wie Sie der Tabelle entnehmen können, gibt es neben dem Gleichheitszeichen noch weitere Vergleichsoperatoren. Die folgende Tabelle enthält die wichtigsten:

Wichtige Vergleichsoperatoren

Operator	Beschreibung
=	Gleich
<	Kleiner
>	Größer
<=	Kleiner gleich
>=	Größer gleich
<>	Ungleich

Tabelle 1.5

Eine Voraussetzung für den Einsatz von Vergleichsoperatoren ist, dass der linke Teil auch wirklich mit dem rechten Teil vergleichbar ist. So führt beispielsweise die Bedingung

`5 = "Hallo"`

zu einem Fehler. Wie soll Access schließlich bewerten, ob der numerische Wert 5 dem Text "Hallo" entspricht? Da dieser Fehler während der Ausführung des fehlerhaften Makros auftritt, wird er auch als *Laufzeitfehler* bezeichnet. Wie Sie solche Fehler handhaben und beheben, erfahren Sie im Abschnitt *Fehlersuche und Fehlermeldungen*.

Zeichenketten

Weiterhin wird an dem Beispiel deutlich, dass Texte immer in Anführungszeichen angegeben werden müssen. Weil Texte nichts anderes als eine Kette aufeinander folgender Zeichen sind, werden sie auch als *Zeichenketten* bezeichnet.

Verweis auf Felder und Steuerelemente

Die Bedingungen in den obigen Beispielen sollen lediglich die Funktionsweise einiger Vergleichsoperatoren verdeutlichen. Sinnvolle Ausdrücke für eine Bedingung sind sie jedoch nicht, da das Ergebnis von vornherein feststeht. Interessant wird dies erst, wenn Sie anstelle der Zahlen oder Texte Verweise auf Datenfelder und Steuerelemente angeben. Die genaue Syntax zur Bezugnahme auf Steuerelemente finden Sie in *Kapitel 3 Die Programmiersprache VBA* bei der Besprechung der VBA-Programmierung.

Die folgende Bedingung hat beispielsweise nur dann das Ergebnis *Wahr*, wenn der Inhalt des Steuerelements *Plan Abgabe* auf dem geöffneten Formular *Projekte* kleiner ist als das aktuelle Datum – das Abgabedatum also verstrichen ist:

`Formulare![Projekte]![Plan Abgabe]<Datum()`



Auswirkung einer Bedingung

Wenn Sie in die *Bedingung*-Spalte einer Aktionszeile einen Ausdruck eingegeben haben, der bei der Auswertung den logischen Wert *Wahr* zum Ergebnis hat, führt Access diejenige Aktion aus, die direkt rechts neben der Bedingung in der Spalte *Aktion* angegeben ist. Die Bedingung bezieht sich also immer nur auf eine einzelne Aktionszeile und damit auf eine einzige Aktion. Die folgende Tabelle beschreibt den schematischen Aufbau eines einfachen Makros:

Auswirkung einer Bedingung

Spalte <i>Bedingung</i>	Spalte <i>Aktion</i>
<Bedingung1>	Aktion1
	Aktion2
	Aktion3
	Aktion4
<Bedingung2>	Aktion5

Tabelle 1.6

In dem Beispiel befindet sich vor der ersten und der fünften Aktion jeweils eine Bedingung. Access macht es nun von diesen Bedingungen abhängig, ob die *Aktion1* bzw. die *Aktion5* ausgeführt wird. Da vor den Aktionen 2, 3 und 4 keine Bedingung angegeben ist, werden diese in jedem Fall ausgeführt.

Soll die Ausführung mehrerer Aktionen von einer einzigen Bedingung abhängig gemacht werden, müssen Sie in der *Bedingung*-Spalte vor jeder Aktion drei Punktzeichen angeben. Das folgende Beispiel zeigt dies:

Fortsetzungs-
zeichen

Bedingte Verzweigung

Spalte <i>Bedingung</i>	Spalte <i>Aktion</i>
<Bedingung1>	Aktion1
...	Aktion2
...	Aktion3
	Aktion4
<Bedingung2>	Aktion5

Tabelle 1.7



Hat der Ausdruck *<Bedingung1>* den Wert *Wahr* zum Ergebnis, führt Access die Aktionen 1, 2 und 3 aus. Ist das Ergebnis *Falsch*, werden diese drei Aktionen übersprungen. Unabhängig von der *<Bedingung1>* führt Access die *Aktion4* grundsätzlich immer aus. Die *Aktion5* wird dagegen wiederum nur dann ausgeführt, wenn die *<Bedingung2>* das Ergebnis *Wahr* hat.

Durch die drei Punkte vor den Aktionen 2 und 3 werden diese praktisch mit der *Aktion1* zu einem Aktionsblock zusammengefasst, der nur dann ausgeführt wird, wenn die vorhergehende Bedingung den logischen Wert *Wahr* zum Ergebnis hat. Andernfalls überspringt Access den Aktionsblock.



Hinweis

Zur besseren Strukturierung der Bedingungen können Sie im Makrofenster leere Aktionszeilen einfügen. Diese werden von Access ignoriert und beeinflussen die Ausführung des Makros nicht.

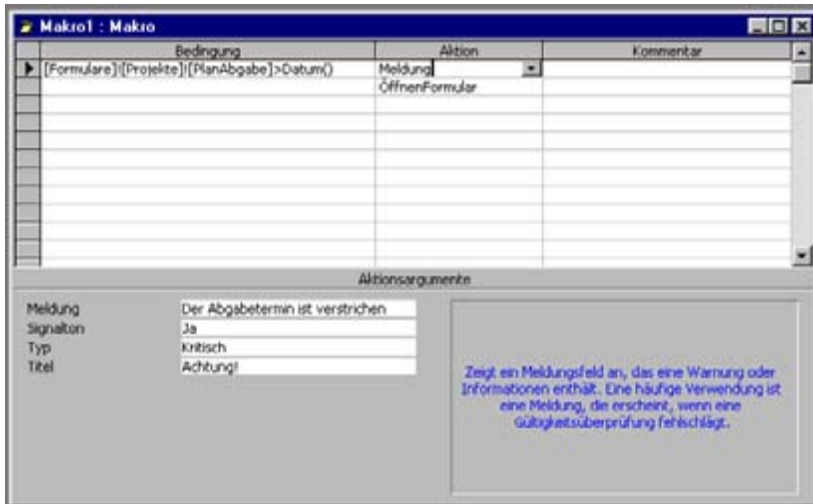
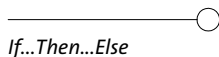


Abb. 1.8

Bedingungen in einem Makro

Im Zusammenhang mit Bedingungen ist die Aktion mit dem Namen *StopMakro* von besonderem Interesse. Wie der Name schon vermuten lässt, wird ein Makro durch diese Aktion unterbrochen.



If...Then...Else

Hat eine Bedingung den logischen Wert *Wahr* zum Ergebnis, werden die entsprechenden Aktionen des nachfolgenden Aktionsblocks ausgeführt. Ist dies nicht der Fall, überspringt Access die betreffenden Aktionen und setzt die Ausführung des Makros fort. Oftmals ist es jedoch erwünscht, dass in diesem Fall – und nur in diesem Fall – ein anderer Aktionsblock ausgeführt wird. Bei der Programmierung in VBA wird Ihnen dieses Verfahren noch als *If...Then...Else*-Konstrukt begegnen.



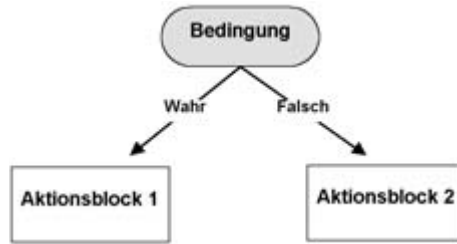


Abb. 1.9

Bedingte Ausführung von zwei Aktionsblöcken

In dem Beispiel der folgenden Tabelle werden die Aktionen 1 bis 3 nur dann ausgeführt, wenn die Bedingung den logischen Wert *Wahr* zum Ergebnis hat. Da auch die vierte Aktion *StopMakro* von der Bedingung abhängig ist, beendet Access das Makro nach der dritten Aktion. Liefert die Bedingung hingegen den logischen Wert *Falsch*, überspringt Access die ersten vier Zeilen und führt ausschließlich die Aktionen 4 und 5 aus.

If...Then...Else-Verzweigung

Spalte <i>Bedingung</i>	Spalte <i>Aktion</i>
<Bedingung>	Aktion1
...	Aktion2
...	Aktion3
...	StopMakro
	Aktion4
	Aktion5

Tabelle 1.8

An diesem Makro wird die Unübersichtlichkeit solcher Konstrukte besonders deutlich. Selbst erfahrene Makroprogrammierer müssen zwei Mal hinsehen, um zu erkennen, dass es sich um eine Verzweigung mit der alternativen Ausführung zweier Aktionsblöcke handelt. Abhilfe schafft hier nur eine gute Kommentierung des Makros über die Spalte *Kommentar* oder die wesentlich übersichtlichere und auch einfachere Programmierung in VBA.

Gruppenmakros

Bei komplexen Anwendungen sammelt sich schnell eine große Anzahl von Makros an, was der Übersichtlichkeit bei der Makroprogrammierung nicht gerade dienlich ist. Aus diesem Grund ist es möglich, mehrere Makros in einer Gruppe zusammen-



zufassen. Streng genommen handelt es sich hierbei um mehrere Untermakros, die in einem Makro – in diesem Fall auch *Gruppenmakro* genannt – angeordnet werden.



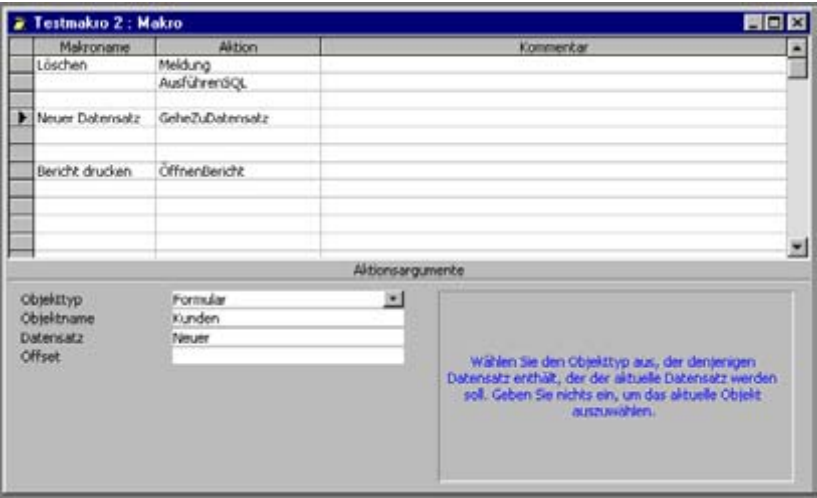
Um ein Gruppenmakro zu erzeugen, erstellen Sie wie gewohnt ein normales Makro und öffnen dieses in der Entwurfsansicht. Als Nächstes wählen Sie den Menüpunkt *Ansicht > Makronamen* aus oder klicken das gleichnamige Symbol in der Symbolleiste an, woraufhin Access eine neue Spalte mit dem Namen *Makroname* einblendet. In diese Spalte können Sie jeweils den Namen eines Untermakros angeben. Sie sollten darauf achten, dass diese Bezeichnung in dem aktuellen Makro nicht doppelt vorkommt: Obwohl dies nicht zu einem Fehler führt, wird beim Aufruf des Untermakros stets nur das erste Untermakro ausgeführt, was in der Regel nicht im Sinne des Makroprogrammierers ist.

Die Spalte *Makro-
name*

Ein Eintrag in die Spalte *Makroname* markiert den Anfang eines Untermakros. Das Ende wird durch das Makroende selbst oder einen weiteren Eintrag in der *Makro-
name*-Spalte gekennzeichnet. Das Makro in *Abbildung 1.10* enthält beispielsweise drei Untermakros mit den Namen *Löschen*, *Neuer Datensatz* und *Bericht drucken*.

Abb. 1.10

Ein Gruppenmakro



Aufruf eines Unter-
makros

Ein Gruppenmakro können Sie jedoch nicht so ohne weiteres über das Symbol *Ausführen* starten, da nicht klar ist, welches Untermakro des Gruppenmakros Access ausführen soll. Der Aufruf eines Untermakros muss daher immer in der Form

<Gruppenmakroname>.<Untermakroname>



erfolgen. Um das Untermakro *Löschen* des Gruppenmakros *Makro1* aus *Abbildung 1.10* auszuführen, müssen Sie zunächst den Menüpunkt *Extras > Makro > Makro ausführen* aufrufen. Es erscheint das Dialogfenster *Makro ausführen*, in dem Sie über das Kombinationsfeld ein beliebiges Untermakro auswählen können. In dem obigen Beispiel muss der folgende Ausdruck verwendet werden:

`Makro1.Löschen`

Wenn Sie ein Gruppenmakro starten, ohne ein spezielles Untermakro anzugeben, führt Access automatisch das erste Untermakro aus. Hierbei gibt es jedoch eine Besonderheit zu beachten: Alle Aktionszeilen vor dem ersten Untermakro werden von Access ebenfalls als Untermakro angesehen. Sollte das erste Untermakro also nicht in der ersten Aktionszeile beginnen, führt Access beim Start des Makros alle vor dem ersten Untermakro angeordneten Aktionen aus. Wenn Sie hier keine Aktionen ausgewählt haben, führt Access also nicht automatisch das erste Untermakro aus, sondern stoppt das Makro. Die folgenden zwei Beispielmakros machen dies noch einmal deutlich:

Ausführung von Untermakros, Beispiel 1

Aktionszeile	Spalte <i>Makronamen</i>	Spalte <i>Aktion</i>
1	<i>Untermakro1</i>	Aktion1
2		Aktion2
3	<i>Untermakro1</i>	Aktion1
4		Aktion2
5		Aktion3

Tabelle 1.9

Dieses Beispielmakro beginnt bereits in der ersten Aktionszeile mit einem Untermakro, das den Namen *Untermakro1* besitzt. Wird das komplette Makro ausgeführt, startet Access automatisch (und ausschließlich) das *Untermakro1*. Das folgende Makro unterscheidet sich von diesem Makro nur darin, dass es mit einer leeren Aktionszeile beginnt:

Ausführung von Untermakros, Beispiel 2

Aktionszeile	Spalte <i>Makronamen</i>	Spalte <i>Aktion</i>
1		
2	<i>Untermakro1</i>	Aktion1
3		Aktion2



Ausführung von Untermakros, Beispiel 2 (Forts.)

Aktionszeile	Spalte <i>Makronamen</i>	Spalte <i>Aktion</i>
4	<i>Untermakro1</i>	Aktion1
5		Aktion2
6		Aktion3

Tabelle 1.10

Wird dieses Makro nun gestartet, trifft Access bei der Ausführung auf die erste (leere) Aktionszeile und stoppt die Ausführung direkt wieder.

Editieren von Makros

Makros werden im Datenbankfenster wie alle anderen Datenbankobjekte verwaltet. So können Sie ein markiertes Makro mithilfe der **Entf**-Taste oder über den Menüpunkt *Bearbeiten > Löschen* löschen. Auch die Funktionen der Zwischenablage lassen sich ohne Probleme nutzen, so dass Sie beispielsweise ein Makro kopieren und damit duplizieren können. Bei der Namensvergabe gelten die gleichen Regeln wie bei den anderen Datenbankobjekten. So darf in Makronamen kein Punkt (.) enthalten sein. Außerdem ist die Vergabe von doppelten Makronamen nicht zulässig.

Hinweis



Wenn Sie das Makro auf dem Drucker ausgeben möchten, wählen Sie den Menüpunkt *Datei > Drucken* aus. Es erscheint ein Dialogfenster, in dem Sie entscheiden können, welche Informationen des Makros gedruckt werden sollen. Nachdem Sie mit OK bestätigt haben, wird das Makro auf den Drucker ausgegeben. Wenn Sie vorher die Seitenansicht des Ausdrucks auf dem Bildschirm begutachten möchten, wählen Sie den Menüpunkt *Datei > Seitenansicht* aus.

Rückgängigmachen einer Änderung



Oft ist es notwendig, bestimmte Änderungen an Makros vorzunehmen. Beschränken sich diese Änderungen auf die Aktionsargumente einiger Aktionen, so können Sie die Inhalte auf die gewohnte Art und Weise editieren. Sobald Sie jedoch in der Spalte *Aktion* eine bereits vorhandene Aktion durch eine neue ersetzen, gehen alle Einstellungen der ursprünglichen Aktionsargumente verloren. Sollte dies versehentlich passieren, können Sie die letzte Änderung über den Menüpunkt *Bearbeiten > Rückgängig* oder einen Mausklick auf das gleichnamige Symbol in der Symbolleiste rückgängig machen.

Verschieben von Aktionszeilen



Wenn Sie eine Aktion an eine andere Position verschieben möchten, klicken Sie mit der Maus auf die Schaltfläche am linken Rand der Aktionszeile. Diese Schaltfläche wird auch als *Aktionszeilenmarkierer* bezeichnet. Durch das Anklicken des Aktionszeilenmarkierers wird die komplette Aktionszeile markiert und mit einem Pfeil versehen. Die so markierte Zeile können Sie nun per Drag & Drop an eine neue Position verschieben. Dazu positionieren Sie den Mauszeiger auf die Pfeilschaltfläche und drücken die linke Maustaste nieder. Bewegen Sie den Mauszeiger bei gedrück



gehaltener Maustaste an die neue Position. Die genaue Position, an der die Aktion eingefügt wird, kennzeichnet Access durch eine horizontale Linie im Makrofenster. Nachdem Sie die Maustaste gelöst haben, fügt Access die Aktion inklusive den zugehörigen Aktionsargumenten an der neuen Position ein. Sind an der neuen Position bereits Aktionen vorhanden, werden diese automatisch nach unten verschoben.

Mit der gleichen Technik lassen sich auch mehrere Aktionszeilen gleichzeitig verschieben. Hierzu müssen Sie lediglich mehrere Zeilen gleichzeitig markieren. Um dies zu bewerkstelligen, markieren Sie zunächst die erste Aktion, indem Sie den Aktionszeilenmarkierer der betreffenden Aktionszeile anklicken. Bevor Sie die Maustaste lösen, ziehen Sie ihn nun über alle folgenden Zeilen, die Sie markieren möchten. Access stellt daraufhin auch die Aktionszeilenmarkierer dieser Aktionszeilen dunkel hinterlegt dar. Wenn Sie die Maustaste jetzt lösen, sind die Aktionszeilen markiert.

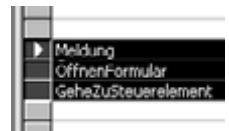
Eine weitere Möglichkeit besteht darin, dass Sie zunächst die erste Aktionszeile markieren und anschließend bei gedrückter -Taste auf den Aktionszeilenmarkierer der letzten Aktionszeile klicken. Bei beiden Vorgehensweisen ist es nur möglich, einen zusammenhängenden Block von Aktionszeilen zu markieren.

Auf markierte Aktionszeilen können Sie die Funktionen der Zwischenablage auf die gewohnte Art und Weise anwenden. Der Menüpunkt *Bearbeiten* > *Löschen* löscht beispielsweise die markierten Aktionszeilen, wogegen der Menüpunkt *Bearbeiten* > *Ausschneiden* diese vor dem Löschen in die Zwischenablage kopiert. Um schnellen Zugriff auf die Funktionen der Zwischenablage zu erhalten, können Sie über einen Klick mit der rechten Maustaste auch das Kontextmenü aktivieren, in dem die Menüpunkte *Ausschneiden*, *Kopieren* und *Einfügen* zur Verfügung stehen. Natürlich stehen auch die gewohnten Tastenkombinationen **Strg**+**X** (Ausschneiden), **Strg**+**C** (Kopieren) und **Strg**+**V** (Einfügen) zur Verfügung.

Möchten Sie bestimmte Aktionszeilen duplizieren, markieren Sie diese und wählen den Menüpunkt *Bearbeiten* > *Kopieren* aus. Bewegen Sie die Einfügemarke nun auf die Aktionszeile, nach der die Aktionen der Zwischenablage eingefügt werden sollen, und wählen Sie den Menüpunkt *Bearbeiten* > *Einfügen* aus. Natürlich erreichen Sie das Ziel auch, wenn Sie die gleichnamigen Menüpunkte des Kontextmenüs oder die o.g. Tastenkombinationen verwenden.

Wenn Sie eine bestimmte Aktionszeile löschen möchten, positionieren Sie den Mauszeiger auf der Aktionszeile und aktivieren mit der rechten Maustaste das Kontextmenü. Hier wählen Sie nun den Menüpunkt *Zeilen löschen* aus. Auf ähnliche Weise können Sie auch eine leere Aktionszeile einfügen. Wählen Sie dazu den Menüpunkt *Zeilen einfügen* des Kontextmenüs aus. Die leere Zeile wird oberhalb der aktuellen Aktionszeile eingefügt.

Markieren mehrerer Aktionszeilen



Zwischenablage



Hinweis





Fehlersuche und Fehlermeldungen

Access überprüft bei der Eingabe von Aktionen und deren Aktionsargumenten, ob alle Eingaben in der richtigen Schreibweise und Syntax erfolgen. Dennoch kann ein Makro fehlerhaft sein.



Um den Fehler genauer einzugrenzen, bietet Access die Möglichkeit zur schrittweisen Ausführung des Makros. Dazu markieren Sie den Menüpunkt *Einzelschritt* des Menüs *Ausführen* oder das gleichnamige Symbol in der Symbolleiste. Das Makro befindet sich nun im Einzelschrittmodus. Um den Einzelschrittmodus wieder zu deaktivieren, wählen Sie den Menüpunkt ein zweites Mal aus oder klicken erneut auf das *Einzelschritt*-Symbol.

Einzelschritt-
ausführung

Sobald Sie das Makro im Einzelschrittmodus auf die gewohnte Art und Weise ausführen, erscheint vor der Abarbeitung der ersten Aktion das Dialogfenster *Einzelschritt*.

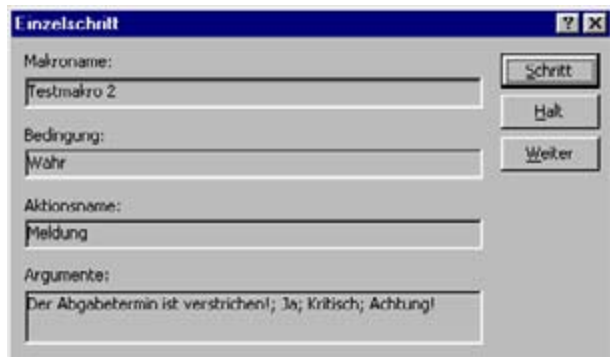


Abb. 1.11

Das *Einzelschritt*-Dialog-
fenster

Hier zeigt Access den Makronamen sowie den Namen der Aktion an, die momentan ausgeführt wird. Zusätzlich wird die Bedingung angezeigt, die zur Ausführung der Aktion geführt hat. Der Text im Feld *Bedingung* besitzt den folgenden Aufbau: Zunächst wird das Ergebnis der Bedingung angezeigt (*Wahr* oder *Falsch*). Haben Sie für die aktuelle Aktion keine Bedingung angegeben, erscheint stets der Text *Wahr*. Anschließend folgt ein Doppelpunkt und – falls eine Bedingung angegeben wurde – der vollständige Text dieser Bedingung. Das Feld *Argumente* enthält die Inhalte aller Aktionsargumente. Diese werden – jeweils durch ein Semikolon getrennt – in der gleichen Reihenfolge angegeben, in der die Aktionsargumente im Makrofenster angeordnet sind. Die vier Felder dienen lediglich zur Kontrolle der Aktion. Änderungen können nicht vorgenommen werden!

Funktion der
Schaltflächen

Das Dialogfenster *Einzelschritt* erscheint auf dem Bildschirm, *bevor* Access die betreffende Aktion ausgeführt hat. Über die Schaltfläche *Schritt* können Sie nun die Anweisung geben, die Aktion auszuführen. Access arbeitet diese Aktion ab und zeigt die nächste Aktion im *Einzelschritt*-Fenster an. Über die Schaltfläche *Halt* kön-



nen Sie die Ausführung des Makros unterbrechen. Die Schaltfläche *Weiter* bewirkt, dass Access den Einzelschrittmodus deaktiviert und die restlichen Aktionen des Makros ohne Unterbrechung abarbeitet.

Der Einzelschrittmodus bleibt auch dann aktiviert, wenn Sie alle Makros schließen. Um ihn zu deaktivieren, klicken Sie nach dem Start eines Makros im Dialogfenster *Einzelschritt* auf die Schaltfläche *Weiter*.



Hinweis

Tritt im Unterbrechungsmodus oder während der normalen Makroausführung ein Fehler auf – beispielsweise wenn auf ein nicht geöffnetes Formular Bezug genommen wird –, zeigt Access zunächst ein Dialogfenster mit dem jeweiligen Fehlertext an. Anschließend erscheint das Dialogfenster *Aktion ist fehlgeschlagen*, das den gleichen Aufbau wie das *Einzelschritt*-Fenster hat. Anhand der Informationen in diesem Dialogfenster können Sie die Fehlerursache weiter eingrenzen. Zum Schließen des Dialogfensters betätigen Sie die *Halt*-Schaltfläche. Alle anderen Schaltflächen des Dialogfensters sind beim Auftreten eines Fehlers deaktiviert und haben damit keine Funktion.



Fehlermeldungen

Ereignisse

In den vorhergehenden Abschnitten haben Sie ein Makro zu Testzwecken immer nur über das *Ausführen*-Symbol in der Symbolleiste, das Datenbankfenster oder den Menüpunkt *Extras > Makro* gestartet. Dies mag zwar in einigen Fällen praktisch sein; es wäre aber schön, wenn das Makro automatisch zum passenden Zeitpunkt ausgeführt würde. Der "passende Zeitpunkt" wird in Access als *Ereignis* bezeichnet. Ein Ereignis kann beispielsweise das Öffnen oder Schließen eines bestimmten Formulars oder der Klick auf eine Schaltfläche sein.



Um den Umgang mit Ereignissen besser kennen zu lernen, legen Sie eine Datenbank und ein neues, leeres Formular an. Auf diesem Formular platzieren Sie anschließend über das Werkzeug *Befehlsschaltfläche* der Toolbox eine neue Schaltfläche. Der Ordnung halber öffnen Sie das Eigenschaftfenster der neuen Schaltfläche und ändern sowohl die *Name*-Eigenschaft als auch die *Beschriftung*-Eigenschaft in "Schaltfläche1".



Anlegen einer Schaltfläche

Im fertigen Formular soll nun ein Mausklick auf die Schaltfläche genügen, um ein bestimmtes Makro auszuführen. Zum Test dient ein einfaches Makro, das eine *Meldung* anzeigt. Legen Sie das Makro an und speichern Sie es unter dem Namen *Meldung anzeigen*. Um sicherzustellen, dass Ihnen beim Erzeugen des Makros kein Fehler unterlaufen ist, sollten Sie es zu Testzwecken starten.

Sie haben nun zwei Komponenten: eine Schaltfläche und ein Makro. Um das Makro bei einem Klick auf die Schaltfläche ausführen zu lassen, müssen Sie die beiden Komponenten miteinander verknüpfen. Dies erledigen Sie über das Eigenschaftfenster der Befehlsschaltfläche. Hier finden Sie im Register *Ereignis* eine Eigenschaft mit dem Namen *Beim Klicken*. Bei der *Beim Klicken*-Eigenschaft handelt es sich nicht um eine normale Eigenschaft, sondern um eine *Ereigniseigenschaft*. Sie dient als Bindeglied zwischen einem Ereignis und einem Makro.



Verknüpfen des Makros mit der Schaltfläche



Wenn Sie auf eine Schaltfläche klicken, löst Access für Sie das *Klick*-Ereignis aus. Sobald dieses Ereignis eintritt, untersucht Access den Inhalt der *Beim Klicken*-Eigenschaft und führt das hier eingetragene Makro aus. Sie müssen also lediglich den Namen des eben erstellten Makros *Meldung anzeigen* in die *Beim Klicken*-Eigenschaft der *Schaltfläche* eintragen. Das Eintragen des Makros können Sie sich erleichtern, indem Sie einfach die Kombinationsfeldliste der Ereignisseigenschaft aktivieren. Aus der Liste aller Makros der aktuellen Datenbank können Sie nun das Makro *Meldung anzeigen* auswählen.

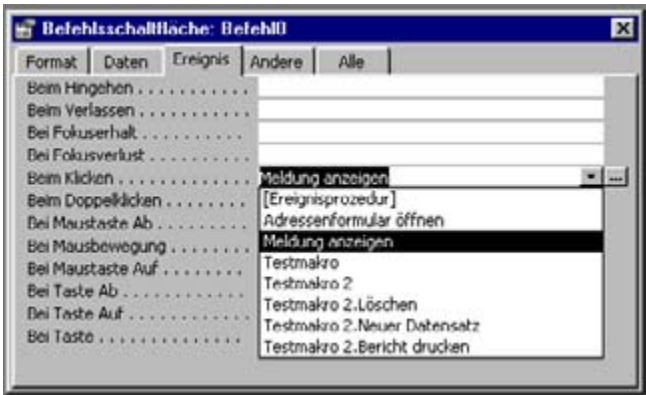


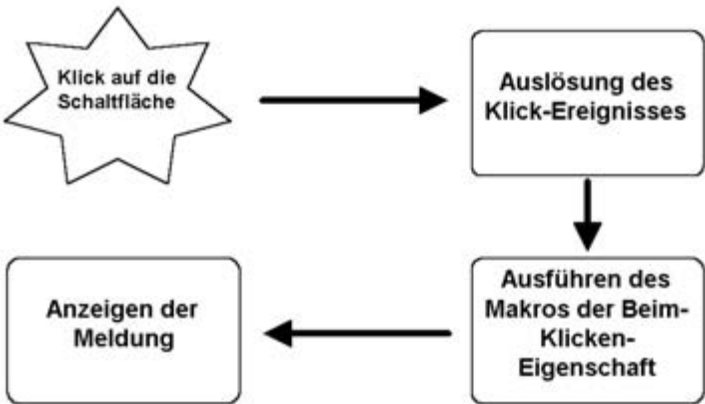
Abb. 1.12

Die Ereignisseigenschaft *Beim Klicken* der Schaltfläche

Wenn Sie das Formular anschließend in der Formularansicht öffnen und auf die Schaltfläche klicken, startet Access das Makro und zeigt schließlich die Meldung an.

Abb. 1.13

Abfolge bis zur Anzeige des Meldungsfensters



Auf die gleiche Weise legen Sie nun noch eine zweite Schaltfläche mit dem Namen *Schaltfläche2* an. Auch hier finden Sie wieder die *Beim Klicken*-Ereigniseigenschaft, in die Sie nun wiederum einen Makronamen eintragen können. Um beim Klicken auf diese Schaltfläche eine andere Meldung anzuzeigen, müssen Sie ein neues Makro anlegen und den Namen dieses Makros schließlich in die *Beim Klicken*-Ereigniseigenschaft der zweiten Schaltfläche eintragen.

Das fertige Formular finden Sie in der Datenbank *Kapitel1.mdb* unter dem Namen *Ereignis1*.

Wenn Sie ein Untermakro durch eines der Ereignisse ausführen lassen möchten, geben Sie das Makro in der folgenden Form an:

```
<Gruppenmakroname>.<Untermakroname>
```

Um beispielsweise das Untermakro *Formular Schließen* des Gruppenmakros *Projekt* zu starten, geben Sie den folgenden Eintrag in eine der Ereigniseigenschaften ein:

```
Projekt.Formular Schließen
```

Zur Vereinfachung der Eingabe können Sie im Eigenschaftenfenster auch wieder auf die Makroliste der betreffenden Ereigniseigenschaft zurückgreifen. Neben den Namen der Gruppenmakros werden hier auch alle Untermakros aufgeführt.

In der Makroliste finden Sie an erster Stelle den Eintrag *[Ereignisprozedur]*. Dieser ermöglicht es Ihnen, auf das Ereignis mithilfe einer VBA-Prozedur zu reagieren. Mehr dazu erfahren Sie im *Kapitel 2 Schnellstart VBA*.



Hinweis

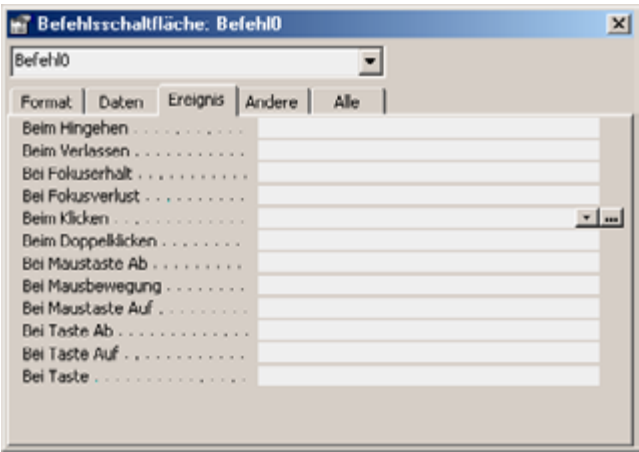
Der Makro-Generator

Etwas komfortabler wird der Weg zum fertigen Makro mithilfe des *Makro-Generators*, der das Anlegen des Makros und das Eintragen des Makronamens in die gewünschte Ereigniseigenschaft automatisch erledigt. Es kann dann im Makro-Editor weiterbearbeitet werden. Zum Start des Generators positionieren Sie die Einfügemarke in das Eingabefeld der gewünschten Ereigniseigenschaft. Rechts neben der Ereigniseigenschaft blendet Access nun die Editor-Schaltfläche (drei Punkte) ein, die Sie mit der Maus anklicken. Es erscheint das Dialogfenster *Generator auswählen*, das den Start von drei verschiedenen Generatoren erlaubt. Zum Start des Makro-Generators klicken Sie doppelt auf den gleichnamigen Listeneintrag. Access legt nun ein neues Makro an und fordert Sie direkt anschließend zur Eingabe des Makronamens auf. Nach Ihrer Eingabe wird dieser Makroname automatisch in die Ereigniseigenschaft eingetragen. Damit Sie direkt mit der Bearbeitung des Makros beginnen können, wird es von Access außerdem in der Entwurfsansicht geöffnet.



Abb. 1.14

Die Editor-Schaltfläche
der *Beim Klicken*-
Eigenschaft



Hinweis



Ist in eine Ereignisseigenschaft bereits ein Makro eingetragen, bewirkt ein Klick auf die Editor-Schaltfläche, dass Access dieses direkt zur Bearbeitung in der Entwurfsansicht öffnet.

Übersicht aller Ereignisse

Beim Doppelklicken

Eine Befehlsschaltfläche stellt neben der *Beim Klicken*-Ereigniseigenschaft noch weitere Ereignisseigenschaften – beispielsweise die *Beim Doppelklicken*-Ereigniseigenschaft – zur Verfügung. Das hier eingetragene Makro wird von Access ausgeführt, wenn das *Doppelklicken*-Ereignis eintritt, d.h. wenn Sie kurz hintereinander doppelt auf die Schaltfläche klicken.

Nicht nur Schaltflächen, sondern auch fast alle anderen Steuerelemente besitzen Ereignisseigenschaften. Die folgende Tabelle enthält eine Übersicht mit einer kurzen Erläuterung.

Ereigniseigenschaften von Steuerelementen und deren Auslöser

Ereigniseigenschaft	Beschreibung
<i>Vor Aktualisierung</i>	Der Inhalt des Steuerelements wurde geändert und es verliert den Fokus. Das Ereignis tritt jedoch ein, bevor Access die Daten des Steuerelements in der zugrunde liegenden Tabelle speichert.
<i>Nach Aktualisierung</i>	Der Inhalt des Steuerelements wurde geändert und es verliert den Fokus. Das Ereignis tritt ein, nachdem Access die Daten des Steuerelements bereits in der zugrunde liegenden Tabelle gespeichert hat.
<i>Bei Änderung</i>	Der Inhalt des Steuerelements hat sich geändert.



Ereigniseigenschaften von Steuerelementen und deren Auslöser (Forts.)

Ereigniseigenschaft	Beschreibung
<i>Beim Hingehen</i>	Das Steuerelement erhält den Fokus beispielsweise durch Anklicken oder Auswahl über die  -Taste von einem anderen Steuerelement des gleichen Formulars.
<i>Beim Verlassen</i>	Das Steuerelement verliert den Fokus an ein anderes Steuerelement des gleichen Formulars.
<i>Bei Fokuserhalt</i>	Das Steuerelement erhält den Fokus.
<i>Bei Fokusverlust</i>	Das Steuerelement verliert den Fokus.
<i>Beim Klicken</i>	Das Steuerelement wird einmal angeklickt.
<i>Beim Doppelklicken</i>	Das Steuerelement wird doppelt angeklickt.
<i>Bei Maustaste Ab</i>	Eine Maustaste wird niedergedrückt.
<i>Bei Mausbewegung</i>	Die Maus wird über ein Objekt hinweg bewegt.
<i>Bei Maustaste Auf</i>	Eine Maustaste wird gelöst.
<i>Bei Taste Ab</i>	Eine Taste wird niedergedrückt.
<i>Bei Taste</i>	Eine Taste wurde betätigt.
<i>Bei Taste Auf</i>	Eine Taste wurde gelöst.
<i>Bei Nicht In Liste</i>	Diese Eigenschaft steht nur bei Kombinationsfeldern zu Verfügung. Das <i>Nicht In Liste</i> -Ereignis tritt ein, wenn ein Text eingegeben wird, der nicht in der Kombinationsfeldliste enthalten ist.
<i>Bei OLE-Aktualisierung</i>	Diese Eigenschaft steht nur bei Objektfeldern und ActiveX-Steuerelementen zur Verfügung und tritt ein, wenn die Verknüpfung vom OLE-Server aktualisiert wird.

Tabelle 1.11

Ereigniseigenschaften von Formularen

Auch Formulare stellen eine Reihe von Ereigniseigenschaften zur Verfügung. Das *Anzeigen*-Ereignis der *Beim Anzeigen*-Ereigniseigenschaft löst Access beispielsweise immer dann aus, wenn Sie den Datensatz eines Formulars wechseln. Es folgt eine Übersicht über alle Ereigniseigenschaften von Formularen:



Ereigniseigenschaften von Formularen

Ereigniseigenschaft	Beschreibung
<i>Beim Anzeigen</i>	Ein anderer Datensatz wird ausgewählt.
<i>Vor Eingabe</i>	Eingabe eines neuen Datensatzes, bevor Access den neuen Datensatz in der zugrunde liegenden Tabelle anlegt.
<i>Nach Eingabe</i>	Eingabe eines neuen Datensatzes, nachdem Access den neuen Datensatz in der zugrunde liegenden Tabelle angelegt hat.
<i>Vor Aktualisierung</i>	Wechsel von einem veränderten Datensatz zu einem anderen Datensatz, bevor Access die Änderungen in der zugrunde liegenden Tabelle speichert.
<i>Nach Aktualisierung</i>	Wechsel von einem veränderten Datensatz zu einem anderen Datensatz, nachdem Access die Änderungen in der zugrunde liegenden Tabelle gespeichert hat.
<i>Bei Änderung</i>	Änderung von Daten in einem Kombinations-, Text- oder Registersteuerelement.
<i>Beim Löschen</i>	Der aktuelle Datensatz wird gelöscht.
<i>Vor Löschbestätigung</i>	Bevor die Sicherheitsabfrage beim Löschen eines Datensatzes angezeigt wird.
<i>Nach Löschbestätigung</i>	Nachdem die Sicherheitsabfrage beim Löschen eines Datensatzes angezeigt wurde.
<i>Beim Öffnen</i>	Das Formular wird geöffnet.
<i>Beim Laden</i>	Die Datensätze eines Formulars wurden angezeigt.
<i>Bei Größenänderung</i>	Die Größe des Formulars wurde verändert.
<i>Beim Entladen</i>	Die Datensätze eines Formulars werden entfernt.
<i>Beim Schließen</i>	Das Formular wird geschlossen.
<i>Bei Aktivierung</i>	Das Formular wird aktiviert.
<i>Bei Deaktivierung</i>	Das Formular wird deaktiviert.
<i>Bei Fokuserhalt</i>	Das Formular erhält den Fokus. Dies ist nur möglich, wenn sich kein aktives oder sichtbares Steuerelement auf dem Formular befindet.



Ereigniseigenschaften von Formularen (Forts.)

Ereigniseigenschaft	Beschreibung
<i>Bei Fokusverlust</i>	Das Formular verliert den Fokus. Dies ist nur möglich, wenn sich kein aktives oder sichtbares Steuerelement auf dem Formular befindet.
<i>Beim Klicken</i>	Das Formular wird angeklickt.
<i>Beim Doppelklicken</i>	Das Formular wird doppelt angeklickt.
<i>Bei Maustaste Ab</i>	Eine Maustaste wird niedergedrückt.
<i>Bei Mausbewegung</i>	Die Maus wird über ein Objekt hinweg bewegt.
<i>Bei Maustaste Auf</i>	Eine Maustaste wird gelöst.
<i>Bei Taste Ab</i>	Eine Taste wird niedergedrückt.
<i>Bei Taste</i>	Eine Taste wurde betätigt.
<i>Bei Taste Auf</i>	Eine Taste wurde gelöst.
<i>Tastenvorschau</i>	Tastaturereignisse für Formulare vor Steuerelementen aufrufen (<i>Ja/Nein</i>).
<i>Bei Fehler</i>	Im Formular tritt ein Fehler auf.
<i>Bei Filter</i>	Ein Filter wird bearbeitet.
<i>Bei angewendetem Filter</i>	Ein Filter wird aktiviert oder deaktiviert.
<i>Bei Zeitgeber</i>	Das <i>Zeitgeber</i> -Ereignis wurde ausgelöst.
<i>ZeitgeberIntervall</i>	Intervall des Zeitgebers in Millisekunden.

Tabelle 1.12

Bei Berichten spielen Ereignisse eine weniger wichtige Rolle. Wenn Sie beispielsweise ein Steuerelement auf einem Bericht platzieren, stehen hier überhaupt keine Ereigniseigenschaften zur Verfügung. Die Bereiche eines Berichts haben dagegen eine Reihe von Ereigniseigenschaften, mit denen Sie die Formatierung beeinflussen können. Mehr dazu erfahren Sie im zweiten Teil dieses Buches. Hier erfahren Sie auch, wie Sie die Ereigniseigenschaften von Formularen und Steuerelementen in der Praxis einsetzen.

Jede Ereigniseigenschaft korrespondiert mit einem bestimmten Ereignis. Der Name der Ereigniseigenschaft wird in fast allen Fällen durch das Wort *Bei* bzw. *Beim* und den Namen des betreffenden Ereignisses gebildet (z. B. *MaustasteAb*-Ereignis => *Bei Maustaste Ab*-Eigenschaft). Eine Ausnahme bilden lediglich die mit *Vor* oder *Nach* beginnenden Ereigniseigenschaften, deren Namen mit dem Ereignisnamen übereinstimmen. In der auf der CD zum Buch enthaltenen Referenz sind die Ereignisse



Hinweis





unter den entsprechenden Einträgen der Ereignisseigenschaften aufgeführt. Wenn Sie also beispielsweise Informationen zum *Klick*-Ereignis benötigen, suchen Sie den Eintrag *Beim Klicken-Eigenschaft*.

Sie haben nun das notwendige Wissen, um mithilfe von Makros einfache Aufgaben zu automatisieren. Im folgenden Kapitel erfahren Sie, wie sich diese Aufgaben mit sehr wenig Aufwand mithilfe von VBA lösen lassen.





Schnellstart VBA



Prozeduren	32
Ausführen von Aktionen	44
<i>If...Then...Else</i>	50



2



Was ist VBA?

Access stellt zwei Möglichkeiten zur Programmierung zur Verfügung: *Makros* und *VBA-Programme*. Makros haben den Vorteil, dass sich einfache Aufgaben sehr schnell und problemlos erledigen lassen. Wenn es allerdings komplizierter wird, stoßen Makros leicht an ihre Grenzen. Mithilfe von *VBA (Visual Basic for Applications)* lassen sich Probleme genauso einfach wie mit Makros lösen. Allerdings ist etwas mehr Grundwissen als beim Umgang mit Makros erforderlich.

In dem vorhergehenden *Kapitel 1 Einführung in die Makroprogrammierung* haben Sie die grundlegende Technik der Ereignisprogrammierung anhand einiger kleiner Makros kennen gelernt. In diesem Kapitel geht es nun darum, wie Sie diese Beispiele in VBA umsetzen. Lassen Sie sich überraschen, wie einfach das geht!

Prozeduren

Im ersten Kapitel haben Sie ein Beispielformular angelegt, das zwei Schaltflächen enthielt. Die beiden Schaltflächen wurden mithilfe der *Beim Klicken*-Eigenschaft an zwei verschiedene Makros geknüpft. Sobald Sie nun das Formular in der Formularansicht öffnen und auf eine der beiden Schaltflächen klicken, löst Access das *Klick*-Ereignis aus und startet das entsprechende Makro.

Anlegen einer Prozedur

Bei der Programmierung dieses Beispiels mithilfe von VBA läuft alles genau auf die gleiche Weise ab. Der einzige Unterschied besteht darin, dass Sie das *Klick*-Ereignis nicht mit einem Makro, sondern mit einer *VBA-Prozedur* verknüpfen. Um nun auf einen Klick reagieren zu können, müssen Sie eine Prozedur anlegen. Dies erledigen Sie am einfachsten unter Zuhilfenahme des Assistenten mit dem Namen *Code-Generator*. Dieser Generator ist von der Wirkung her mit dem *Makro-Generator* vergleichbar, den Sie im ersten Kapitel kennen gelernt haben.

Öffnen Sie zunächst ein neues Formular und legen Sie hier eine Schaltfläche mit dem Namen "Schaltfläche1" an. Achten Sie darauf, dass Sie nicht nur die *Beschriftung*-Eigenschaft, sondern auch die *Name*-Eigenschaft in "Schaltfläche1" abändern. Die Änderung der *Name*-Eigenschaft ist besonders wichtig, da Namen bei der Programmierung mit VBA eine sehr große Rolle spielen.



Genau wie vorher im Makrobeispiel positionieren Sie die Einfügemarke anschließend im Eigenschaftenfenster der Schaltfläche auf die *Beim Klicken*-Eigenschaft. Am rechten Rand erscheint die bekannte Editor-Schaltfläche, mit der Sie das Dialogfenster *Generator auswählen* aktivieren.





Abb. 2.1

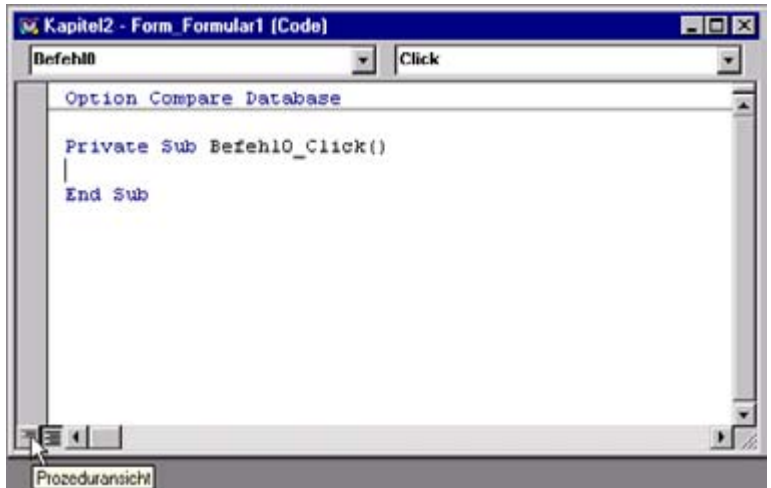
Das Dialogfenster
Generator auswählen

Da Sie nun kein Makro anlegen, sondern mithilfe von VBA-Code auf das Ereignis reagieren möchten, wählen Sie den Eintrag *Code-Generator* aus und bestätigen mit *OK*. Daraufhin öffnet Access in der Visual Basic-Umgebung ein Modulfenster und legt hier automatisch die folgende – noch leere – Prozedur an:

```
Private Sub Schaltfläche1_Click ()  
  
End Sub
```

Abb. 2.2

Das Modulfenster mit
der Prozedur *Schaltfläche1_Click*



Das Modulfenster bietet zwei Ansichten. Mit den Schaltflächen in der unteren linken Ecke können Sie auswählen, ob Sie in der *Prozeduransicht* nur eine Prozedur oder in der *vollständigen Modulansicht* das gesamte Modul bearbeiten möchten. Die standardmäßig von Ihnen bevorzugte Ansicht können Sie in der Registerkarte *Extras > Optionen > Editor* einstellen.



Hinweis





Im Gegensatz zu Makros besteht eine Prozedur aus einem geschriebenen Text, der bestimmten Regeln folgen muss. Diese Regeln werden in der Fachsprache auch als *Syntax* bezeichnet. Wird die Syntax nicht beachtet, kann Access die Prozedur nicht verstehen und gibt einen Fehler aus (den berühmten *Syntax-Error*).

Aufbau einer Prozedur

An der obigen Prozedur werden die ersten Regeln für den Aufbau einer Prozedur deutlich:

- Eine Prozedur wird immer mit dem Wort *Sub* eingeleitet. Solche Wörter, die innerhalb des Programmcodes eine besondere Bedeutung haben, werden auch als *Schlüsselwörter* bezeichnet. Das Schlüsselwort *Private* hat eine spezielle Bedeutung für den Geltungsbereich einer Prozedur. Die Angabe dieses Schlüsselworts ist nicht unbedingt erforderlich.
- Dem Schlüsselwort *Sub* folgt der Name der Prozedur. Der Name muss ebenfalls einigen Regeln folgen. So darf er beispielsweise nicht mit einem reservierten Schlüsselwort wie z.B. *Sub* übereinstimmen und keine Sonderzeichen enthalten. Die genauen Regeln zur Namensbildung lernen Sie im nächsten Kapitel kennen. Wenn Sie – wie im Beispiel der Prozedur *Schaltfläche1_Click* – mit dem *Code-Generator* arbeiten, gibt Access der Prozedur automatisch einen passenden Namen, so dass Ihnen diese Arbeit erspart bleibt.
- Dem Namen muss ein Paar runder Klammern folgen. Wie Sie im folgenden Kapitel noch näher erfahren, können innerhalb dieser Klammern bei Bedarf Parameter in Form einer Parameterliste angegeben werden.
- Mit dem Klammernpaar ist der Kopf der Prozedur abgeschlossen. Hinter dem Prozedurkopf darf in derselben Zeile kein weiterer Text mehr angegeben werden.
- Das Ende einer Prozedur wird immer mit den Schlüsselwörtern *End Sub* markiert. Dies ist notwendig, da mehrere Prozeduren aufeinander folgen können. Access erfährt so, wo es die Ausführung der Prozedur beenden muss.

—○ Um die Regeln etwas kürzer zu fassen, werden diese in Form einer Befehlssyntax angegeben. Eine Befehlssyntax ist nichts anderes als ein Lückentext, in den Sie an den passenden Stellen etwas einfügen:

Befehlssyntax

```
[Private] Sub <Prozedurname> ( [<Parameterliste>] )  
  
[<Anweisungen>]  
  
End Sub
```



Alle Texte, die von spitzen Klammern <...> umgeben sind, müssen von Ihnen ersetzt werden. Ein Beispiel ist der Text <Prozedurname>, den Sie durch den Namen Ihrer Prozedur ersetzen. Die Angabe einer <Parameterliste> ist dagegen nicht zwingend erforderlich, sie wird daher in eckigen Klammern angegeben. Gleiches gilt für das Schlüsselwort *Private*. Alle anderen Texte inklusive der Leerzeichen und Zeilenwechsel müssen ohne Änderung übernommen werden. Lediglich die Groß- und Kleinschreibung spielt keine Rolle.

Wie Sie der Befehlssyntax entnehmen, können zwischen dem Kopf der Prozedur und dem durch *End Sub* markierten Ende nacheinander mehrere <Anweisungen> angegeben werden. Diese sind mit den einzelnen Aktionen vergleichbar, die Sie im Entwurfsbereich eines Makros angegeben haben. An den eckigen Klammern des Arguments <Anweisungen> erkennen Sie, dass die Angabe von Anweisungen nicht unbedingt notwendig ist; genauso wie ein Makro ohne Aktionen wäre allerdings auch eine Prozedur ohne Anweisungen überflüssig.

Da eine Prozedur als Text (auch *Programmcode* genannt) eingegeben wird, stellt Access zur Eingabe das Modulfenster (Codefenster) zur Verfügung. Bei diesem Fenster handelt es sich um nichts anderes als einen Texteditor, der in der integrierten Visual-Basic-Umgebung (VBA) enthalten ist und einige spezielle Funktionen zur Programmierung bietet. Wenn Sie schon einmal mit einer Textverarbeitung oder einem Texteditor gearbeitet haben, werden Sie mit der Bedienung keine Probleme haben.

Wie Sie im Modulfenster sehen, hat Access bereits den kompletten Rumpf der Prozedur inklusive eines passenden Namens angelegt. Diese Arbeit wurde vom *Code-Generator* erledigt, den Sie vorher über das Dialogfenster *Generator auswählen* gestartet haben.

Der automatisch vom Code-Generator vergebene Prozedurname *Schaltfläche1_Click* setzt sich aus dem Namen des Steuerelements, einem Unterstrich und dem Namen des Ereignisses zusammen, für das diese Prozedur vorgesehen ist. Alle Ereignisnamen werden hierbei grundsätzlich in englischer Sprache angegeben, da Access fast ausschließlich Englisch versteht. Dies mag auf den ersten Blick als Einschränkung erscheinen; es bringt jedoch Vorteile mit sich, wenn Sie eine Prozedur unter einer anderssprachigen Version von Access ausführen möchten. Die Ausführbarkeit Ihres Programmcodes ist also sprachübergreifend sichergestellt.

Über den Namen der Prozedur kann Access direkt erkennen, welchem Steuerelement und welchem Ereignis die Prozedur zugeordnet ist. Der Name *Schaltfläche1_Click* verrät Access beispielsweise, dass die Prozedur mit dem *Klick*-Ereignis der Schaltfläche *Schaltfläche1* verknüpft ist. Ein Problem tritt nun auf, wenn Sie den Namen der Schaltfläche nachträglich ändern. In diesem Fall geht die Verknüpfung zwischen dem Steuerelement und der Prozedur verloren. Die Prozedur wird jedoch nicht gelöscht. Sie können sie aktivieren, indem Sie im Kombinationsfeld *Objekt* (links) den Eintrag (*Allgemeine*) und im Kombinationsfeld *Prozedur* (rechts) den Namen der Prozedur auswählen. Beide Kombinationsfelder finden Sie oben im Modulfenster.

Anweisungen

Das Modulfenster

Aufbau des Prozedurnamens



Abb. 2.3

Die Kombinationsfelder
Objekt und Prozedur



Um die Verknüpfung wiederherzustellen, passen Sie den Namen der Prozedur einfach an. Alternativ können Sie auch über den Code-Editor eine neue Ereignisprozedur anlegen und den Programmcode via Zwischenablage aus der alten in die neue Prozedur kopieren. Anschließend löschen Sie die alte Prozedur.

Anzeigen einer Meldung

Die noch leere Prozedur können Sie nun mit der *MsgBox*-Anweisung füllen, die mit der *Meldung*-Aktion von Makros vergleichbar ist. Um festzulegen, welcher Text in dem Meldungsfenster angezeigt werden soll, geben Sie diesen einfach direkt hinter der *MsgBox*-Anweisung in Anführungszeichen an.

Hinweis



Sobald Sie das Leerzeichen hinter der Anweisung *MsgBox* eingetippt haben, erscheint unterhalb der aktiven Zeile ein Tipp-Fenster mit der kompletten Syntax der *MsgBox*-Anweisung. Sie können die Anzeige dieser automatischen Hilfe mit der Option *automatische Direkthilfe* auf der Registerkarte *Extras* > *Optionen* > *Editor* aus- bzw. einschalten. Weitere Informationen zur Syntax der *MsgBox*-Anweisung finden Sie im nächsten Abschnitt. Geben Sie im Moment nur den Meldungstext ein.

Die fertige Prozedur könnte folgendermaßen aussehen:

```
Private Sub Schaltfläche1_Click ()
MsgBox "Klick auf Schaltfläche 1"
End Sub
```

Strukturierung des
Programmcodes

Um die Lesbarkeit zu verbessern, wird der Programmcode oft mit Leerzeichen und Leerzeilen strukturiert. Bei der Ausführung einer Prozedur werden solche Elemente von Access ignoriert, so dass Ihnen hier freie Hand gelassen wird. Jeder Programmierer hat bei der Strukturierung seine eigene Philosophie, die zwischen schlechter Lesbarkeit und Platzverschwendung schwankt. Die folgende Formatierung ist ein guter Kompromiss:

```
Private Sub Schaltfläche1_Click ()

    MsgBox "Klick auf Schaltfläche 1"

End Sub
```

Das Einrücken können Sie mit der Tabulatortaste () erledigen. Standardmäßig fügt der Tabulator vier Leerzeichen ein. Den Wert können Sie jedoch bei Bedarf über das Dialogfenster *Extras* > *Optionen* verändern. Im Register *Editor* steht dazu die Option *Tabulatorbreite* zur Verfügung.



Nachdem Sie die Anweisung eingegeben und am Ende der Zeile die -Taste betätigt haben, nimmt Access eine Syntax-Kontrolle der Eingabe vor. Sollte Ihnen bei der Eingabe also ein Fehler unterlaufen sein, erscheint eine entsprechende Fehlermeldung. In diesem Fall quittieren Sie das Meldungsfenster und beheben den Schreibfehler.

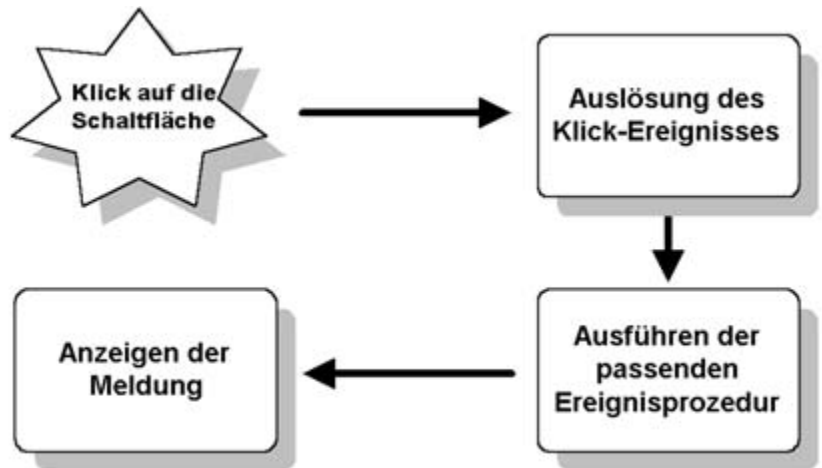


Hinweis

Die Prozedur ist damit fertig gestellt, und Sie können einen Test des Formulars vornehmen. Dazu schließen Sie das Modulfenster, in dem sich Ihre Prozedur befindet, und schalten in die Formularansicht um. Durch einen Klick auf die Schaltfläche erscheint erwartungsgemäß die Meldung. Nachdem Sie diese bestätigt haben, ist die Ausführung der Prozedur beendet.

Abb. 2.4 ○

Abfolge bis zur Anzeige
des Meldungsfensters



Streng genommen handelt es sich bei *MsgBox* nicht um eine Anweisung, sondern eine Funktion. Access erlaubt jedoch, jede Funktion wie eine Anweisung aufzufügen. Der Rückgabewert wird hierbei ignoriert. Mehr zu Funktionen lernen Sie in *Kapitel 3 Die Programmiersprache VBA* kennen.



Hinweis

Erweitern der Prozedur

Im nächsten Schritt soll die Ereignisprozedur weiter ausgebaut werden. Schalten Sie dazu wieder in die Entwurfsansicht des Formulars um und positionieren Sie die Einfügemarke in die *Beim Klicken*-Ereigniseigenschaft. Wie Sie sehen, hat Access hier den Text *[Ereignisprozedur]* eingetragen. Dieser Text kennzeichnet, dass das betreffende Ereignis – in diesem Fall das *Klick*- bzw. *Click*-Ereignis – bereits mit einer Prozedur verknüpft ist. Da die Aufgabe der Prozedur die Reaktion auf ein Ereignis ist, wird sie auch als *Ereignisprozedur* bezeichnet. Diese Bezeichnung hat allerdings keinen Einfluss auf den Aufbau der Prozedur.



Bearbeiten der
Prozedur



Um nun die Prozedur bearbeiten zu können, klicken Sie auf die Editor-Schaltfläche. Access erkennt nun, dass bereits eine Prozedur vorhanden ist und öffnet daher direkt das Modulfenster mit der Prozedur *Schaltfläche1_Click*.

Zur Ergänzung der Meldung soll nun zusätzlich ein Titeltext sowie ein Symbol ausgewählt werden. Bei der *Meldung*-Aktion haben Sie diese Einstellungen über die Aktionsargumente vorgenommen, die bei VBA-Anweisungen allerdings nicht zur Verfügung stehen. Stattdessen müssen die Angaben direkt hinter der Anweisung angegeben werden. Die einzelnen Angaben werden hierbei auch als *Argumente* oder *Parameter* bezeichnet.

Befehlssyntax der
MsgBox-Anweisung

Im Zusammenhang mit der *MsgBox*-Anweisung haben Sie bereits einen Parameter angegeben – nämlich den Text, der in der Meldung erscheinen soll. Auf die gleiche Weise können Sie dem Namen der Anweisung noch weitere Parameter folgen lassen. Welche Parameter die Anweisung unterstützt und in welcher Reihenfolge diese angegeben werden müssen, zeigt die Befehlssyntax der *MsgBox*-Anweisung:

```
MsgBox ( <Prompt> [, <Buttons>] [,  
<Title> ] [, <Helpfile>, <Context>])
```

Es können insgesamt fünf Parameter angegeben werden. Der erste Parameter *<Prompt>* bestimmt den Text, der innerhalb des Meldungsfensters angezeigt wird.

Der zweite Parameter *<Buttons>* bestimmt das Symbol, das innerhalb des Meldungsfensters angezeigt wird. Dieser Parameter ist also mit dem Aktionsargument *Typ* der *Meldung*-Aktion vergleichbar. Innerhalb von VBA steht Ihnen ebenfalls der Komfort zur Verfügung, das gewünschte Symbol über ein Kombinationsfeld auszuwählen. Sie können für den Parameter *<Buttons>* einen bestimmten Zahlenwert angeben oder auch eine Konstante aus der im Editor automatisch erscheinenden Liste auswählen. Welcher Zahlenwert welchem Symbol entspricht, ist aus der Befehlssyntax der *MsgBox*-Anweisung nicht ersichtlich. In der Dokumentation von Access werden solche Dinge daher stets im Begleittext zu einer Anweisung angegeben. Die folgende Tabelle enthält alle Werte und Konstanten sowie die zugehörigen Symbole.

Mögliche Einstellungen des Parameters *<Buttons>*

Einstellung	Symbol	Konstante
0		kein Symbol
16		<i>vbCritical</i>
32		<i>vbQuestion</i>
48		<i>vbExclamation</i>
64		<i>vbInformation</i>

Tabelle 2.1





Anstelle der Werte können auch die Konstanten verwendet werden, die vom Editor in einem Popup-Menü angeboten werden. Dies ist auch empfehlenswert, da sich die Lesbarkeit des Programms damit deutlich verbessert.



Hinweis

Der dritte Parameter *<Title>* ist eine Zeichenkette, die den Titeltext des Meldungsfensters bestimmt.

Mit den Parametern *<Helpfile>* und *<Context>* lässt sich eine kontextsensitive Hilfe implementieren. Wie Sie sehen, sind beide Parameter in einer eckigen Klammer zusammengefasst. Dies bedeutet, dass Sie beide Parameter immer zusammen angeben oder auslassen müssen. Da die Definition eines kontextsensitiven Hilfesystems nur für Experten oder Entwickler interessant ist, wird diese Möglichkeit im Folgenden nicht näher erläutert.

Um der *MsgBox*-Anweisung die fünf Informationen zu übergeben, gibt es zwei Möglichkeiten: Bei der ersten Möglichkeit geben Sie die einzelnen Parameter durch Kommas getrennt hintereinander an. Hierbei ist es wichtig, dass Sie die Reihenfolge der Parameter gemäß der Befehlssyntax richtig einhalten. Das folgende Beispiel zeigt, wie dies aussehen kann:

```
MsgBox "Klick auf Schaltfläche 1", vbExclamation,
"Schaltfläche 1"
```

Angabe der
Parameter

Bei der zweiten Möglichkeit spielt die Reihenfolge der Parameter keine Rolle. Damit Access die einzelnen Parameter jedoch richtig zuordnen kann, müssen Sie diese zusammen mit ihren Namen angeben. Dies könnte folgendermaßen aussehen:

```
MsgBox Prompt:="Klick auf Schaltfläche 1",
Buttons:=vbExclamation, Title:="Schaltfläche 1"
```

Benannte
Parameter

Das folgende Beispiel erfüllt die gleiche Aufgabe. Die Argumente sind jedoch in einer anderen Reihenfolge angegeben:

```
MsgBox Buttons:=vbExclamation, Title:="
Schaltfläche 1", Prompt:="Klick auf Schaltfläche 1"
```

Diese Art von Parametern, die Sie zusammen mit ihren Namen angeben, werden auch als *benannte Parameter* bezeichnet. Jedem einzelnen Parameter müssen Sie hierbei über die Zeichenfolge := einen Ausdruck zuweisen. Optionale Parameter, deren Angabe freigestellt ist, können Sie einfach auslassen.

Bei der Verwendung benannter Parameter ist es erforderlich, dass Sie die Namen der einzelnen Parameter kennen. Sie finden diese sowohl in der Onlinehilfe als auch im zuvor bereits erwähnten Direkthilfe-Tooltip. Eine weitere Möglichkeit zum Ausfindigmachen der richtigen Parameternamen stellt der Objektkatalog zur Verfügung, den Sie in den folgenden Kapiteln noch kennen lernen werden.



Hinweis

Das folgende Beispiel zeigt die vollständige Prozedur, die mit dem *Klick*-Ereignis der *Schaltflächen* verknüpft ist. Zur Parameterübergabe wurde hierbei die erste Möglichkeit ohne Angabe der Parameternamen verwendet.





```
Private Sub Schaltfläche1_Click ()

    MsgBox "Klick auf Schaltfläche 1", vbExclamation, _
        "Schaltfläche 1"

End Sub
```

Zum Test schalten Sie das Formular wieder in die Formularansicht um. Ein Klick auf die Schaltfläche bringt das Meldungsfenster auf den Bildschirm – diesmal allerdings mit Symbol und selbstdefiniertem Fenstertitel.



Abb. 2.5

Die Auswirkungen der drei Parameter

Hinweis



Programmcode ist mindestens genauso wertvoll wie geschriebener Text. Um dem Verlust von Programmcode durch einen Rechnerabsturz oder einen Access-Programmfehler vorzubeugen, sollten Sie den Code regelmäßig speichern. Da Ereignisprozeduren zusammen mit dem Formular gespeichert werden, können Sie dies über den Menüpunkt *Datei > Speichern* oder einen Klick auf das Symbol *Speichern* erledigen.

Mehrere Anweisungen



Wie Sie sicherlich bemerkt haben, ertönt beim Erscheinen des Meldungsfensters kein Signalton (sofern dies nicht in den Windows-Einstellungen vorgegeben wurde). Dies ist normal, da die Aufgabe der *MsgBox*-Anweisung lediglich die Anzeige des Meldungsfensters, nicht jedoch die Ausgabe des Signaltons ist. Für diese Aufgabe stellt Access eine spezielle Anweisung mit dem Namen *Beep* zur Verfügung. Da der Signalton vor dem Anzeigen des Meldungsfensters ertönen soll, muss die Ereignisprozedur folgendermaßen aussehen:

```
Private Sub Schaltfläche1_Click ()

    Beep
    MsgBox "Klick auf
    Schaltfläche 1", vbExclamation, "Schaltfläche 1"

End Sub
```



Wenn Sie die *Beep*-Anweisung dagegen hinter der *MsgBox*-Anweisung platzieren, ertönt der Signalton erst dann, wenn das Meldungsfenster mit *OK* geschlossen wird:

```
Private Sub Schaltfläche1_Click ()

    MsgBox "Klick auf Schaltfläche 1", vbExclamation,
        "Schaltfläche 1"
    Beep

End Sub
```

Wenn Sie in einer Prozedur mehrere Anweisungen angeben möchten, können Sie diese entweder in mehreren Zeilen untereinander oder aber auch innerhalb einer Zeile anordnen. Im letzteren Fall geben Sie zwischen den Anweisungen einen Doppelpunkt (:) an. Access führt die Anweisungen der Zeile dann von links nach rechts aus:

Angabe mehrerer
Anweisungen in
einer Zeile

```
Sub Schaltfläche1_Click ()

    Beep:MsgBox "Klick auf Schaltfläche 1",
        vbExclamation, _
            "Schaltfläche 1"

End Sub
```

Bei dem obigen Beispiel wird jedoch leider ein Fehler in VBA deutlich: Access interpretiert den Text *Beep*: als Zeilensprungmarke und nicht als Anweisung. Aufgrund dieses Fehlers ertönt leider kein Signalton. Bei der umgekehrten Anordnung der Anweisungen tritt dieses Problem dagegen nicht auf:

```
Sub Schaltfläche1_Click ()

    MsgBox "Klick auf Schaltfläche 1", vbExclamation, _
        "Schaltfläche 1" : Beep

End Sub
```

Neben dem Zusammenfassen mehrerer Anweisungen in eine Zeile ist es auch möglich, eine einzelne Anweisung auf mehrere Zeilen aufzuteilen. Dazu muss am Ende der Zeile ein Leerzeichen gefolgt von einem Unterstrich (_) angegeben werden. Dadurch wird Access mitgeteilt, dass die Anweisung in der nächsten Zeile fortgesetzt wird. Das folgende Beispiel demonstriert, wie eine längere Anweisung in zwei Zeilen aufgeteilt wird:

Aufteilen einer
Anweisung auf
mehrere Zeilen





```
Sub Schaltfläche1_Click ()  
  
    Beep  
    MsgBox Prompt:="Klick auf Schaltfläche 1", _  
        Buttons:=48, Title:=" Schaltfläche 1"  
End Sub
```

Bei der Aufteilung einer Anweisung ist es wichtig, dass Sie diese nicht innerhalb eines Ausdrucks oder eines Schlüsselworts trennen. Der Ausdruck *"Klick auf Schaltfläche 1"* muss beispielsweise immer vollständig innerhalb einer Zeile angegeben werden. Gleiches gilt für den Namen der Anweisung *MsgBox*.

Hinweis



In den Programmlistings dieses Buches werden lange Anweisungen häufig auf mehrere Zeilen aufgeteilt, da sonst die Lesbarkeit darunter leiden würde. Wenn Sie die Prozeduren abtippen möchten, können Sie selbst entscheiden, ob Sie die entsprechenden Anweisungen in einer einzelnen Zeile oder wie abgedruckt eingeben.

Anlegen einer zweiten Schaltfläche



Genau wie vorher im *Kapitel 1 Einführung in die Makroprogrammierung* können Sie auf dem Formular eine zweite Schaltfläche anlegen, der Sie den Namen *Schaltfläche2* geben. Auch hier legen Sie wieder über die *Beim Klicken*-Eigniseigenschaft eine neue Ereignisprozedur an, die beispielsweise den folgenden Aufbau besitzen könnte:

```
Sub Schaltfläche2_Click ()  
  
    Beep  
    MsgBox "Klick auf Schaltfläche 2", 48, _  
        "Schaltfläche 2"  
  
End Sub
```

Das Formular hat nun insgesamt zwei verschiedene Ereignisprozeduren, die dem *Klick*-Ereignis der jeweiligen Schaltfläche zugeordnet sind.

Das Fokuserhalt-Ereignis

Eine Schaltfläche unterstützt neben dem *Klick*-Ereignis noch eine Reihe weiterer Ereignisse, auf die Sie ebenfalls mithilfe von Ereignisprozeduren reagieren können. Eines dieser Ereignisse ist beispielsweise das *Fokuserhalt*-Ereignis (engl. *GotFocus*). Access löst dieses Ereignis aus, wenn das betreffende Steuerelement den Fokus erhält.

Ein Steuerelement besitzt dann den Fokus, wenn es auf Ihre Tastatureingaben reagiert. Sie kennen dies z.B. von Textfeldern: In einem Access-Formular können Sie immer nur in einem bestimmten Textfeld Eingaben vornehmen. Auch Schaltflächen können den Fokus besitzen. Sie erkennen dies daran, dass der Rahmen stärker gezeichnet und der Text gestrichelt umrandet wird.



Um den Fokus auf ein anderes Steuerelement zu übertragen, genügt es, dieses mit der Maustaste anzuklicken. Per Tastatur verschieben Sie den Fokus durch Drücken der Tabulatortaste (**Tab**). Wenn nun ein bestimmtes Steuerelement – in dem Beispiel die Schaltfläche – den Fokus erhält, löst Access das *Fokuserhalt*-Ereignis aus.

Das *Fokuserhalt*-Ereignis steht bei fast allen Steuerelementtypen zur Verfügung. Eine Ausnahme stellen nur die Steuerelementtypen dar, die generell keinen Fokus haben können. Dazu gehört beispielsweise das *Rechteck*- oder das *Bezeichnungsfeld*-Steuerelement.

Das *Fokuserhalt*-Ereignis

Um das *Fokuserhalt*-Ereignis der *Schaltfläche1* mit einer Ereignisprozedur zur verknüpfen, gehen Sie auf die gewohnte Art und Weise vor. Anstelle der *Beim Klicken*-Ereigniseigenschaft verwenden Sie nun aber die *Bei Fokuserhalt*-Eigenschaft. Die vollständige Ereignisprozedur könnte den folgenden Aufbau haben:

```
Private Sub Schaltfläche1_GotFocus ()
    Beep
    MsgBox "Schaltfläche 1 erhält den Fokus", _
        vbExclamation, "Schaltfläche 1"
```

```
End Sub
```

Der *Schaltfläche1* sind nun insgesamt zwei verschiedene Ereignisprozeduren zugeordnet: Eine dient zur Reaktion auf das *Klicken*-Ereignis, die zweite zur Reaktion auf das *Fokuserhalt*-Ereignis.

Wenn Sie jetzt in die Formularansicht umschalten, zeigt Access direkt die Meldung "Schaltfläche 1 erhält den Fokus". Der Grund dafür liegt darin, dass Access beim Öffnen des Formulars automatisch der *Schaltfläche1* den Fokus zuweist. Dies führt wiederum dazu, dass Access das *Fokuserhalt*-Ereignis auslöst und die Meldung anzeigt.

Sie können nun etwas mit dem Formular experimentieren, um das Verhalten der Ereignisse kennen zu lernen. Interessant ist besonders die folgende Reaktion: Wenn die *Schaltfläche2* den Fokus hat und Sie anschließend die *Schaltfläche1* anklicken, löst Access die beiden Ereignisse *Fokuserhalt* und *Klick* aus.

Da Access niemals zwei Prozeduren parallel ausführen kann, startet es zunächst die Prozedur der *Bei Fokuserhalt*-Eigenschaft. Erst wenn diese Prozedur vollständig ausgeführt wurde, löst Access das *Klick*-Ereignis aus und startet damit die zweite Prozedur. Das zweite Meldungsfenster erscheint also erst dann, wenn Sie das erste Meldungsfenster mit *OK* geschlossen haben. Die Reihenfolge, mit der Access solche Ereignisketten ausführt, ist fest vorgegeben und in einigen Fällen wichtig für die Programmierung.

Das Gegenstück zum *Fokuserhalt*-Ereignis ist das *Fokusverlust*-Ereignis. Access löst das *Fokusverlust*-Ereignis aus, wenn ein Steuerelement den Fokus verliert. Wenn Sie mit diesem Ereignis experimentieren möchten, legen Sie einfach eine weitere Ereignisprozedur für die Eigenschaft *Bei Fokusverlust* an.

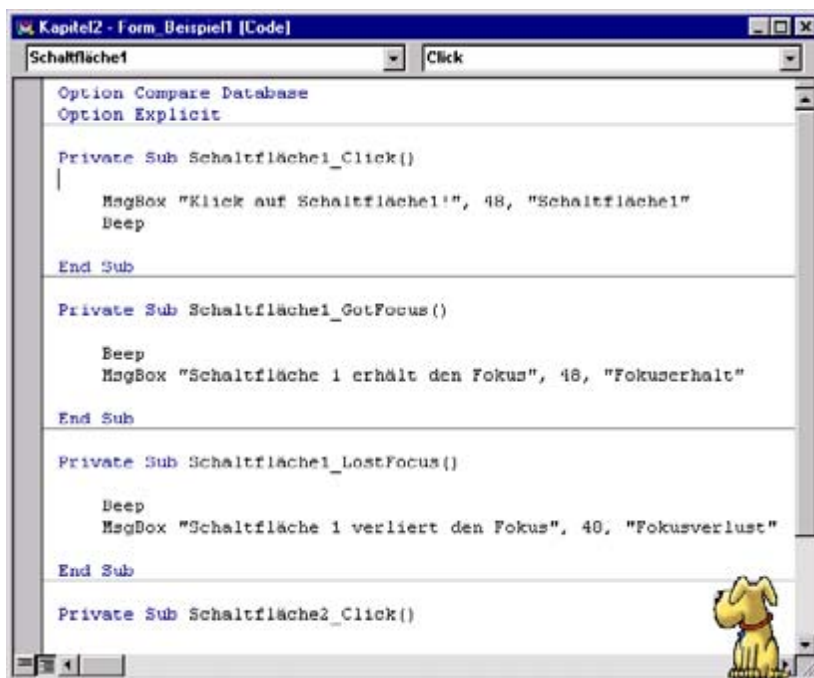
Das *Fokusverlust*-Ereignis



Das fertige Formular mit den beiden Schaltflächen finden Sie in der Beispieldatenbank *Kapitel2.mdb* unter dem Namen *Beispiel*.

Abb. 2.6

Die Ereignisprozeduren
des Formulars



Ausführen von Aktionen

Im ersten Kapitel dieses Buches haben Sie bereits die Arbeit mit Makroaktionen kennen gelernt. Auch von VBA aus lassen sich alle Makroaktionen ausführen. Ein Beispiel ist die *ÖffnenFormular*-Aktion, mit der sich ein Formular in einer beliebigen Ansicht öffnen lässt. Um eine Makroaktion von VBA aus auszuführen, stellt Access ein Objekt mit dem Namen *DoCmd* zur Verfügung. Direkt hinter dem Objektnamen kann ein Punkt und der Aktionsname zusammen mit den Aktionsargumenten angegeben werden. Wie bei VBA üblich, muss der Name allerdings in Englisch angegeben werden. Die Übersetzung von *ÖffnenFormular* lautet hierbei *OpenForm*.

Hinweis



Sobald Sie den Punkt nach der Anweisung *DoCmd* eingetippt haben, erscheint daneben eine Listbox mit allen möglichen Aktionen der *DoCmd*-Anweisung. Sie können die Anzeige dieser automatischen Hilfe mit der Option *Elemente automatisch auflisten* auf der Registerkarte *Extras > Optionen > Editor* aus- bzw. einschalten.



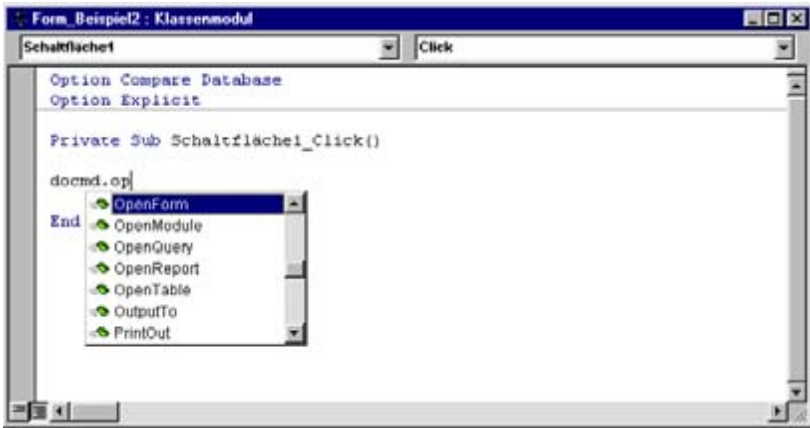


Abb. 2.7

Aktionen auswählen

Nachdem Sie die *DoCmd*-Aktion ausgewählt haben, zeigt Ihnen die Direkthilfe (sofern sie eingeschaltet ist) die korrekte Syntax an.

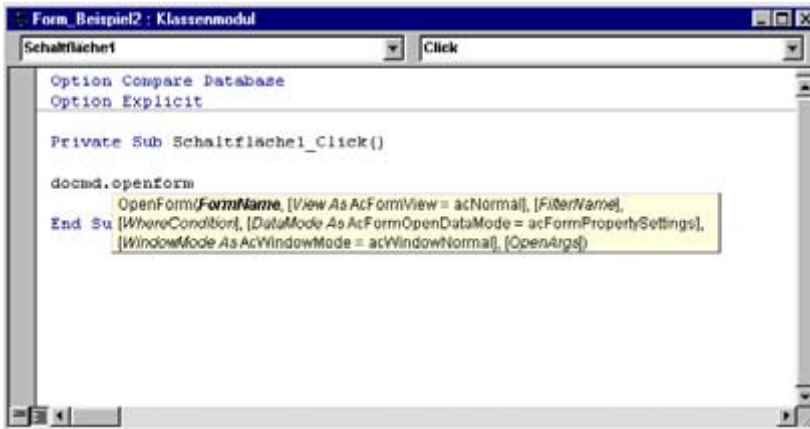


Abb. 2.8

Syntaxinformation per
Direkthilfe

Das folgende Beispiel zeigt die bereits bekannte Ereignisprozedur der *Schaltfläche1*. Anstelle der Meldung wird nun jedoch ein Formular mit dem Namen *Adressen* geöffnet. Damit diese Aktion nicht fehlschlägt, muss das Formular *Adressen* natürlich vorhanden sein.

Wenn Sie dieses Beispiel nachvollziehen möchten, finden Sie das entsprechende Formular in der Beispieldatenbank *Kapitel2.mdb*.





```
Private Sub Schaltfläche1_Click ()
```

```
    DoCmd.OpenForm "Adressen"
```

```
End Sub
```

Den ersten Parameter, über den Sie den Namen des zu öffnenden Formulars angeben, können Sie auch zusammen mit dem Parameternamen angeben. Der Parameter hat in diesem Fall den Namen *FormName*:

```
Private Sub Schaltfläche1_Click ()
```

```
    DoCmd.OpenForm FormName:="Adressen"
```

```
End Sub
```

Angabe der Aktionsargumente

Wenn Sie eine Aktion in VBA verwenden, müssen Sie die Werte der Aktionsargumente als Parameter angeben. Im obigen Beispiel haben Sie bereits einen Parameter – nämlich den Parameter *FormName* kennen gelernt, der den Namen des zu öffnenden Formulars bestimmt. Dieser erste Parameter entspricht damit dem Aktionsargument *Formularname*, das Sie von der *ÖffnenFormular*-Aktion kennen. Insgesamt stellt die *ÖffnenFormular*-Aktion sechs Aktionsargumente zur Verfügung, die innerhalb von VBA sechs Parametern entsprechen.

Die Reihenfolge, mit der Sie die einzelnen Parameter innerhalb von VBA angeben, entspricht im Allgemeinen der Reihenfolge, mit der die einzelnen Aktionsargumente im Makrofenster angegeben sind. Wenn Sie also keinen benannten Parameter verwenden möchten, bleibt Ihnen das Nachschlagen in der Dokumentation diesbezüglich weitestgehend erspart. Die Aktionsargumente der *ÖffnenFormular*-Aktion werden im Makrofenster beispielsweise in folgender Reihenfolge aufgelistet:

Formularname

Ansicht

Filtername

Bedingung

Datenmodus

Fenstermodus

Die entsprechende Befehlssyntax innerhalb von VBA hat folgenden Aufbau:

```
DoCmd.OpenForm <FormName> [,<View>] [,<FilterName>]  
[,<WhereCondition>] [,<DataMode>]  
[,<WindowMode>] [,<OpenArgs>]
```



Die Reihenfolgen stimmen also überein. Allerdings steht in VBA ein siebter Parameter *<Öffnungsargumente>* zur Verfügung, zu dem kein passendes Aktionsargument existiert. Diese zusätzlichen Parameter tauchen bei einigen Aktionen auf und dienen speziellen Funktionen, die sich nicht von Makros, sondern nur von VBA aus nutzen lassen.

Werte für Parameter ermitteln

Eine weitere Besonderheit von Aktionen stellen die Angaben dar, die für die einzelnen Parameter zulässig sind. Den Parameter *<View>* (deutsch *Ansicht*) haben Sie bereits im ersten Kapitel kennen gelernt, um die Ansicht zu bestimmen, in der Access das Formular öffnen soll. Beim Entwurf eines Makros haben Sie hierbei die Auswahl zwischen den folgenden Einstellungen:

Formular
Entwurf
Seitenansicht
Datenblatt

Genau wie bei der Auswahl des Symbols im Zusammenhang mit der *MsgBox*-Anweisung wird auch hier jede Einstellung durch einen Wert repräsentiert. Die folgende Tabelle gibt Auskunft über diese Werte:

Festlegen der
Formularansicht

Werte des Parameters *<Ansicht>*

Wert	Ansicht	Konstante
0	Formularansicht	<i>acNormal</i>
1	Entwurfsansicht	<i>acDesign</i>
2	Seitenansicht	<i>acPreview</i>
3	Datenblattansicht	<i>acFormDS</i>

Tabelle 2.2

Um nun das Formular *Adressen* in der Entwurfsansicht zu öffnen, geben Sie einfach die folgende Anweisung an:

```
DoCmd.OpenForm "Adressen", 1
```

Die Verwendung benannter Parameter ist ebenfalls zulässig:

```
DoCmd.OpenForm FormName:="Adressen", View:=1
```



Konstanten

Da der Gebrauch von Zahlen keine komfortable Angelegenheit ist, gibt es in VBA so genannte *Konstanten*, die stellvertretend für bestimmte Zahlenwerte stehen. In der Spalte *Konstante* der obigen Tabelle ist für jeden einzelnen Wert auch der Name der Konstanten angegeben. Das Öffnen des *Adressen*-Formulars in der Entwurfsansicht können Sie also auch auf folgende Weise bewerkstelligen:

```
DoCmd.OpenForm "Adressen", acDesign
DoCmd.OpenForm FormName:="Adressen", View:=acDesign
```

Hinweis



Beachten Sie, dass es sich bei den Namen der Konstanten nicht um Zeichenketten handelt. Aus diesem Grund müssen Konstantennamen immer ohne Anführungszeichen angegeben werden. Alle Konstanten, die im Zusammenhang mit Access stehen, beginnen mit dem Präfix *ac*. Im folgenden Kapitel erfahren Sie, wie Sie Konstanten selbst definieren und dabei nahezu beliebige Namen vergeben können.

Die Onlinehilfe von Access

Wie erfahren Sie nun, wann Sie einen Text oder eine Zahl angeben müssen und vor allem welche Zahlen und Konstanten Sie verwenden dürfen? Diese Informationen entnehmen Sie am besten der Onlinehilfe von Access. Die richtige Hilfeseite finden Sie, indem Sie die Einfügemarke auf den Namen der Anweisung positionieren (z.B. auf *OpenForm*) und anschließend die Taste **F1** drücken. Access startet daraufhin das Hilfesystem und wechselt zur passenden Seite. Hier werden nun die einzelnen Parameter mit allen zulässigen Werten aufgelistet.

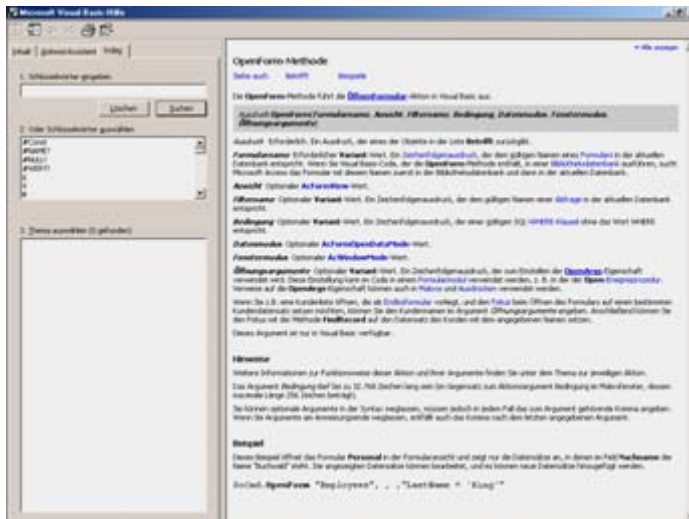
Hinweis



In der Onlinehilfe wird bei den Parametern oftmals auf die Angabe der Werte verzichtet. Stattdessen finden Sie hier nur die jeweiligen Konstantennamen.

Abb. 2.9

Die Hilfeseite der *OpenForm*-Methode



Standardparameter



Die Befehlssyntax der *OpenForm*-Aktion enthält insgesamt sieben Parameter. In dem ersten Beispiel zu dieser Aktion wurde aber nur ein einziger Parameter angegeben – nämlich der Name des zu öffnenden Formulars:

`DoCmd.OpenForm "Adressen"`

Dies ist zulässig, da die übrigen Parameter optional sind und damit nicht unbedingt angegeben werden müssen. In der Befehlssyntax erkennen Sie dies daran, dass die einzelnen Parameter in eckigen Klammern angegeben sind.

Für diejenigen Parameter, die Sie nicht angeben, verwendet Access automatisch Standardvorgaben. So wird der Parameter `<View>` standardmäßig mit dem Wert `0` belegt. Wenn Sie diesen Parameter also nicht angeben, öffnet Access das Formular automatisch in der Formularansicht. Die folgenden beiden Anweisungen sind damit gleichwertig:

`DoCmd.OpenForm "Adressen"`

`DoCmd.OpenForm "Adressen", 0`

Erst wenn Sie einem Parameter einen Wert geben möchten, der von der Standardvorgabe abweicht, ist die Angabe erforderlich. Dies ist beispielsweise beim Öffnen des Formulars in der Entwurfsansicht der Fall:

`DoCmd.OpenForm "Adressen", acDesign`

Welche Parameter welche Standardvorgaben besitzen, können Sie der Onlinehilfe zu Access entnehmen.

Das Auslassen bestimmter Parameter dient nicht nur der Schreiberleichterung. In vielen Fällen sind die Standardwerte vom aktuellen Zustand der Datenbank abhängig. Da Ihnen dieser nicht unbedingt bekannt ist und Sie ihn auch nicht verändern möchten, können Sie den jeweiligen Parameter freilassen.



Hinweis

In diesem Zusammenhang kommt es oft vor, dass Sie alle Standardwerte abgesehen von einem weiter hinter gelegenen Parameter beibehalten möchten. Das folgende Beispiel zeigt dies anhand des Formulars *Adressen*, das im Datenmodus *Nur Lesen* geöffnet werden soll. In diesem Datenmodus können an den Datensätzen des Formulars keine Änderungen vorgenommen werden. Um das Formular in diesem Modus zu öffnen, ist lediglich die Angabe des Formularnamens und des Datenmodus erforderlich. Die folgende Tabelle enthält die Konstanten zur Auswahl des Datenmodus:



Datenmodus

Konstanten des Parameters *Datenmodus*



Konstante	Beschreibung
<i>acFormAdd</i>	Hinzufügen: Es können ausschließlich neue Datensätze hinzugefügt werden.
<i>acFormEdit</i>	Bearbeiten: Datensätze können verändert, hinzugefügt und gelöscht werden.



Konstanten des Parameters *Datenmodus* (Forts.)

Konstante	Beschreibung
<i>acFormPropertySettings</i>	Die im Formular definierten Einstellungen werden beim Öffnen verwendet.
<i>acFormReadOnly</i>	Nur Lesen: Es können nur bereits vorhandene Datensätze betrachtet, aber nicht verändert werden.

Tabelle 2.3

Die Standardvorgabe für den Parameter *<Datenmodus>* ist die Konstante *acFormPropertySettings*. Wird das Formular also ohne Angabe dieses Parameters geöffnet, werden die im Formularentwurf eingestellten Werte für die Eigenschaften *Bearbeiten zulassen*, *Löschen zulassen*, *Anfügen zulassen* und *Daten eingeben* verwendet. Um das Formular nun im *Nur Lesen*-Datenmodus zu aktivieren, geben Sie die folgende Anweisung ein:

```
DoCmd.OpenForm "Adressen", , , , acFormReadOnly
```

Für jeden Parameter, der zwischen dem ersten und fünften Argument liegt, wird lediglich ein Komma angegeben, so dass Access die Standardwerte verwendet. Die Angabe der Kommas ist notwendig, damit Access die einzelnen Angaben den richtigen Parametern zuordnen kann.

Wesentlich übersichtlicher ist in diesem Fall die Verwendung benannter Parameter. Die folgende Anweisung erfüllt die gleiche Aufgabe:

```
DoCmd.OpenForm FormName:=  
"Adressen", DataMode:=acFormReadOnly
```

If...Then...Else

Was Ihnen nun noch zur Umsetzung von Makros in eine VBA-Prozedur fehlt, sind die Bedingungen. In Makros haben Sie Bedingungen dazu verwendet, bestimmte Aktionen auszuführen oder nicht auszuführen. In VBA wird so etwas über die *If...Then*-Anweisung erledigt. Bei der *If...Then*-Anweisung handelt es sich um eine Anweisung, die aus den beiden Schlüsselwörtern *If* und *Then* besteht. Die Befehlsyntax gibt Auskunft über die Verwendung der Anweisung:

```
If <Bedingung> Then <Anweisung>
```

Für *<Bedingung>* können Sie in der Regel den gleichen Ausdruck verwenden, den Sie beim Makroentwurf angegeben haben. Die einzige Voraussetzung ist, dass die Bedingung entweder *Wahr* oder *Falsch* zum Ergebnis hat. In VBA werden diese beiden Zustände durch die beiden Schlüsselwörter *True* und *False* repräsentiert.



Hinter dem Schlüsselwort *Then* kann in derselben Zeile eine einzelne Anweisung angegeben werden, die Access nur dann ausführt, wenn die Bedingung den logischen Wert *True* zum Ergebnis hat. Das folgende Beispiel öffnet das Formular *Adressen*, wenn das heutige Datum der 8.10.2001 ist:

```
If Date = #10/08/01# Then DoCmd.OpenForm "Adressen"
```

Im dem Beispiel wird eine Funktion mit dem Namen *Date* verwendet, die das aktuelle Tagesdatum zum Ergebnis hat. Informationen zu dieser Funktion und allen weiteren Datumsfunktionen erhalten Sie im folgenden *Kapitel 3 Die Programmiersprache VBA*. In diesem Kapitel erfahren Sie außerdem, wie Bedingungen prinzipiell aufgebaut sein müssen. Eine weitere Besonderheit betrifft die Angabe von Datumswerten im Programmcode, die im amerikanischen Format angegeben werden müssen. Mehr dazu erfahren Sie ebenfalls in Kapitel 3.



Hinweis

Ausführung mehrerer Anweisungen

Im obigen Beispiel wird die Ausführung einer einzelnen Anweisung von der Bedingung abhängig gemacht. Wenn Sie allerdings mehrere Anweisungen ausführen möchten, müssen Sie diese zu einem Anweisungsblock zusammenfassen und das Ende dieses Blocks gesondert kennzeichnen. Zu diesem Zweck stellt Access das *End If*-Schlüsselwort zur Verfügung. Die erweiterte *If...Then*-Anweisung hat folgende Befehlssyntax:

```
If <Bedingung> Then
    <Anweisungsblock>
End If
```

Die Befehlssyntax
der *If...Then*-Anweisung

Das folgende Beispiel führt insgesamt drei Anweisungen aus, wenn die Bedingung erfüllt ist:

```
If Date = #10/08/01# Then
    Beep
    MsgBox "Das Adressen-Formular wird geöffnet!"
    DoCmd.OpenForm "Adressen"
End If
```

Ist die Bedingung nicht erfüllt, setzt Access die Programmausführung direkt nach der *End If*-Anweisung fort.

Eine weitere Ergänzung ist das Schlüsselwort *Else*. Es ermöglicht die alternative Ausführung eines Anweisungsblocks, falls die Bedingung nicht erfüllt ist:

Else





```
If <Bedingung> Then  
    <Anweisungsblock1>  
  
Else  
    <Anweisungsblock2>  
  
End If
```

<Anweisungsblock1> wird ausgeführt, wenn die Bedingung den Wert *True* zum Ergebnis hat. Dagegen wird <Anweisungsblock2> nur dann ausgeführt, wenn die Bedingung nicht erfüllt ist, also den logischen Wert *False* zum Ergebnis hat. Es folgt ein Beispiel:

```
If Date = #10/08/01# Then  
    Beep  
    MsgBox "Das Adressen-Formular wird geöffnet!"  
    DoCmd.OpenForm "Adressen"  
  
Else  
    Beep  
    MsgBox "Das Adressen-Formular wird nicht geöffnet!"  
  
End If
```

Hinweis



Die Verwendung benannter Parameter ist bei Konstrukten wie dem *If...Then...Else*-Konstrukt nicht zulässig. Die <Bedingung> dürfen Sie also beispielsweise nicht mit dem Parameternamen angeben.

Wie geht es weiter?

In diesem Kapitel haben Sie nun den grundlegenden Umgang mit VBA kennen gelernt. Mithilfe des *DoCmd*-Objekts und etwas Recherche in der Onlinehilfe von Access können Sie alle Makroaktionen von VBA aus ausführen. Zusammen mit dem *If...Then...Else*-Konstrukt ist es sogar möglich, die Funktion eines Makros komplett auf eine VBA-Prozedur zu übertragen.

Im folgenden Kapitel lernen Sie zunächst einmal den grundlegenden Umgang mit VBA sowie wichtige Funktionen und Anweisungen kennen, die bei der Programmierung in Access benötigt werden. Dazu zählt beispielsweise auch die Datumsfunktion *Date*, die Sie bereits in diesem Kapitel kennen gelernt haben.

Drei weitere Kapitel beschäftigen sich anschließend mit den Schnittstellen zwischen der Programmiersprache und den Datenbankobjekten. Dazu gehört beispielsweise der Zugriff auf Formulare, Steuerelemente sowie auf die Daten von Tabellen.

Mit dem praktischen Einsatz von VBA beschäftigen sich schließlich die *Kapitel 7 Steuerelemente programmieren*, *Kapitel 8 Berichte programmieren* und *Kapitel 9 Objektorientierte Programmierung*.





Die Programmiersprache VBA



Das Modulfenster	55
Module	67
Prozeduren	69
Ausdrücke	78
Variablen	108
Konstanten	136
Parametervariablen	137
Kontrollstrukturen	149
Fehlersuche und Fehlerbehandlung	160
Fehlersuche	165
Kompilieren des Programmcodes	175
Datenbankverweise	179
Ereignisorientierte Programmierung	185





Im vorhergehenden haben Sie bereits den grundlegenden Umgang mit VBA kennen gelernt, um Makros ohne großen Aufwand in VBA umzusetzen. Dieses Kapitel beschäftigt sich nun mit denjenigen Eigenschaften von VBA, die weit über den Funktionsumfang von Makros hinausgehen.

VBA bietet einen sehr großen Funktionsumfang. Waren es bei Makros gerade noch 54 Aktionen, die Sie nutzen können, stehen Ihnen bei VBA mehrere Hundert Anwendungen, Funktionen und Methoden zur Verfügung. Darüber hinaus können Sie mithilfe des vorhandenen Funktionsumfangs neue Funktionen in Form von selbst-erstellten Prozeduren hinzufügen, die oft benötigte Aufgaben erledigen.

Was ist VBA?

VBA ist eine Programmiersprache, die stark an die bekannte Programmiersprache Basic angelehnt ist. Basic zeichnet sich dadurch aus, dass es sehr leicht zu erlernen ist und damit schnell zu einem Ergebnis führt. Für den professionellen Einsatz ist Basic allerdings weniger geeignet, da es beispielsweise keine guten Möglichkeiten zur Strukturierung bietet und sich längere Programme daher sehr schlecht überblicken lassen. Aus diesem Grund haben die Entwickler von Access die Programmiersprache VBA um einige Funktionen erweitert, die aus Hochsprachen wie C/C++ oder Pascal bekannt sind. Das wichtigste Strukturierungskonzept dieser Hochsprachen sind die Prozeduren, in denen kleine überschaubare Probleme gelöst werden. Solche Programmiersprachen werden daher auch als *prozedural* bezeichnet. Wie Sie im *Kapitel 2 Schnellstart VBA* gesehen haben, ist auch VBA eine solche prozedurale Programmiersprache.

Die Programmiersprache von Access besteht aus drei Komponenten, die in der Programmiersprache VBA zusammengefasst werden:

- allgemeine Visual-Basic-Sprachelemente
- Access-spezifische Sprachanteile
- Datenzugriffsobjekte:
DAO (*Data Access Objects*) bzw.
ADO (*Active(X) Data Objects*)

Die Grundlage der Programmiersprache stellen die allgemeinen Visual-Basic-Sprachelemente zur Verfügung. Hieraus stammen beispielsweise Konstrukte wie die *If...Then...Else*-Anweisung oder allgemeine Funktionen (z. B. die *Date*-Funktion zur Ermittlung des aktuellen Datums). Diese sind nicht nur in Access XP, sondern beispielsweise auch in Excel oder Visual Basic 4/5/6 integriert. Wenn Sie also einmal die Grundelemente von Visual Basic kennen, werden Sie sich in den Programmiersprachen anderer VBA-kompatibler Windows-Anwendungen schnell zurechtfinden.

Die zweite Komponente stellt Funktionen zur Verfügung, die speziell auf Access zugeschnitten sind. Diese Komponente enthält beispielsweise das *DoCmd*-Objekt, mit dem Sie Makroaktionen ausführen können, oder die *Forms*-Auflistung zum Zugriff auf Formulare.





Die dritte und letzte Komponente erledigt schließlich den Zugriff auf Daten. Sie erlaubt beispielsweise den Zugriff auf Tabellen oder Abfragen. Auch diese Komponente wird nicht nur in Access, sondern beispielsweise auch in Visual Basic eingesetzt.

Alle drei Komponenten sind sehr gut integriert, so dass Sie als Programmierer in der Regel nichts von dieser Aufteilung bemerken.

In diesem Kapitel werden einige Listings vorgestellt, die Ihnen die Elemente der Programmiersprache praxisnah vorstellen. Alle Prozeduren dieses Kapitels finden Sie in der Beispieldatenbank *Kapitel3.mdb*!



Hinweis

Das Modulfenster

Um effektiv mit VBA arbeiten zu können, sind grundlegende Kenntnisse über die Techniken zur Eingabe und zur Bearbeitung von Programmcode erforderlich. Damit beschäftigt sich dieser Abschnitt.

Im zweiten Kapitel dieses Buches haben Sie bereits mit dem Modulfenster gearbeitet. Dieses Fenster ist im Prinzip nichts anderes als ein Texteditor, der um verschiedene Sonderfunktionen zur Unterstützung der Programmcodeeingabe erweitert wurde. Es wird in der VBA-Entwicklungsumgebung angezeigt, in der neben dem eigentlichen Modulfenster weitere Funktionen für die Bearbeitung der Formulare und Programmteile enthalten sind.

Die VBA-Entwicklungsumgebung

Wenn Sie das Modulfenster aufrufen, erscheint die VBA-Entwicklungsumgebung auf dem Bildschirm. Dieses Fenster enthält je nach Einstellung eine Reihe von Unterfenstern, die im Menü *Ansicht* eingeblendet werden können:

Projekt-Explorer

Hier werden die Formulare und Module des Projekts aufgelistet, deren Codes und Eigenschaften Sie leicht auswählen können.

Eigenschaften

Hier sind die Eigenschaften des im Projekt-Explorer gewählten Objekts (z. B. Formular) aufgeführt und können direkt geändert werden.

Direktfenster

Hier können Sie jederzeit Anweisungen eingeben oder aus dem Programm heraus Ausgaben vornehmen, etwa für die Anzeige von Variablenwerten.





Lokal-Fenster

Dieses Fenster dient zur Anzeige der aktuellen Variablenwerte während der Ausführung. Sie können hier beispielsweise bei einer Programmunterbrechung die Zustände Ihres Programms ablesen, um etwa Fehlern auf die Spur zu kommen.

Überwachungsfenster

Hier werden die Überwachungsausdrücke angezeigt, die im Menü *Debuggen* unter *Überwachung hinzufügen* definiert werden können. Auf diese Weise können Sie wichtige Zustände jederzeit im Blick behalten.

Den praktischen Einsatz dieser Hilfsmittel zur Programmentwicklung werden Sie im Verlauf dieses Kapitels noch näher kennen lernen. Kehren wir aber zunächst zurück zur eigentlichen Bearbeitung des Programmcodes im Modulfenster.

Das Bearbeiten von Prozeduren im Modulfenster

Um eine bestimmte Prozedur zu bearbeiten, stellt Access im Kopf des Modulfensters die zwei Kombinationsfelder *Objekt* bzw. *Prozedur* zur Verfügung.

Im linken Kombinationsfeld *Objekt* finden Sie stets den Eintrag (*Allgemein*). Wenn dieser Eintrag ausgewählt ist, können Sie über das rechte Kombinationsfeld *Prozedur* die einzelnen Prozeduren über ihre Namen auswählen. Nach der Auswahl erscheint die Prozedur an oberster Position des Modulfensters. Dies erspart Ihnen bei umfangreichen Modulen die Suche nach einer bestimmten Prozedur. Alternativ dazu ist es möglich, die einzelnen Prozeduren über die Tastenkombination **Strg**+**↑** oder **Strg**+**↓** nacheinander auszuwählen.

Bei bestimmten Modulen – den an Formulare oder Berichte gebundenen Klassenmodulen – werden im Kombinationsfeld *Objekt* neben dem Eintrag (*Allgemein*) alle Steuerelemente des Formulars bzw. Berichts aufgelistet. Mit einem solchen Modul haben Sie bereits bei der Eingabe von Ereignisprozeduren im *Kapitel 2 Schnellstart VBA* gearbeitet.

Sobald Sie ein Steuerelement auswählen, zeigt Access im Kombinationsfeld *Prozedur* alle Ereignisse an, die von dem Steuerelement ausgelöst werden können. Ereignisse, denen bereits eine Ereignisprozedur zugewiesen wurde, werden hierbei in fetter Schriftart angezeigt. Durch Auswahl eines Ereignisses zeigt Access die entsprechende Ereignisprozedur im Modulfenster an. Sollten Sie ein noch nicht verknüpftes Ereignis auswählen, legt Access eine neue Ereignisprozedur an.

Hinweis



Um die Ereignisse eines Formulars oder Berichts zu bearbeiten, wählen Sie im Kombinationsfeld *Objekt* den Eintrag *Form* bzw. *Report* aus.

Wenn Sie im Kombinationsfeld *Objekt* den Eintrag (*Allgemein*) und im Kombinationsfeld *Prozedur* den Eintrag (*Deklarationen*) auswählen, zeigt Access im Modulfenster den Deklarationsteil an. Dieser Bereich ist für allgemeine Einstellungen reserviert, die das Modul betreffen.



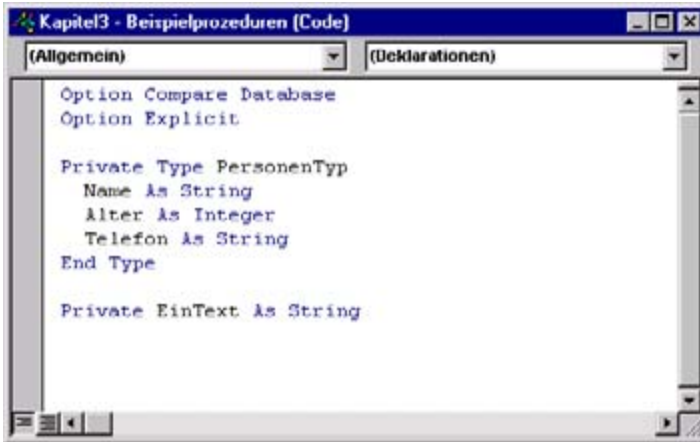
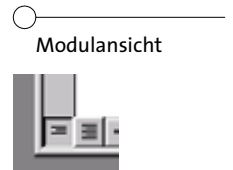


Abb. 3.1

Ein Modulfenster

Standardmäßig aktiviert Access die volle Modulansicht, in der alle Prozeduren eines Moduls untereinander aufgelistet werden. Bei Bedarf können Sie in die Prozeduransicht umschalten, in der Access jeweils nur eine einzelne Prozedur anzeigt. Die Umschaltung zwischen den beiden Ansichten bewerkstelligen Sie über die beiden Schaltflächen *Vollständige Modulansicht* und *Prozeduransicht*, die Sie in der linken unteren Ecke des Modulfensters finden.



Im Register *Editor* des *Optionen*-Dialogs (Menüpunkt *Extras > Optionen*) haben Sie zusätzlich die Möglichkeit, die von Ihnen bevorzugte Ansicht beim Öffnen eines Moduls voreinzustellen. Die entsprechende Option trägt den Namen *Standardmäßig ganzes Modul anzeigen*. Wenn Sie die einzelnen Prozeduren zusätzlich durch eine horizontale Trennlinie unterteilen möchten, markieren Sie im *Optionen*-Dialog die Option *Prozedurtrennlinie*.



In einigen Fällen ist es vorteilhaft, zwei verschiedene Prozeduren gleichzeitig im Modulfenster darzustellen. Zu diesem Zweck können Sie das Modulfenster zweiteilen. Dies erledigen Sie entweder über den Menüpunkt *Fenster > Teilen* oder durch Verschieben des Fensterteilers mit der Maus. Der Fensterteiler ist als kleines graues Rechteck am rechten Fensterrand direkt über den vertikalen Bildlaufleiten sichtbar. Um ihn zu verschieben, bewegen Sie den Mauszeiger auf das Rechteck, betätigen die linke Maustaste und bewegen den Mauszeiger bei gedrückt gehaltener Maustaste nach unten. Nach dem Lösen der Maustaste ist das Fenster in zwei Abschnitte geteilt. Durch das Verschieben des Fensterteilers kann die Größe der beiden Abschnitte auch nachträglich verändert werden.



Die Auswahl der darzustellenden Prozedur über die Kombinationsfelder *Objekt* und *Prozedur* bezieht sich immer auf den Abschnitt, in dem sich die Einfügemarke momentan befindet.



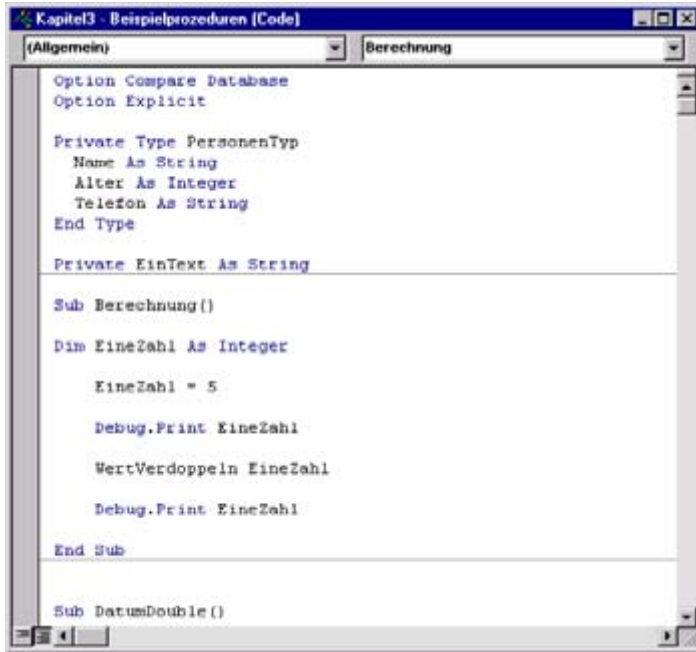


Abb. 3.2

Ein Modulfenster in der vollen Modulansicht

Wird der Menüpunkt *Fenster > Teilen* ein zweites Mal ausgewählt oder der Fenster- teiler an die oberste oder unterste Position verschoben, hebt Access die Teilung des Modulfensters wieder auf.

Arbeiten mit dem Modulfenster

Die Eingabe von Programmcode erfolgt auf die gleiche Art und Weise, wie Sie es von einem Texteditor gewohnt sind. Die senkrechte Linie markiert die Einfügemarke, an der alle Texteingaben ausgegeben werden. Bei der Eingabe von Text verschiebt Access alle Zeichen, die sich rechts der Einfügemarke befinden, nach rechts. Dieser Modus wird daher auch *Einfügemodus* genannt. Haben Sie dagegen den Überschreibmodus über die **[Einfg]**-Taste aktiviert, überschreibt Access die nachfolgenden Zeichen. Die Einfügemarke verwandelt sich in diesem Fall in ein schwarzes Rechteck. Um den Überschreibmodus zu deaktivieren, drücken Sie die **[Einfg]**-Taste ein zweites Mal.

Positionieren der Einfügemarke

Die Einfügemarke kann entweder über die Pfeiltasten (**[←]**, **[↓]**, **[↑]**, **[→]**) oder durch einen Klick mit der Maus verschoben werden. Um eine neue Zeile einzufügen, bewegen Sie die Einfügemarke an das Ende der vorhergehenden Zeile und drücken die **[↵]**-Taste. Zum Löschen von Zeichen verwenden Sie die Tasten **[←]** und **[Entf]**. Die **[←]**-Taste löscht das Zeichen, das sich links von der Einfügemarke befindet, die **[Entf]**-Taste das Zeichen rechts der Einfügemarke. Um eine leere Zeile zu entfernen, positionieren Sie die Einfügemarke an den Anfang dieser Zeile und drücken die **[↵]** oder **[Entf]**-Taste.



Die Tasten **Bild ↑** und **Bild ↓** bewirken ein seitenweises Verschieben der Einfügemarke und damit des Textausschnitts nach oben bzw. nach unten. Eine ähnliche Wirkung lässt sich auch durch das Betätigen der vertikalen Bildlaufleiste am rechten Modulfensterrand erreichen. Da auch die Länge einer Zeile die Breite des Modulfensters überschreiten kann, ermöglicht die horizontale Bildlaufleiste ein horizontales Verschieben des Textes.

Verschieben des Textausschnitts

Ganze Textpassagen können Sie per Drag & Drop verschieben. Markieren Sie dazu die zu verschiebende Textstelle und ziehen Sie diese bei gedrückter linker Maustaste an die neue Position. Wenn Sie die Maustaste loslassen, wird der Text eingefügt. Wenn Sie dabei die **Strg**-Taste gedrückt halten, wird die Markierung kopiert. Diese Funktionalität können Sie in der Registerkarte *Editor* des *Optionen*-Dialogs deaktivieren.

Textbearbeitung per Drag & Drop

Befindet sich die Einfügemarke innerhalb des Modulfensters auf einem Prozedurnamen, einer Variablen oder einem Schlüsselwort, erscheint die entsprechende Prozedur bzw. die Deklarationsanweisung der Variablen nach dem Drücken der Tastenkombination **Strg**+**F2** direkt im Modulfenster. Bei Schlüsselwörtern wird die Definition im Objektkatalog angezeigt. Alternativ kann auch der Menüpunkt *definition* aus dem Kontextmenü aufgerufen werden. Mit **Strg**+**Strg**+**F2** bzw. dem Menüpunkt *Letzte Position* können Sie danach wieder zu der Stelle im Code springen, von der Sie zur Definition gesprungen sind.

Hinweis

Bearbeiten von Programmcode

Wie es bei Windows-Anwendungen üblich ist, können Sie auch vom Programmeditor aus von der Zwischenablage Gebrauch machen. In dem Menü *Bearbeiten* stehen hierfür die bekannten Menüpunkte zur Verfügung:

Funktionen der Zwischenablage

Menüpunkt	Tastatur	Funktion
Ausschneiden	Strg + X , Strg + Entf	Löscht den markierten Textbereich in die Zwischenablage.
Kopieren	Strg + C , Strg + Einfg	Kopiert den markierten Textbereich in die Zwischenablage.
Einfügen	Strg + V , Strg + Einfg	Fügt den Inhalt der Zwischenablage an der Position der Einfügemarke ein.
Löschen	Entf	Löscht den markierten Textbereich.

Tabelle 3.1



Markieren eines Textbereichs

Zum Markieren eines bestimmten Textes bewegen Sie den Mauszeiger an den Anfangspunkt des zu markierenden Textbereichs, drücken die linke Maustaste und bewegen den Mauszeiger bei gedrückt gehaltener Maustaste an das Ende des Bereichs. Nachdem Sie die Maustaste gelöst haben, wird der markierte Textbereich invertiert dargestellt.

Alles markieren

Um den gesamten Modultext zu markieren, müssen Sie sich in der vollständigen Modulansicht befinden. Wählen Sie anschließend den Menübefehl *Bearbeiten > Alles auswählen* bzw. drücken Sie die Tastenkombination **(Strg)+[A]**. Wenn Sie sich in der Prozeduransicht befinden, wird nur die angezeigte Prozedur markiert.

Tipp



Alternativ dazu lässt sich ein Textbereich auch über die Tastatur markieren. Bewegen Sie hierfür die Einfügemarke an den Anfang des Textbereichs. Über die Tastenkombination **(Strg)+[↑]+[→]** bzw. **(Strg)+[↑]+[←]** lässt sich die Markierung von der aktuellen Position der Einfügemarke aus um jeweils ein Wort nach rechts bzw. links erweitern. Die Tastenkombination **(Strg)+[↑]+[Pos 1]** bzw. **(Strg)+[↑]+[Ende]** erweitert die Markierung bis zum Anfang bzw. Ende der aktuellen Prozedur.

Einrücken von Programmteilen

Um den Programmcode möglichst übersichtlich zu gestalten, ist es normalerweise üblich, bestimmte Programmzeilen nach rechts einzurücken. Zu diesem Zweck bietet sich die Tabulatortaste (**[Tab]**) an. Beim Drücken der Tabulatortaste wird der hinter der Einfügemarke folgende Text automatisch an die nächste Tabulatorposition verschoben. Der Abstand der einzelnen Tabulatorpositionen kann im Register *Editor* des *Optionen*-Dialogs angepasst werden. Die Voreinstellung für diese Option ist vier Zeichen.

Hinweis



Möchten Sie mehrere Zeilen gleichzeitig einrücken, müssen Sie diese Zeilen zuerst markieren und danach die **[Tab]**-Taste drücken. Um die Einrückung der markierten Zeilen rückgängig zu machen, betätigen Sie die Tastenkombination **[↑]+[Tab]**.

Das Einrücken kann alternativ auch über die Menüpunkte *Einzug vergrößern* und *Einzug verkleinern* des Menüs *Bearbeiten* oder die entsprechenden Symbole in der Symbolleiste erfolgen.

Wenn Sie eine neue Programmzeile einfügen, rückt Access diese Zeile automatisch passend zur vorhergehenden Zeile ein. Dies können Sie bei Bedarf im *Optionen*-Dialog (Menüpunkt *Extras > Optionen*) im Register *Editor* unterbinden. Löschen Sie hier die Markierung *Automatisch Einzug vergrößern*.

Automatische Eingabehilfen

Automatische Syntaxüberprüfung

Bei der Eingabe von Programmtext in das Modulfenster führt Access automatisch eine Syntaxkontrolle durch, sobald Sie die Einfügemarke in eine andere Zeile bewegen. Durch die Syntaxkontrolle werden Sie auf Fehler im Programmcode aufmerksam gemacht. Access kann auf diese Weise jedoch nur einige grobe Fehler ausfindig machen. Besonders die Fehler, die sich erst bei der Ausführung des Programms bemerkbar machen, werden übersehen. Um die automatische Syntaxkontrolle aus-



zuschalten, wählen Sie im *Optionen*-Dialogfenster (Menüpunkt *Extras > Optionen*) das Register *Editor* aus. Hier deaktivieren Sie nun die Option *Automatische Syntaxüberprüfung*.

Access färbt den Programmtext im Modulfenster unterschiedlich ein, was Ihnen die Orientierung erleichtert. So stellt Access beispielsweise bekannte Schlüsselwörter in blauer Farbe und Kommentare in Grün dar. Fehlerhafte Programmzeilen erscheinen in Rot. Die verwendeten Farben und auch die Schriftart und -größe verändern Sie bei Bedarf im Register *Editor* des *Optionen*-Dialogs.

Farbkodierungen

Wenn Sie einmal Hilfe zu einem Schlüsselwort (z.B. einer Anweisung oder einer Methode) benötigen, positionieren Sie einfach die Einfügemarke auf den betreffenden Text. Anschließend drücken Sie die Taste **F1**, um das kontextsensitive Hilfesystem zu aktivieren. Access zeigt automatisch die zum Schlüsselwort passende Hilfe-seite an, so dass Sie ohne Umwege zu den gewünschten Informationen gelangen.

Hilfe anfordern

Auch erfahrene VBA-Programmierer konsultieren häufig die Onlinehilfe, um beispielsweise den Namen einer Konstanten oder die Reihenfolge von Parametern zu ermitteln. Oft erweist sich dies jedoch als zu umständlich. Access stellt daher zwei Funktionen namens *Automatische QuickInfo* und *Elemente automatisch auflisten* zur Verfügung, die Sie bereits im vorhergehenden Kapitel kennen gelernt haben. Diese Funktionen machen das Ermitteln von Konstanten und Parametern zum Kinderspiel.

Automatische QuickInfo und *Elemente automatisch auflisten*

Sobald Sie im Modulfenster einen Prozedur-, Methoden- oder Eigenschaftsnamen gefolgt von einem Punkt, einer Klammer oder einem Leerzeichen eingeben, zeigt Access unterhalb der aktuellen Programmzeilen die komplette Syntax des Elements als QuickInfo an. Sie erhalten so einen schnellen Überblick über die Namen und die Reihenfolge der einzelnen Parameter.

Automatische Direkthilfe

Damit Sie bei vielen Parametern nicht die Übersicht verlieren, markiert Access den aktuellen Parameter mit einer fetten Schriftart. Access achtet hierbei darauf, dass diese Anzeige immer synchron zu Ihrer Eingabe ist.

Noch einen Schritt weiter geht die Funktion *Elemente automatisch auflisten*: Sie listet – falls möglich – alle zu einem Parameter verfügbaren Elemente auf. Dies können beispielsweise alle passenden Konstanten oder (bei einem Objekt), die Namen aller Eigenschaften und Methoden sein.

Elemente automatisch auflisten

Ein Beispiel hierfür ist das *DoCmd*-Objekt. Sobald Sie im Modulfenster den Text

`DoCmd.`

eingetragen haben, erscheint eine Liste aller Methoden dieses Objekts. Die Auswahl eines Elements erledigen Sie mit den Pfeiltasten Ihrer Tastatur (**↓**, **↑**) gefolgt von einem Leerzeichen (**␣**). Access fügt nun den Namen der Methode mit anschließendem Leerzeichen in das Modul ein.



Wenn Sie direkt nach der Eingabe eines Parameters den nächsten Parameter eingeben möchten, schließen Sie die Auswahl mit der -Taste ab. Access ergänzt den Programmcode durch das ausgewählte Listenelement und ein Komma. Außerdem erscheint direkt eine Auswahlliste für den nächsten Parameter.

Die Werte einiger Parameter setzen sich aus mehreren Konstanten zusammen. Dies ist beispielsweise beim zweiten Parameter der *MsgBox*-Funktion der Fall, über den Sie z. B. die verfügbaren Schaltflächen sowie das angezeigte Symbol auswählen. Der Ausdruck

```
vbYesNo + vbQuestion
```

bedeutet beispielsweise, dass Access die Schaltflächen *Ja* und *Nein* sowie ein Fragezeichensymbol anzeigt. Zur Eingabe solcher Konstantensummen wählen Sie zunächst die erste gewünschte Konstante (in dem Beispiel *vbYesNo*) aus der Liste aus und drücken anschließend die -Taste. Access öffnet die Liste nun ein zweites Mal, so dass Sie die nächste Konstante auswählen können. Die Eingabe des kompletten Parameters beenden Sie wie üblich mit der .

Möchten Sie die Eingabe der kompletten Anweisung mit der Auswahl des nächsten Elements abschließen, drücken Sie anstelle der Leertaste die -Taste.

Hinweis



Bei bestimmten Parametern zeigt Access generell keine Auswahlliste an. Dies trifft beispielsweise für Textparameter zu. Geben Sie in diesem Fall den gewünschten Text samt Anführungszeichen auf die herkömmliche Art direkt in das Modulfenster ein.

Abb. 3-3

Die automatischen Eingabehilfen des Modulfensters



Hinweis



Sollte Sie eine der beiden Eingabehilfen stören, können Sie diese im *Optionen*-Dialog (Menüpunkt *Extras > Optionen*) deaktivieren. Zu diesem Zweck enthält das Register *Modul* die beiden Optionen *Automatische QuickInfo* und *Automatische Daten-Tips*.

Das Kontextmenü des Modulfensters sowie das Menü *Bearbeiten* enthalten einige zusätzliche Menüpunkte, die Ihnen die Eingabe von Programmcode weiter erleichtern. Alle Befehle beziehen sich immer auf den Bezeichner, auf dem momentan die Einfügemarke positioniert ist. Die folgende Tabelle gibt Ihnen eine Übersicht der Befehle:



Eingabehilfe über das Kontextmenü

Option	Beschreibung
<i>Eigenschaften > Methode auflisten</i>	Listet alle alternativen Elemente zu dem Bezeichner auf. Neben Eigenschaften und Methoden können dies beispielsweise auch Konstanten sein.
<i>Konstanten anzeigen</i>	Zeigt zu einer Konstante eine Liste aller alternativen Konstanten an.
<i>QuickInfo</i>	Zeigt die Deklaration des Bezeichners an.
<i>Parameterinfo</i>	Zeigt die komplette Syntax der aktuellen Anweisung an.
<i>Wort vervollständigen</i>	Ermöglicht Ihnen die Vervollständigung eines Bezeichners durch Anzeige einer Liste.

Tabelle 3.2

Suchen nach Zeichenfolgen

Access stellt über das Menü *Bearbeiten* Funktionen zur Verfügung, die das automatische Suchen bzw. Suchen und Ersetzen von Programmcode erleichtern. Nach Auswahl des Menüpunkts *Suchen* zeigt Access das gleichnamige Dialogfenster an. In das Textfeld *Suchen nach* können Sie eine beliebige Zeichenfolge als Suchbegriff eingeben, nach dem im Programmcode gesucht werden soll. Standardmäßig trägt Access in dieses Feld immer das Wort des Programmcodes ein, auf dem sich die Einfügemarke beim Aufruf des Dialogfensters befindet.

Abb. 3.4

Das Dialogfenster
Bearbeiten > Suchen



Die folgende Tabelle enthält eine Beschreibung aller Optionen des Dialogfensters:

Die Optionen des Dialogfensters *Suchen*

Option	Beschreibung
<i>Suchen in</i>	Über diese Sektion lässt sich die Suche auf die aktuelle Prozedur, das komplette Modul, die komplette Datenbank oder den momentan markierten Programmcodebereich beschränken.
<i>Suchrichtung</i>	Die Richtung, in der von der aktuellen Position der Einfügemarke an gesucht wird. Bei der Einstellung <i>Alle</i> durchsucht Access den kompletten Suchbereich.
<i>Nur ganzes Wort suchen</i>	Wenn diese Option markiert ist, vergleicht Access nur komplette Wörter mit dem Suchbegriff.
<i>Groß-/Kleinschreibung beachten</i>	Bestimmt, ob Access die Groß- und Kleinschreibung bei der Suche beachtet.
<i>Mit Mustervergleich</i>	Sucht nach bestimmten Mustern im Programmtext. Sobald das Kontrollkästchen markiert ist, werden einige Zeichen im Suchbegriff nicht als normale Zeichen, sondern als Platzhalterzeichen interpretiert. So würde der Suchbegriff "Fun*n" alle Zeichenfolgen suchen, die mit "Fun" beginnen und mit dem Buchstaben "n" enden wie z. B. "Funktionen".

Tabelle 3.3

Starten der Suche

Die Suche wird von der aktuellen Position der Einfügemarke aus gestartet. Ist der Suchlauf erfolgreich, stellt Access die gefundene Zeichenfolge im Modulfenster markiert dar. Um die Suche mit dem gleichen Suchbegriff zu wiederholen, klicken Sie auf die Schaltfläche *Weitersuchen*. Zwischen den einzelnen Suchvorgängen können Sie zu anderen Access-Fenstern wechseln, ohne das aktive Fenster zu schließen.

Weitersuchen

Wenn Sie das offene *Suchen*-Dialogfenster stört, können Sie es nach Eingabe der Kriterien und dem ersten Betätigen der *Weitersuchen*-Schaltfläche schließen und die Suche über den Menübefehl *Bearbeiten* > *Weitersuchen* oder noch komfortabler durch Drücken der Funktionstaste **F3** fortsetzen.



Suchen und Ersetzen

Möchten Sie eine bestimmte Zeichenfolge im Programmcode automatisch durch eine andere Zeichenfolge ersetzen lassen, wählen Sie den Menüpunkt *Bearbeiten > Ersetzen* aus. Alternativ können Sie das Dialogfenster auch über die *Ersetzen*-Schaltfläche des Fensters *Suchen* aktivieren.

Das Dialogfenster *Ersetzen* ist im Gegensatz zum Dialogfenster *Suchen* um das Textfeld *Ersetzen durch* erweitert.



Abb. 3.5

Das Dialogfenster
Bearbeiten > Ersetzen

In das Textfeld *Suchen nach* geben Sie die Zeichenfolge ein, die durch die Zeichenfolge im Textfeld *Ersetzen durch* ersetzt werden soll. Die Zeichenzahl der beiden Eingaben braucht hierbei nicht übereinzustimmen. Über die Schaltfläche *Weitersuchen* starten Sie die Suche nach der ersten passenden Textpassage. War die Suche erfolgreich, stellt Access den gefundenen Text im Modulfenster invertiert dar. Mit der Schaltfläche *Ersetzen* weisen Sie Access an, den gefundenen Text zu ersetzen. Auf diese Weise können Sie für jede Textpassage individuell bestimmen, ob Access diese ersetzen soll.

Die Schaltfläche *Alle ersetzen* ermöglicht es, alle Textpassagen ohne vorherige Rückfrage zu ersetzen. Diese Funktion ist mit allergrößter Vorsicht zu genießen, da auf diese Weise oft ungewollt Textpassagen ersetzt werden.

Alles ersetzen

Import und Export von Programmcode

Manchmal ist es sinnvoll, eine Textdatei in das Modulfenster zu importieren. In der Regel handelt es sich hierbei um Programmcode, der im Textformat in einer Datei vorliegt. Zu diesem Zweck stellt der Menüpunkt *Einfügen > Datei* ein entsprechendes Dialogfenster zur Verfügung, über das der Pfad und der Name der Datei ausgewählt werden können. Der Text in der Datei sollte hierbei im ANSI-Format vorliegen.



Abb. 3.6

Das Dialogfenster *Datei einfügen*



Export Den Export von Programmcode führen Sie über den Menüpunkt *Datei > Datei exportieren* durch.

Hinweis



Der Programmcode des Deklarationsteils wird zuerst in die Textdatei exportiert. Danach folgen die Prozeduren in der gleichen Reihenfolge, wie sie in der vollen Modulansicht angeordnet sind.

Ausdruck eines Listings



Über den Menüpunkt *Datei > Drucken* oder das entsprechende Symbol in der Symbolleiste kann ein zusammenhängendes Listing des Programmcodes auf dem Drucker ausgegeben werden. Im Dialogfenster *Drucken* können Sie neben der Anzahl der Exemplare auch die auszudruckenden Seiten angeben. Ist das Kontrollkästchen *Druckausgabe in Datei umleiten* markiert, erfolgt die Textausgabe nicht auf den Drucker, sondern in eine Datei. In diesem Fall werden Sie zur Eingabe eines Dateinamens aufgefordert.

Die Einstellungen des aktuellen Druckers können Sie verändern, indem Sie die Schaltfläche *Einrichten* betätigen. In dem darauf erscheinenden Dialogfenster können Sie beispielsweise die Seitenränder anpassen.



Abb. 3.7

Das Dialogfenster *Drucken*



Module

Module stellen eine Art Behälter für Prozeduren dar. Access stellt zwei verschiedene Typen von Modulen zur Verfügung: *Standardmodule* und *Klassenmodule*. Klassenmodule haben Sie bereits im vorhergehenden Kapitel im Zusammenhang mit Ereignisprozeduren kennen gelernt. Jedes Formular und jeder Bericht hat genau ein Klassenmodul (auch als *Formular-* bzw. *Berichtsmodul* oder Objektmodul bezeichnet). Im Klassenmodul werden normalerweise alle Prozeduren untergebracht, die das Formular bzw. den Bericht direkt betreffen. Dazu gehören beispielsweise die Ereignisprozeduren.

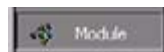
Sie können ein an ein Formular oder einen Bericht gebundenes Klassenmodul auch bearbeiten, ohne das Formular oder den Bericht zu öffnen. Markieren Sie dazu nur das entsprechende Objekt im Datenbankfenster und klicken Sie auf das Symbol *Code* in der Symbolleiste.

Mithilfe der Eigenschaft *Enthält Modul* – die sowohl von Formularen als auch von Berichten zur Verfügung gestellt wird – können Sie das Klassenmodul eines Formulars oder Berichts entfernen. Stellen Sie die Eigenschaft dazu einfach auf *Nein* ein. Diese Einstellung bietet sich besonders bei Formularen an, deren Modul keinen Programmcode enthält. Das Öffnen wird dadurch erheblich beschleunigt.

Neben Klassenmodulen, die an Formulare oder Berichte geknüpft sind, stellt Access auch eigenständige Klassenmodule zur Verfügung, die objektorientierte Programmierung ermöglichen. Mehr zu diesem Thema erfahren Sie im *Kapitel 9 Objektorientierte Programmierung*.

Standardmodule sind nicht an ein Formular oder einen Bericht geknüpft. Sie können einzeln vom Datenbankfenster aus angelegt und verwaltet werden. Hierzu stellt Access im Datenbankfenster eine Kategorie mit dem Namen *Module* zur Verfügung. Von hier aus können Sie Standardmodule auf die gleiche Art und Weise wie beispielsweise Makros handhaben. Mit der Schaltfläche *Neu* legen Sie ein neues leeres Modul an. Ein vorhandenes Modul öffnen Sie mit der Schaltfläche *Entwurf*. Der einzige Unterschied zu Makros besteht darin, dass Sie ein Modul nicht ausführen können. Der Grund dafür ist, dass Access nur einzelne Prozeduren, aber keine kompletten Module ausführen kann. Die *Ausführen*-Schaltfläche des Datenbankfensters ist daher stets deaktiviert.

Klassenmodule



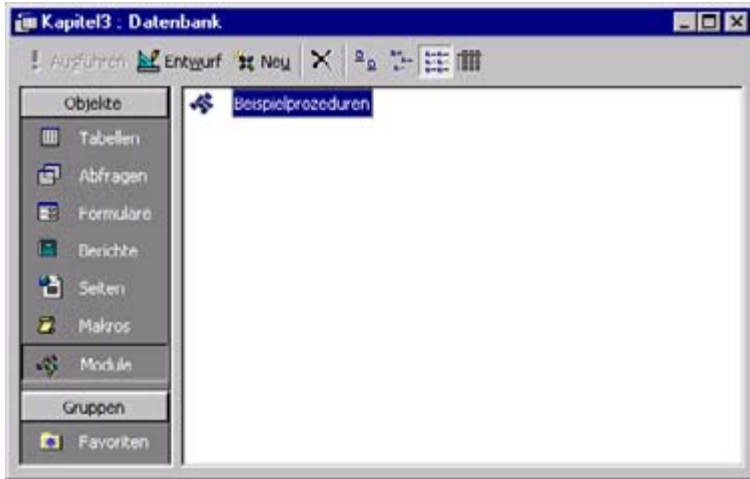
Hinweis

Standardmodule



Abb. 3.8

Die Kategorie *Module* des Datenbankfensters



Modulnamen

Jedes Standardmodul hat einen Namen. Wenn Sie beispielsweise in einem Modul mehrere Prozeduren zur Steuerung eines Datenaustausches erstellt haben, könnten Sie dem Modul den Namen *Datenaustausch* geben. Folgende Regeln müssen bei der Namensvergabe beachtet werden:

- Der Name darf maximal 64 Zeichen lang sein.
- Folgende Zeichen dürfen nicht im Modulnamen auftauchen: . ! ' [].
- Ein Modulname darf nicht mit einem oder mehreren Leerzeichen beginnen.
- Der Name sollte nicht mit den Zeichenfolgen *Form_* oder *Report_* beginnen.
- Der Modulname muss für die aktuelle Datenbank eindeutig sein.

Formular- und Berichtsmodule

Formular- und Berichtsmodule werden von Access automatisch zusammen mit den betreffenden Formularen und Berichten verwaltet. Sie erscheinen daher nicht im Datenbankfenster. Auch die Namensvergabe wird von Access geregelt. Der Name setzt sich grundsätzlich aus dem Text *Form_* und dem Namen des Formulars zusammen. Bei Berichten beginnt der Name mit dem Text *Report_*.



Wenn Sie das Klassenmodul eines Formulars oder Berichts öffnen möchten, öffnen Sie zunächst das betreffende Formular oder den betreffenden Bericht in der Entwurfsansicht. In der Symbolleiste finden Sie ein Symbol mit dem Namen *Code*, über das sich das Klassenmodul öffnen lässt. Alternativ steht Ihnen auch der Menüpunkt *Ansicht > Code* zur Verfügung.

Aufbau eines Moduls

Ein Modul besteht aus dem *Deklarations-* und dem *Prozedurbereich*. Im Deklarationsbereich werden allgemeine Einstellungen vorgenommen, die die Prozeduren des Moduls beeinflussen. Die Prozeduren werden schließlich im Prozedurbereich untergebracht.



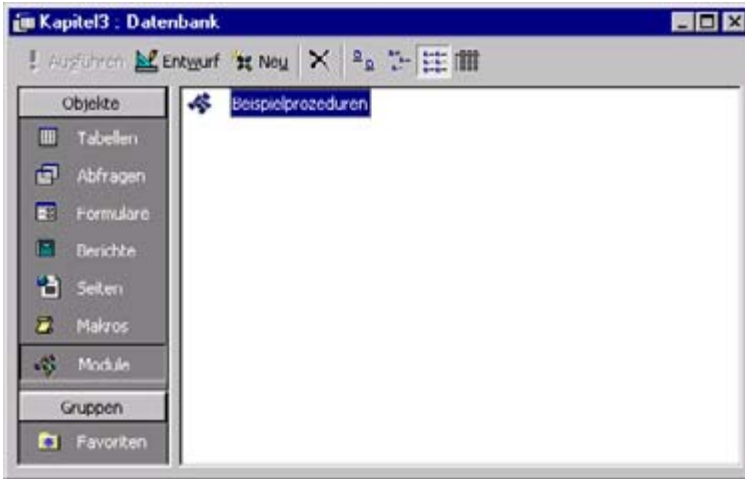


Abb. 3.9

Der Aufbau eines Moduls

Prozeduren

Der Herzstück eines VBA-Programms sind die Prozeduren. Jede Prozedur hat eine bestimmte Aufgabe zu erledigen. Eine komplette Datenbank besteht in der Regel aus einer Vielzahl solcher Prozeduren, die erst im Zusammenspiel mit den Datenbankobjekten von Access eine komplette Datenbankanwendung ergeben.

Damit die einzelnen Prozeduren auseinander gehalten werden können, hat jede Prozedur einen Namen, der den folgenden Regeln entsprechen muss:

- Der Name muss mit einem Buchstaben beginnen.
- Er darf nur Buchstaben, Zahlen und das Unterstrichungszeichen enthalten. Leer-, Satz- oder Sonderzeichen sind nicht erlaubt.
- Die Länge darf 255 Zeichen nicht überschreiten.
- Der Name darf nicht mit einem reservierten Access-Schlüsselwort übereinstimmen. Dazu gehören beispielsweise die Namen von Anweisungen oder Befehlen.

Namens-
konventionen von
Prozeduren

Außerdem müssen Sie darauf achten, dass der Name einer Prozedur innerhalb eines Moduls eindeutig ist. Es ist jedoch zulässig, dass Sie in zwei verschiedenen Modulen eine Prozedur mit demselben Namen anlegen.



Prozedurtypen

Access unterscheidet zwischen *Sub*- und *Function*-Prozeduren. *Sub*-Prozeduren haben Sie bereits im *Kapitel 2 Schnellstart VBA* kennen gelernt. Alle Ereignisprozeduren, die Sie über den Code-Editor anlegen, sind *Sub*-Prozeduren. Mithilfe von *Sub*-Prozeduren können Sie Prozeduren anlegen, die Sie von anderen Prozeduren aus wie die in Access eingebauten Anweisungen aufrufen können. Sie können so den Sprachumfang von Access auf einfache Weise erweitern.

Function-Prozeduren haben die Aufgabe, die in Access eingebauten Funktionen durch selbstdefinierte Funktionen zu ergänzen. Sie könnten beispielsweise eine *Function*-Prozedur anlegen, die das Alter einer Person anhand des Geburtsdatums berechnet. Diese Funktion könnten Sie dann innerhalb von Abfragen oder auch in VBA einsetzen. Da zum Verständnis von *Function*-Prozeduren Kenntnisse über Variablen notwendig sind, werden im Folgenden lediglich *Sub*-Prozeduren erläutert. Näheres zu *Function*-Prozeduren erfahren Sie im Abschnitt *Parametervariablen*.

Erstellen einer Prozedur

Eine Möglichkeit zum Erstellen einer Prozedur haben Sie bereits im vorhergehenden Kapitel in Verbindung mit dem Code-Editor kennen gelernt. Der Code-Editor ist allerdings nur zum Anlegen von Ereignisprozeduren geeignet. Wie bereits erwähnt, können Module auch Prozeduren enthalten, die nicht direkt mit einem Ereignis verknüpft sind. Solche Prozeduren werden in der Regel von anderen Prozeduren – beispielsweise Ereignisprozeduren – aufgerufen.



Zum Erstellen einer neuen Prozedur innerhalb eines Moduls steht der Menüpunkt *Einfügen > Prozedur* bzw. das Symbol *Prozedur hinzufügen* zur Verfügung. Nach Auswahl erscheint das Dialogfenster *Prozedur hinzufügen*.

Abb. 3.10

Das Dialogfenster *Prozedur hinzufügen*



Im Textfeld *Name* können Sie den Namen der neuen Prozedur angeben. Den Typ der Prozedur legen Sie im Bereich *Typ* fest. Access stellt hier die Optionen *Sub* und *Function* zur Verfügung.

Neben den Typen *Sub* und *Function* stellt Access noch eine dritte Option mit dem Namen *Property* zur Verfügung. Über diese Option können Sie dem Modul zwei *Property*-Prozeduren hinzufügen, die im Prinzip nichts anderes als *Sub*- und *Function*-Prozeduren sind. Sie unterscheiden sich jedoch in der Art und Weise, wie sie von einer anderen Prozedur aus aufgerufen werden. Mehr zu *Property*-Prozeduren erfahren Sie im *Kapitel 9 Objektorientierte Programmierung*.



Hinweis

Nachdem Sie das Dialogfenster mit der *OK*-Schaltfläche bestätigt haben, erscheint die noch leere Prozedur im Modulfenster. Von diesem Zeitpunkt an steht die Prozedur auch über die Prozedurliste zur Verfügung.

Um das Anlegen einer Prozedur einmal auszuprobieren, geben Sie im Feld *Name* den Prozedurnamen *Testprozedur* ein. Als *Typ* wählen Sie die Option *Sub* aus. Nachdem Sie das Dialogfenster mit *OK* bestätigt haben, erscheint die folgende Prozedur:

```
Public Sub Testprozedur()
```

```
End Sub
```

Eine andere Möglichkeit, eine neue Prozedur zu erstellen, besteht darin, den Kopf der Prozedur mit der Anweisung *Sub* direkt in das Modulfenster einzugeben:

```
Public Sub Testprozedur()
```



Direkte Eingabe des Prozedurkopfs

Bei der Eingabe müssen Sie darauf achten, dass Sie diese nicht inmitten einer Prozedur machen. Gut geeignet ist der Deklarationsbereich oder ein freier Bereich zwischen zwei Prozeduren. Wenn Sie nach der Eingabe des Prozedurkopfs die -Taste drücken, ergänzt Access ihn automatisch um die Anweisung *End Sub* und zeigt die neue Prozedur im Modulfenster an.

Aufbau von *Sub*-Prozeduren

Den grundlegenden Aufbau einer *Sub*-Prozedur haben Sie bereits im vorhergehenden Kapitel kennen gelernt. Die einfachste Form einer *Sub*-Prozedur muss der folgenden Syntax folgen:

```
Public Sub <Prozedurname>()
```

```
<Anweisungen>
```

```
End Sub
```





Im Folgenden wird die Verwendung von Prozeduren anhand einer Beispielprozedur mit dem Namen *MeldungAnzeigen* erläutert:

Die Prozedur
MeldungAnzeigen

```
Public Sub MeldungAnzeigen()  
    Beep  
    MsgBox "Dies ist eine Meldung"  
  
End Sub
```

Zur Eingabe der Prozedur legen Sie zuerst über das Datenbankfenster ein neues Modul an. Über den Menüpunkt *Einfügen > Prozedur* öffnen Sie das Dialogfenster *Prozedur einfügen* und geben den Prozedurnamen *MeldungAnzeigen* ein. Zusätzlich markieren Sie die Option *Sub*. Nachdem Sie mit *OK* bestätigt haben, erscheint im Modulfenster der leere Prozedurrumpf:

```
Public Sub MeldungAnzeigen()  
  
End Sub
```

Diesen füllen Sie anschließend mit den beiden Anweisungen zur Ausgabe eines Signals und zur Anzeige der Meldung.

Ausführen der
Prozedur

Im Gegensatz zu den Beispielprozeduren, die Sie im zweiten Kapitel kennen gelernt haben, handelt es sich bei dieser Prozedur nicht um eine Ereignisprozedur. Genauer gesagt besteht von der Prozedur aus keine Verknüpfung zu einem Ereignis, das zum Start der Prozedur führen könnte. Die einzige Möglichkeit, eine solche "normale" Prozedur wie die *MeldungAnzeigen*-Prozedur zu starten, besteht darin, sie von einer anderen Prozedur aus aufzurufen. Die aufrufende Prozedur kann nun wiederum eine "normale" Prozedur oder aber eine Ereignisprozedur sein.

Hinweis



Das Ende einer *Sub*-Prozedur wird durch die Anweisung *End Sub* markiert. Soll die Prozedur vorzeitig beendet werden, geben Sie innerhalb der Prozedur einfach folgende Anweisung an:

```
Exit Sub
```

Aufruf einer *Sub*-Prozedur

Um eine *Sub*-Prozedur von einer anderen Prozedur aus aufzurufen, stellt Access die *Call*-Anweisung zur Verfügung:

```
Call <Prozedurname>
```

Wie Sie der Befehlssyntax entnehmen können, muss hinter dem Schlüsselwort *Call* lediglich der Name der Prozedur angegeben werden, die Sie aufrufen möchten. Die folgende Anweisung ruft beispielsweise die eben erstellte Beispielprozedur auf:

```
Call MeldungAnzeigen
```



Das Schlüsselwort *Call* kann allerdings ausgelassen werden, so dass sich folgende Kurzform zum Start der Prozedur ergibt:

MeldungAnzeigen

Prozedurstart über das Testfenster

Bei Prozeduren bietet es sich an, diese direkt nach der Fertigstellung auf Fehler zu untersuchen. So können Sie sicherstellen, dass die Prozedur die gestellte Aufgabe auch wirklich nach Ihren Anforderungen erledigt.

Zum Start der Prozedur positionieren Sie einfach die Einfügemarke auf einer beliebigen Anweisung der gewünschten Prozedur oder dem Kopf der Prozedur. Anschließend klicken Sie auf das Symbol *Ausführen* in der Symbolleiste.



Alternativ starten Sie die Prozedur vom Testfenster aus, das Sie über den Menüpunkt *Ansicht > Direktfenster* oder das Symbol *Direktfenster* in der Symbolleiste öffnen.



Abb. 3.11

Das Direktfenster

Das Direktfenster dient unter anderem dazu, eine Anweisung, die Sie hier eingeben, direkt auszuführen. Um die Beispielprozedur *MeldungAnzeigen* zu starten, geben Sie einfach in die erste Zeile des unteren Bereichs die Anweisung

```
Call MeldungAnzeigen
```

ein. Wenn Sie nun am Ende der Zeile die -Taste drücken, führt Access die Anweisung aus und startet damit die Prozedur.

Wie bereits erwähnt, kann das Schlüsselwort *Call* auch ausgelassen und lediglich der Name der aufzurufenden Prozedur angegeben werden:

```
MeldungAnzeigen
```

Anstatt die *Call*-Anweisung im Direktfenster einzusetzen, können Sie diese auch in Prozeduren verwenden. Die folgende Prozedur startet beispielsweise zweimal nacheinander die Prozedur *MeldungAnzeigen*:





```
Public Sub Testprozedur1()
```

```
    MeldungAnzeigen  
    MeldungAnzeigen
```

```
End Sub
```

Wenn Sie diese zweite Prozedur ebenfalls ausprobieren möchten, geben Sie sie auf die gleiche Weise im Modulfenster ein wie vorher die *MeldungAnzeigen*-Prozedur. Den Start der Prozedur nehmen Sie wieder über das Direktfenster vor, indem Sie hier in eine freie Zeile eine der beiden folgenden Anweisungen eingeben:

```
Call Testprozedur  
Testprozedur
```

Natürlich können Sie die Prozedur auch über das *Ausführen*-Symbol starten. Als Ergebnis erscheint das Meldungsfenster zweimal hintereinander.

Hinweis



Wenn Sie bei der Eingabe der Prozedur oder beim Start einen Fehler machen, zeigt Access eine entsprechende Meldung an. Einige Fehler werden von Access erst dann erkannt, wenn Sie die Prozedur ausführen. Kurz vor der Ausführung der Prozedur nimmt Access die so genannte *Kompilierung* vor, bei der der Programmtext auf Fehler untersucht wird. Diese Fehler werden auch als *Kompilierfehler* bezeichnet. Mehr zur Kompilierung erfahren Sie im Abschnitt *Kompilieren des Programmcodes* später in diesem Kapitel.

Strukturierte Programmierung

Eine einzelne Prozedur kann eine nahezu unbegrenzte Anzahl an Anweisungen beinhalten. Wenn Sie also eine komplizierte Aufgabe lösen möchten, können Sie eine entsprechend lange Prozedur schreiben.

Wenn Sie noch einmal das letzte Beispiel des vorhergehenden Abschnitts betrachten, hätten Sie die zwei Meldungsfenster auch mithilfe einer einzigen Prozedur anzeigen lassen können:

```
Public Sub Testprozedur2()
```

```
    Beep  
    MsgBox "Dies ist eine Meldung"  
    Beep  
    MsgBox "Dies ist eine Meldung"
```

```
End Sub
```

In dieser Prozedur werden die Anweisungen jeweils doppelt angegeben. Dies ist ein typischer Fall, bei dem Sie eine weitere Prozedur anlegen sollten.





Aber auch dann, wenn sich Anweisungen nicht wiederholen, ist die Unterteilung in mehrere Prozeduren sinnvoll. Sie erhalten so mehrere kleine Programmstücke, die Sie getrennt voneinander programmieren und testen können. Die Hauptprozedur, von der aus Sie dann die einzelnen Prozeduren (die in diesem Fall auch als *Unterprozeduren* bezeichnet werden) aufrufen, wird zudem wesentlich übersichtlicher. Dieses Prinzip wird auch als *strukturierte Programmierung* bezeichnet.



**Strukturierte
Programmierung**

Besonders wichtig ist die Aufteilung eines Problems in mehrere Prozeduren, wenn Sie den Einsatz der einzelnen Prozeduren an mehreren Stellen des Programms planen. So könnte beispielsweise die Prozedur *MeldungAnzeigen* von verschiedenen Formularen einer Datenbank aus aufgerufen werden. Wenn Sie nun den Text der Meldung ändern möchten, müssen Sie nicht jede einzelne Programmstelle, sondern lediglich die Prozedur *MeldungAnzeigen* abändern. Ein vernünftig strukturierter Programmcode lässt sich also wesentlich leichter warten.

Wenn Sie ein Problem allerdings in zu viele Teilprobleme zerlegen, kann sich dieser Vorteil wieder in einen Nachteil umkehren. In der Praxis sollte eine einzelne Prozedur nicht länger als 100 Programmzeilen werden. Ob Sie ein Problem lieber in mehrere kleine Teilprobleme aufteilen oder aber mit längeren Prozeduren arbeiten möchten, hängt letztendlich jedoch ausschließlich von Ihnen und Ihrem Programmierstil ab.

Geltungsbereiche von Prozeduren

Alle BeispielprozEDUREN der vorhergehenden Abschnitte waren im Prozedurkopf mit dem Schlüsselwort *Public* versehen. Prozeduren mit diesem Attribut werden auch als *öffentliche Prozeduren* bezeichnet. Das Gegenstück dazu sind Prozeduren mit dem Attribut *Private*, die als private Prozeduren bezeichnet werden.



Public und Private

Access betrachtet die einzelnen Standardmodule einer Datenbank als Softwarekomponenten oder Objekte. Ein wichtiges Merkmal solcher Objekte ist die Unterteilung in Elemente, auf die von außen nicht zugegriffen werden kann und Elemente, die einen Zugriff von außen erlauben. Letztere können auch als Schnittstelle des Objekts zur Außenwelt betrachtet werden.

Innerhalb von Access wird diese Schnittstelle durch öffentliche Prozeduren definiert, die von beliebigen Prozeduren innerhalb der Datenbank ausgeführt werden können. Der Geltungsbereich öffentlicher Prozeduren erstreckt sich also über die komplette Datenbank. Private Prozeduren können dagegen nur von Prozeduren aufgerufen werden, die sich im gleichen Modul befinden. Der Geltungsbereich ist also auf das Modul beschränkt.

In der Praxis hat ein Standardmodul immer mindestens eine öffentliche Prozedur, die die Schnittstelle zur Außenwelt bildet. Von dieser Prozedur können nun entweder andere öffentliche Prozeduren oder private Prozeduren des aktuellen Moduls aufgerufen werden.



Hinweis



Wenn Sie bei der Definition einer Prozedur weder *Public* noch *Private* angeben, wird die Prozedur automatisch als *Public* deklariert.

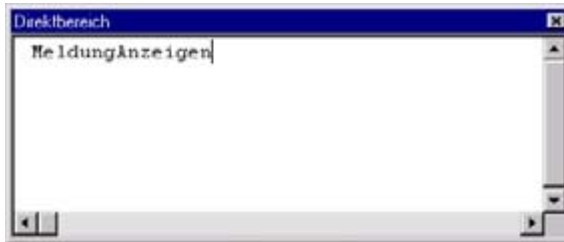
Namenskonflikte

Ein weiteres Prinzip von Objekten ist die Tatsache, dass diese sich gegenseitig nicht stören dürfen. Konflikte können aber bereits dann auftreten, wenn sich in zwei verschiedenen Modulen A und B zwei Prozeduren mit dem gleichen Namen *Testprozedur* befinden.

Wenn beide Prozeduren privat sind, gibt es keine Konflikte. Wird die Prozedur von dem Modul A aus aufgerufen, startet Access automatisch die Prozedur im Modul A. Gleiches gilt für das Modul B.

Abb. 3.12

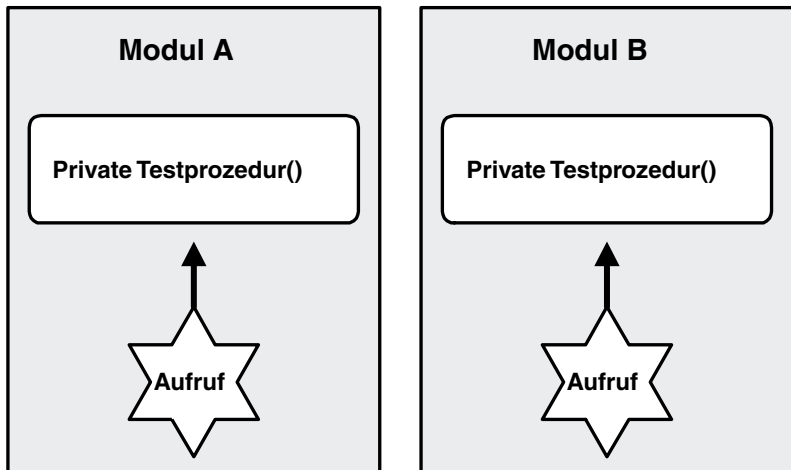
Aufruf zweier
gleichnamiger privater
Prozeduren



Auch wenn nur die Prozedur im Modul A privat, die andere im Modul B aber öffentlich ist, gibt es keine Probleme. Wird die Prozedur vom Modul A aus aufgerufen, startet Access die Prozedur im Modul A. Wird sie dagegen von einem anderen Modul (z. B. B oder C) aus aufgerufen, startet Access die Prozedur im Modul B.

Abb. 3.13

Aufruf einer privaten
und einer öffentlichen
Prozedur mit dem gleichen Namen



Probleme treten allerdings auf, wenn beide Testprozeduren in den Modulen A und B als öffentlich deklariert sind. Hierbei gelten folgende Regeln:

- Wird die Prozedur vom Modul A aus aufgerufen, startet Access die Prozedur im Modul A.
- Wird die Prozedur vom Modul B aus aufgerufen, startet Access die Prozedur im Modul B.
- Wird die Prozedur vom Modul C aus aufgerufen, kann Access den Aufruf nicht eindeutig zuordnen. Es ist hier nicht klar, ob die Prozedur im Modul A oder im Modul B gestartet werden soll. In diesem Fall meldet Access während der Kompilierung des Programmcodes den Fehler "Name nicht eindeutig".



○ Abb. 3.14

Aufruf zweier öffentlicher Prozeduren mit dem gleichen Namen

Wenn ein solcher Namenskonflikt auftritt, müssen Sie beim Aufruf der Prozedur den Namen des Moduls in folgender Form mit angeben:

[<Modulname>] .<Prozedurname>

Die Angabe der eckigen Klammern ist nur dann notwendig, wenn der Modulname Leer- oder Sonderzeichen enthält.

Um beispielsweise die Testprozedur des Moduls mit dem Namen *Modul A* aufzurufen, geben Sie folgende Anweisung an:

[Modul A] .Testprozedur

Nun ist eindeutig bestimmt, welche Prozedur Sie ausführen möchten, und Access meldet keinen Fehler mehr.

Bei der Entwicklung von Datenbanken sollten Sie Namenskonflikte von vornherein vermeiden, indem Sie öffentlichen Prozeduren Namen geben, die in der kompletten Datenbank eindeutig sind. Außerdem sollten Sie Prozeduren, die nur von demjenigen Modul aufgerufen werden, in dem sie implementiert sind, grundsätzlich als privat deklarieren.



Hinweis





Geltungsbereich von Prozeduren in Formular- und Berichtsmodulen

Öffentliche Prozeduren in Formular- oder Berichtsmodulen haben einen eingeschränkten Geltungsbereich. Wenn solche Prozeduren von anderen Prozeduren außerhalb des Klassenmoduls aus aufgerufen werden, muss stets der Name des Moduls mit angegeben werden. Bei Formularen bildet Access den Namen des Klassenmoduls aus dem Text *Form_*, dem der Name des Formulars folgt. Um also beispielsweise die öffentliche Prozedur des Formulars *Adressen* aufzurufen, geben Sie folgende Anweisung an:

```
Form_Adressen.Testprozedur
```

Auch hier gilt wieder, dass Sie den Namen des Moduls in eckigen Klammern angeben müssen, wenn er Leer- oder Sonderzeichen enthält.

Bei Berichten beginnt der Klassenmodulname mit dem Text *Report_*. Das folgende Beispiel führt die öffentliche Prozedur *Testprozedur* aus, die sich im Klassenmodul des Berichts *Adressenliste* befindet:

```
Report_Adressenliste.Testprozedur
```

Eine weitere Möglichkeit zum Aufruf einer öffentlichen Prozedur in einem Klassenmodul besteht darin, dass Sie anstelle des Modulnamens einen Verweis auf das Formular oder den Bericht angeben. Das folgende Beispiel ruft die Testprozedur des Formulars *Adressen* auf:

```
Forms!Adressen.Testprozedur
```

Hierbei müssen Sie allerdings selbst darauf achten, dass das betreffende Formular in der Formular- oder Datenblattansicht geöffnet ist. Sollte dies nicht der Fall sein, tritt ein Laufzeitfehler auf. Gleiches gilt für Berichte, die in der Seitenansicht geöffnet sein müssen.

Genaue Informationen dazu, wie Sie auf Formulare und Berichte verweisen, erhalten Sie im folgenden *Kapitel 4 Programmieren mit Objekten*.

Ausdrücke

Bis jetzt haben Sie zur Programmierung einer Prozedur nur Anweisungen kennen gelernt, die im Prinzip mit den Aktionen von Makros vergleichbar sind. Darüber hinaus stellt VBA auch die so genannten *Ausdrücke* zur Verfügung, die Sie bereits im Zusammenhang mit der Spalte *Bedingungen* bei der Besprechung der Makros kennen gelernt haben.

Im Gegensatz zu Anweisungen führen Ausdrücke keine Aktionen durch, sondern repräsentieren immer einen Wert. Neben numerischen Ausdrücken, die einen Zahlenwert repräsentieren, kennt Access noch einige weitere Typen von Ausdrücken. Dazu zählen beispielsweise Texte, die auch als Zeichenkettenausdrücke bezeichnet werden. In den folgenden Abschnitten werden Ihnen diese unterschiedlichen Typen vorgestellt.



Nummerische Ausdrücke

Ein numerischer Ausdruck kann entweder ein Wert selbst sein – wie beispielsweise eine Zahl – oder eine Berechnung, die zu einem bestimmten Ergebnis führt. Um beispielsweise zwei Zahlen zu addieren, können Sie den folgenden Ausdruck angeben:

`3 + 4`

Bei diesem Beispiel handelt es sich eigentlich um zwei Ausdrücke, die zu einem einzigen Ausdruck verknüpft werden. Die Verknüpfung wird hierbei durch das `+`-Zeichen vorgenommen, das auch als *Operator* bezeichnet wird.

Um etwas mit numerischen Ausdrücken zu experimentieren, eignet sich wieder das Direktfenster, das Sie über den Menüpunkt *Ansicht > Direktfenster* oder das gleichnamige Symbol in der Symbolleiste öffnen. Beachten Sie, dass dieser Menüpunkt nur dann zur Verfügung steht, wenn ein Modulfenster geöffnet und aktiviert ist. Gegebenenfalls müssen Sie also vorher ein vorhandenes Modul öffnen oder aber ein neues leeres Modul anlegen. Danach können Sie das Modul wieder schließen; das Direktfenster bleibt dennoch geöffnet.

Öffnen des Direktfensters



Wenn Sie in das Direktfenster nun direkt den Ausdruck `3 + 4` eingeben und am Ende der Zeile die -Taste drücken, quittiert Access dies mit einer Fehlermeldung. Der Grund dafür ist, dass im Direktfenster Anweisungen angegeben werden müssen. Um nun trotzdem das Ergebnis eines Ausdrucks betrachten zu können, stellt Access die *Debug.Print*-Anweisung zur Verfügung. Die Aufgabe dieser Anweisung ist, einen beliebigen Ausdruck im Direktfenster auszugeben. Wenn Sie in das Direktfenster den folgenden Text eingeben, zeigt Access in der nachfolgenden Zeile das Ergebnis 7 an:

`Debug.Print 3 + 4`

Wenn Sie die *Debug.Print*-Anweisung im Direktfenster einsetzen, können Sie auch eine Kurzform verwenden. Geben Sie hierzu einfach nur *Print* an:

`Print 3 + 4`

Wenn Sie zu den ganz Schreibfaulen gehören, können Sie auch die Abkürzung der Kurzform verwenden. Ersetzen Sie die *Print*-Anweisung durch ein simples Fragezeichen. Bei dieser Form können Sie auch auf das Leerzeichen nach der Anweisung verzichten:

`? 3 + 4`



Abb. 3.15

Die verschiedenen
Formen des *Print*-
Befehls im
Direktfenster



Hinweis



Dezimalzahlen

Wenn Sie negative Zahlen angeben möchten, stellen Sie der Zahl einfach ein Minuszeichen voran.

Access kann allerdings nicht nur mit ganzen Zahlen, sondern auch mit Dezimalbrüchen rechnen:

```
Print 4.6 + 6.3
```

Bei der Angabe von Dezimalbrüchen müssen Sie die amerikanische Schreibweise verwenden, bei der der Punkt als Dezimalbruchtrennzeichen dient. Das Ergebnis der Berechnung wird dann von Access wieder in der vertrauten Schreibweise mit einem Komma angezeigt. Wie bei Ihrem Taschenrechner dürfen Sie innerhalb von VBA Zahlenwerte zwischen -1 und 1 auch ohne führende 0 vor dem Komma angeben:

```
Print .6 + .4
```

Exponentialformat

Als weitere Möglichkeit können Sie eine Zahl auch im Exponentialformat angeben. Hierbei muss die Basis vom Exponenten durch den kleinen oder großen Buchstaben E getrennt werden. Um die Zahl 1000 im Exponentialformat anzugeben, können Sie beispielsweise den Ausdruck *1E3* verwenden. Access selbst verwendet die Exponentialschreibweise erst bei großen Zahlen. Wenn Sie also die Anweisung

```
Print 1E3
```

im Direktfenster angeben, gibt Access als Ergebnis *1000* aus. Dagegen bevorzugt Access bei dem folgenden Ausdruck die Exponentialschreibweise und gibt *1E+15* als Ergebnis aus:

```
Print 1000000000000000
```

Hexadezimal- und
Oktalzahlen

Zahlen können nicht nur im Dezimalsystem, sondern auch im Hexadezimalsystem und im Oktalsystem angegeben werden. In der Programmierpraxis spielen diese Zahlensysteme allerdings keine große Rolle. Damit Access eine normale Zahl im Dezimalsystem von Zahlen in anderen Zahlensystemen unterscheiden kann, müs-



sen Sie einer Hexadezimalzahl die Zeichenfolge `&H` und einer Oktalzahl die Zeichenfolge `&O` voranstellen. Gültige Hexadezimalzahlen sind beispielsweise `&HAFFF` oder `&H100`.

Nummerische Operatoren

Zur Verknüpfung zweier numerischer Ausdrücke stehen eine Reihe arithmetischer Operatoren zur Verfügung, über die sich die elementaren Rechenarten der Mathematik ausführen lassen. In den vorhergehenden Beispielen haben Sie bereits den `+`-Operator kennen gelernt. Die folgende Tabelle enthält alle Operatoren, die Access unterstützt:

Operator	Priorität	Bedeutung
<code>^</code>	1	Potenzierung
<code>*</code>	2	Multiplikation
<code>/</code>	2	Division
<code>\</code>	3	Ganzzahlige Division
<code>Mod</code>	4	Modulo-Berechnung
<code>+</code>	5	Addition
<code>-</code>	5	Subtraktion

Tabelle 3.4

Der Operator `\` führt eine Division ohne Rest durch. Der Ausdruck `7 \ 3` hätte daher den Wert 2 zum Ergebnis. Dagegen führt der `Mod`-Operator eine Division durch und gibt lediglich den Rest aus. Das Ergebnis von `7 Mod 3` ist also der Wert 1. Die folgende Tabelle enthält einige Beispiele zu den einzelnen Operatoren. Wenn Sie die Beispiele im Direktfenster ausprobieren möchten, vergessen Sie nicht, vor den Ausdrücken den `Print`-Befehl anzugeben:



Hinweis

Ausdruck	Ergebnis
<code>10 ^ 2</code>	100
<code>3 + 4 * 2</code>	11
<code>(3 + 4) * 2</code>	14



Beispiele zur Verwendung von Operatoren (Forts.)

Ausdruck	Ergebnis
$5 / 2$	2.5
$2.5 * 3$	7.5
$2 * 2.5E-5$	0.00005
$20 \text{ Mod } 6$	2
$20 \setminus 6$	3

*Tabelle 3.5***Prioritäten der Operatoren**

Wie bei einem Taschenrechner kommt es bei Ausdrücken mit mehreren Verknüpfungsoperatoren – beispielsweise $10 * 5 / 7 + 3$ – auf die Prioritäten an: So berücksichtigt Access beispielsweise die Regel "Punkt- vor Strichrechnung" bei der Auswertung eines Ausdrucks. Die Operatoren mit der höchsten Priorität (Priorität 1) werden zuerst, die mit der niedrigsten Priorität (Priorität 5) zuletzt berücksichtigt. Bei Operatoren mit gleicher Priorität wertet Access den Ausdruck von links nach rechts aus. Die Prioritäten der einzelnen Operatoren entnehmen Sie der vorletzten Tabelle.

Wie in der Mathematik lässt sich die Priorität eines Operators verändern, indem der jeweilige Teilausdruck eingeklammert wird. Das folgende Beispiel zeigt dies:

```
5 * 3 + 4 "Ergebnis 19
5 * (3 + 4) "Ergebnis 35
```

In einem Ausdruck dürfen hierbei beliebig viele Klammernpaare verwendet werden.

Mathematische Funktionen

Neben den Grundrechenarten wie Addition oder Multiplikation stellt Access eine Reihe von mathematischen Funktionen zur Verfügung. Ein Beispiel hierfür ist die *Sin*-Funktion zur Berechnung des Sinuswertes eines Winkels:

```
Sin(<Winkel>)
```

Für <Winkel> können Sie einen beliebigen numerischen Ausdruck angeben, der von Access als Winkel interpretiert wird. Die Angabe des Winkels muss hierbei im Bogenmaß erfolgen. Der Ausdruck

```
Sin(1.2)
```

führt beispielsweise ungefähr zu dem Ergebnis 0,932. Wenn Sie das Beispiel im Direktfenster ausprobieren möchten, geben Sie den Ausdruck mit vorangestelltem *Print*-Befehl an:

```
Print Sin(1.2)
```



Insgesamt stellt Access die folgenden trigonometrischen Funktionen zur Verfügung:

Trigonometrische Funktionen

Funktion	Beschreibung
<i>Sin</i> (<i><Winkel></i>)	Sinus
<i>Cos</i> (<i><Winkel></i>)	Kosinus
<i>Tan</i> (<i><Winkel></i>)	Tangens
<i>Atn</i> (<i><Zahl></i>)	Arkustangens

Tabelle 3.6

Die trigonometrischen Funktionen können natürlich untereinander oder auch mit anderen Funktionen beliebig kombiniert werden, so dass Sie komplexe Ausdrücke bilden können:

$2 * \text{Tan}(\text{Sin}(180 * 3.14 * 45))$

Neben den trigonometrischen Funktionen unterstützt Access noch weitere mathematische Funktionen:

Mathematische Funktionen

Funktion	Beschreibung
<i>Abs</i> (<i><Zahl></i>)	Berechnet den Absolutwert einer Zahl. Das Ergebnis ist immer der positive Wert von <i><Zahl></i> .
<i>Exp</i> (<i><Zahl></i>)	Potenziert den Wert der eulerschen Zahl <i>e</i> mit <i><Zahl></i> .
<i>Fix</i> (<i><Zahl></i>)	Schneidet die Nachkommastellen der <i><Zahl></i> ab.
<i>Int</i> (<i><Zahl></i>)	Führt zum gleichen Ergebnis wie die <i>Fix</i> -Funktion. Bei negativen Zahlen wird <i><Zahl></i> allerdings auf die nächstkleinere Ganzzahl abgerundet.
<i>Log</i> (<i><Zahl></i>)	Berechnet den natürlichen Logarithmus von <i><Zahl></i> .
<i>Sqr</i> (<i><Zahl></i>)	Berechnet die Quadratwurzel von <i><Zahl></i> .



Mathematische Funktionen (Forts.)

Funktion	Beschreibung
<i>Sgn</i> (<Zahl>)	Ermittelt das Vorzeichen einer Zahl. Wenn <Zahl> positiv ist, gibt die Funktion den Wert 1 zurück. Ist <Zahl> dagegen negativ, ist das Ergebnis der Funktion der Wert -1. Sollte <Zahl> den Wert 0 besitzen, gibt die Funktion ebenfalls 0 zurück.
<i>Rnd</i>	Gibt eine Zufallszahl aus.

Tabelle 3.7

Die folgende Tabelle enthält einige Beispiele zu den Funktionen:

Beispiele zu den mathematischen Funktionen

Ausdruck	Ergebnis	Ausdruck	Ergebnis
<i>Abs</i> (5)	5	<i>Fix</i> (3.12)	3
<i>Abs</i> (-5)	5	<i>Fix</i> (-3.12)	-3
<i>Exp</i> (3)	20,08554	<i>Int</i> (3.12)	3
<i>Log</i> (3)	1,098612	<i>Int</i> (-3.12)	-4
<i>Sqr</i> (9)	3	<i>Sgn</i> (5)	1
<i>Sqr</i> (2.5)	1,581138...	<i>Sgn</i> (-4)	-1

Tabelle 3.8

Zufallszahlen

Die *Rnd*-Funktion hat eine besondere Bedeutung: Sie ermittelt auf der Grundlage eines mathematischen Algorithmus eine zufällig erscheinende Zahl, die zwischen 0 und 1 liegt. Wenn Sie die Anweisung

```
Print Rnd
```

im Testfenster angeben, könnte Access beispielsweise die Zahl 0,7055475 ausgeben. Da zur Bestimmung dieser Zufallszahlen ein mathematisches Verfahren verwendet wird, muss dieses beim Start mit einem Zufallswert initialisiert werden. Dafür ist eine Anweisung mit dem Namen *Randomize* zuständig:

```
Randomize [<Zufallszahl>]
```

Wenn Sie in Ihrem Programm Zufallszahlen verwenden möchten, sollten Sie die *Randomize*-Anweisung auf jeden Fall beim Start des Programms ausführen. Wenn Sie dies nicht erledigen, erhalten Sie bei jedem Start des Programms die gleichen "Zufallszahlen".

Ein Problem ist, dass diese Anweisung zur Initialisierung der *Rnd*-Berechnungsvorschrift ebenfalls eine Zufallszahl benötigt. Zu diesem Zweck kann der Anweisung beispielsweise die aktuelle Uhrzeit übergeben werden. Besser eignet sich allerdings das Ergebnis der *Timer*-Funktion. Diese Funktion gibt die Anzahl der seit Mitternacht verstrichenen Sekunden zurück. Sie können dies im Testfenster durch Eingabe der Anweisung *Print Timer* überprüfen. Um also sicherzustellen, immer andere Zufallszahlen zu erhalten, sollten Sie zuerst die *Randomize*-Anweisung ausführen:

```
Randomize Timer
```

Die Initialisierung ist damit erledigt, und Sie können anschließend mithilfe der *Rnd*-Funktion eine oder auch mehrere Zufallszahlen abfragen.

Wie Sie der Befehlssyntax der *Randomize*-Funktion entnehmen können, ist die Angabe des Parameters *<Zufallszahl>* optional und damit nicht unbedingt erforderlich. Sie erkennen dies daran, dass der Parameter von eckigen Klammern umgeben ist. Wenn Sie keine Zufallszahl angeben, verwendet die *Randomize*-Funktion automatisch den gleichen Wert, den auch die *Timer*-Funktion zurückgibt. Die beiden folgenden Anweisungen sind also gleichwertig:



Hinweis

```
Randomize Timer  
Randomize
```

Wie bereits erwähnt, gibt die *Rnd*-Funktion eine Zahl zwischen 0 und 1 zurück. In der Praxis werden aber häufig ganze Zahlen benötigt, die beispielsweise zwischen 1 und 100 liegen. Dies lässt sich mithilfe des folgenden Ausdrucks bewerkstelligen:

```
Int((<Obergrenze> - <Untergrenze> + 1) * Rnd +  
<Untergrenze>)
```

In diesem Ausdruck brauchen Sie nun nur noch *<Obergrenze>* und *<Untergrenze>* durch die entsprechenden Zahlenwerte (z. B. 1 und 100) zu ersetzen. Das Ergebnis ist eine ganze Zahl zwischen den beiden Grenzwerten. Probieren Sie es einfach einmal im Testfenster aus:

```
Print Int((100 - 1 + 1) * Rnd + 1)
```

Da der obige Ausdruck unübersichtlich ist, lässt er sich schlecht handhaben. Wesentlich eleganter lässt sich das Problem mithilfe einer selbstgeschriebenen Funktion lösen. Obwohl Sie den Umgang mit Funktionen erst im Abschnitt *Parametervariablen* kennen lernen, sei hier schon einmal ein kleiner Vorausblick erlaubt:



Hinweis




```
Public Function ZufallszahlAusgeben (Untergrenze,  
Obergrenze)
```

```
    ZufallszahlAusgeben = _  
        Int((Obergrenze - Untergrenze + 1) _  
            * Rnd + Untergrenze)
```

```
End Function
```

Wenn Sie diese Prozedur in ein Standardmodul eingeben, können Sie die Zufallszahl mit dem folgenden – wesentlich übersichtlicheren Ausdruck – ermitteln:

```
Print ZufallszahlAusgeben (1, 100)
```

Logische Ausdrücke

Bis jetzt haben Sie nur Ausdrücke kennen gelernt, die einen numerischen Wert zum Ergebnis haben. Eine weitere Klasse von Ausdrücken bilden die logischen Ausdrücke.

True und False

Logische Ausdrücke können nur zwei Zustände besitzen: Wahr oder Falsch. In VBA werden diese beiden Zustände durch die Schlüsselwörter *True* und *False* dargestellt. Bei der Ausgabe dieser beiden Zustände im Testfenster verwendet Access allerdings die deutschen Bezeichnungen *Wahr* und *Falsch*, wovon Sie sich leicht überzeugen können:

```
Print True  
Print False
```

Als Ergebnis gibt Access im Testfenster *Wahr* und *Falsch* aus. Innerhalb von VBA dürfen Sie allerdings nur die englischen Bezeichnungen verwenden!

Logische Ausdrücke können über logische Operatoren miteinander verknüpft werden. Ein verknüpfter Ausdruck wird hierbei mit den Operatoren höchster Priorität zuerst ausgewertet. Bei Operatoren gleicher Priorität erfolgt die Auswertung von links nach rechts. Access stellt die folgenden logischen Operatoren zur Verfügung:

Logische Operatoren

Operator	Priorität	Bedeutung
<i>Not</i>	1	Negierung
<i>And</i>	2	Und-Verknüpfung
<i>Or</i>	3	Oder-Verknüpfung
<i>Xor</i>	4	Exklusive Oder-Verknüpfung



Logische Operatoren (Forts.)

Operator	Priorität	Bedeutung
<i>Eqv</i>	5	Äquivalent-Verknüpfung
<i>Imp</i>	6	Impliziert-Verknüpfung

Tabelle 3.9

Die folgende Tabelle enthält einige Beispiele zur Verwendung der logischen Operatoren:

Beispiele zur Verwendung logischer Operatoren

Ausdruck	Ergebnis	Ausgabe im Testfenster
<i>True And True</i>	True	Wahr
<i>True And False</i>	False	Falsch
<i>True Or False</i>	True	Wahr
<i>False Or False</i>	False	Falsch
<i>Not False</i>	True	Wahr
<i>True Xor True</i>	False	Falsch
<i>Not Not True</i>	True	Wahr
<i>Not False XOr True</i>	False	Falsch

Tabelle 3.10

Genau wie bei numerischen Operatoren können Sie auch bei logischen Operatoren die Priorität der Auswertung durch Einklammern von Teilausdrücken verändern.

Logische Ausdrücke werden insbesondere im Zusammenhang mit der *If...Then*-Anweisung verwendet, die Sie bereits in *Kapitel 2 Schnellstart VBA* kennen gelernt haben.



Hinweis

Vergleichsoperatoren

Besonders wichtig im Zusammenhang mit logischen Ausdrücken sind die Vergleichsoperatoren. Sie ermöglichen es, zwei beliebige Ausdrücke gleichen Typs miteinander zu vergleichen. Das Ergebnis eines Vergleichs ist immer ein logischer Ausdruck; also *True* oder *False*.



Einer der wichtigsten Vergleichsoperatoren ist der ==-Operator. Der Ausdruck

```
5 = 2 + 3
```

hat beispielsweise *True* zum Ergebnis, da der Ausdruck links des Gleichheitszeichens mit dem Ausdruck rechts davon übereinstimmt. Sie können ihn im Testfenster überprüfen, indem Sie hier

```
Print 5 = 2 + 3
```

eingeben.

Als Ergebnis gibt Access *True* aus. Access stellt folgende Vergleichsoperatoren zur Verfügung:

Vergleichsoperatoren

Operator	Bedeutung
>	Größer
<	Kleiner
=	Gleich
<>	Ungleich
<=	Kleiner oder gleich
>=	Größer oder gleich

Tabelle 3.11

Priorität von Vergleichsoperatoren

Alle Vergleichsoperatoren haben die gleiche Priorität. Der Ausdruck wird daher von links nach rechts ausgewertet. Um bestimmte Prioritäten zu setzen, können wie bei arithmetischen Operatoren runde Klammern verwendet werden. Die folgende Tabelle enthält einige Beispiele:

Beispiele zur Verwendung von Vergleichsoperatoren

Ausdruck	Ergebnis	Ausgabe im Testfenster
4 = 6	False	Falsch
4 < 6	True	Wahr
4 > 6	False	Falsch
4 <> 6	True	Wahr



Beispiele zur Verwendung von Vergleichsoperatoren (Forts.)

Ausdruck	Ergebnis	Ausgabe im Testfenster
$5 \leq 5$	True	Wahr
$5 < 5$	False	Falsch
<i>True And</i> $(5 < 3)$	False	Falsch
$(5 < 3)$ <i>Or</i> $(7 > 2)$	True	Wahr

Tabelle 3.12

Zeichenketten

Neben numerischen und logischen Ausdrücken unterstützt Access natürlich auch die Arbeit mit Texten. Texte werden innerhalb von VBA als eine Folge von Zeichen interpretiert, die auch *Zeichenkette* oder *String* genannt wird. Um eine Zeichenkette im Programmtext anzugeben, muss diese in Anführungszeichen gestellt werden. Eine gültige Angabe wäre beispielsweise der folgende Ausdruck:

```
"Das ist eine Zeichenkette"
```

Auch im Testfenster können Sie Zeichenketten verwenden:

```
Print "Das ist eine Zeichenkette"
```

Ein Problem tritt auf, wenn die Anführungszeichen Teil der Zeichenkette sein sollen. Dies ist beispielsweise dann der Fall, wenn Sie den Text

```
Mein Name ist "Müller"
```

im Testfenster ausgeben möchten. In diesem Fall müssen die Anführungszeichen, die Teil der Zeichenkette sein sollen, doppelt angegeben werden. Das folgende Beispiel zeigt dies:

```
Print "Mein Name ist ""Müller"""
```

Eine Zeichenkette kann aus Buchstaben und Zahlen bestehen. Darüber hinaus werden einige Sonderzeichen unterstützt. Die Gesamtheit aller unterstützten Zeichen wird auch als *Zeichensatz* bezeichnet. Eine Tabelle aller möglichen Zeichen finden Sie in der Zeichentabelle in der Referenz, welche auf der CD zum Buch enthalten ist. Schriftformatierungen wie kursive Schrift oder bestimmte Schriftgrößen werden von Zeichenketten nicht unterstützt.

Access verwendet innerhalb von VBA als Zeichensatz den Unicode, bei dem ein einzelnes Zeichen aus zwei Byte besteht. Bei der Speicherung von Zeichenketten in Tabellen wird dagegen der ANSI-Zeichensatz verwendet. Die Konvertierung zwischen den beiden Zeichensätzen wird von Access automatisch durchgeführt.

Angabe von
Zeichenketten

Zeichensatz



Hinweis



Verknüpfen von Zeichenketten

Verknüpfen und Vergleichen von Zeichenketten

Genau wie numerische Ausdrücke lassen sich auch Zeichenketten verknüpfen. Das Verknüpfen beschränkt sich jedoch auf das Aneinanderhängen zweier Zeichenketten über den &-Operator. Die folgenden Beispiele zeigen die Funktion des &-Operators:

Funktionsweise des &-Operators

Ausdruck	Ergebnis
"Hallo"	Hallo
"Hans" & "Müller"	HansMüller
"Hans" & " " & "Müller"	Hans Müller

Tabelle 3.13

Zeilenwechsel

Wenn Sie einen mehrzeiligen Textausdruck erzeugen möchten, müssen Sie der Zeichenkette einen Zeilenwechsel hinzufügen. Access stellt zu diesem Zweck die Konstante `vbCrLf` zur Verfügung, die die beiden ANSI-Zeichen mit den Codes 13 und 10 repräsentiert (`Chr(13)+Chr(10)`). Die folgende Anweisung zeigt eine Meldung mit einem zweizeiligen Text an:

```
MsgBox "Dies ist ein" & vbCrLf & "zweizeiliger Text"
```

Abb. 3.16

Eine zweizeilige
Meldung



Vergleichen von Zeichenketten

Beim Vergleich zweier Zeichenketten stehen die gleichen Operatoren wie bei numerischen Ausdrücken zur Verfügung. Der `=`-Operator kontrolliert beispielsweise, ob zwei Zeichenketten identisch sind. Hierbei wird die Groß- und Kleinschreibung nicht berücksichtigt. Wenn Sie beispielsweise den Ausdruck

```
Print "Hallo" = "hallo"
```

in das Datenbankfenster eingeben, gibt Access *True* zurück. Der Vergleich

```
Print "Hallo" = "Hall"
```

führt dagegen zum Ergebnis *False*.



Ob in Access beim Vergleich von Zeichenketten die Groß- und Kleinschreibung berücksichtigt wird, legen Sie mithilfe der Anweisung *Option Compare* fest:

```
Option Compare {Binary | Text | Database}
```

Bei dieser Anweisung handelt es sich um eine spezielle Anweisung, die im Deklarationsteil eines Moduls angegeben werden muss und sich nur auf das betreffende Modul auswirkt.

In der Befehlssyntax sind hinter der *Option Compare*-Anweisung drei Schlüsselwörter angegeben, die durch das |-Zeichen voneinander abgegrenzt werden. Außerdem werden diese drei Schlüsselwörter in geschweifte Klammern eingeschlossen. Dies bedeutet, dass Sie genau eines der drei Schlüsselwörter auswählen und angeben können. Die drei Schlüsselwörter haben folgende Bedeutungen:

Die *Option Compare*-Anweisung

Schlüsselwort	Bedeutung
<i>Binary</i>	Berücksichtigt die Groß- und Kleinschreibung.
<i>Text</i> oder <i>Database</i>	Berücksichtigt die Groß- und Kleinschreibung nicht.

Tabelle 3.14

Standardmäßig trägt Access in den Deklarationsteil aller neuen Module die Anweisung

```
Option Compare Database
```

ein. Wenn Sie diese Anweisung löschen, verwendet Access automatisch die Einstellung *Binary*. Bei dieser Einstellung führt beispielsweise der Vergleich

```
Print "Hallo" = "hallo"
```

zu dem Ergebnis *False*.

Wenn Sie überprüfen möchten, ob zwei Zeichenketten nicht identisch sind, können Sie den <>-Operator verwenden. Der folgende Ausdruck hat *True* zum Ergebnis, da sich die beiden Zeichenketten unterscheiden:

```
Print "Meier" <> "Meyer"
```

Das gleiche Ergebnis ließe sich auch mit einer Kombination aus dem =-Operator und dem *Not*-Operator erzielen:

```
Print Not ("Meier" = "Meyer")
```

Option Compare-
Anweisung





<, >, <=, >=

Kontrolle der
Sortierreihenfolge

Die Funktion der übrigen Operatoren wie < oder > bezieht sich auf die alphabetische Sortierreihenfolge. Der Ausdruck

```
"a" < "b"
```

führt zu dem Ergebnis *True*, da der Buchstabe a im Alphabet vor dem Buchstaben b liegt. Enthält die Zeichenkette mehrere Buchstaben, nimmt Access den Vergleich von links nach rechts für jeden einzelnen Buchstaben vor. Das folgende Beispiel hat den logischen Wert *True* zum Ergebnis, da der dritte Buchstabe der ersten Zeichenkette im Alphabet unterhalb des entsprechenden Buchstabens der zweiten Zeichenkette liegt:

```
Print "Meier" < "Meyer"
```

Option Compare
Text und *Option*
Compare Database

Die *Option Compare*-Anweisung wirkt sich nicht nur auf den =-Operator, sondern auch auf die übrigen Operatoren und damit die Sortierreihenfolge aus. Die Option *Binary* bewirkt in diesem Zusammenhang, dass Access alle kleingeschriebenen Buchstaben hinter großgeschriebenen anordnet. Genauer gesagt stimmt die Sortierreihenfolge mit der Reihenfolge überein, mit der die einzelnen Zeichen in der ANSI-Zeichentabelle angeordnet sind.

Die Optionen *Text* und *Database* führen dagegen zu der gewohnten Sortierreihenfolge. Der Unterschied zwischen diesen beiden Optionen besteht lediglich darin, dass die Option *Text* die Ländereinstellungen aus der Systemsteuerung von Windows berücksichtigt. Die *Database*-Option berücksichtigt dagegen die Ländereinstellungen, die zu dem Zeitpunkt aktiv waren, als die Datenbank erstellt oder zuletzt komprimiert wurde. Dies ist besonders bei der Weitergabe von Datenbanken zu berücksichtigen.

Der *Like*-Operator

Mit dem *Like*-Operator stellt Access einen weiteren Operator zur Verfügung, der speziell auf den Vergleich von Zeichenketten zugeschnitten ist. Von der Arbeit mit Access ist Ihnen dieser Operator bestimmt schon unter dem deutschen Namen *Wie* bekannt.

Der *Like*-Operator ermöglicht den Vergleich einer Zeichenkette mit einem Muster, das mit einem Lückentext vergleichbar ist. Die Befehlssyntax des *Like*-Operators hat folgenden Aufbau:

```
<Zeichenkette> Like <Muster>
```

<Muster> ist ebenfalls eine Zeichenkette, die aus Buchstaben, Sonderzeichen und einigen besonderen Platzhalterzeichen bestehen kann. Platzhalterzeichen stehen für einen einzelnen oder eine Gruppe anderer Buchstaben. So liefert der *Like*-Operator beispielsweise bei dem Muster *M?ier* das Ergebnis *True*, wenn <Zeichenkette> eine der Zeichenketten *Meier* oder *Maier* ist:

```
"Meier" Like "M?ier"  
"Maier" Like "M?ier"
```



Das Ergebnis ist außerdem bei jeder anderen Zeichenkette *True*, die mit *M* beginnt, an zweiter Stelle ein beliebiges Zeichen besitzt und mit *ier* endet.

Die folgende Tabelle enthält die wichtigsten Platzhalterzeichen, die zusammen mit dem *Like*-Operator eingesetzt werden können.

Platzhalterzeichen des *Like*-Operators

Zeichen	Beschreibung
?	Platzhalter für ein beliebiges Zeichen (Buchstabe, Ziffer, Sonderzeichen)
*	Platzhalter für mehrere beliebige Zeichen (Buchstaben, Ziffern, Sonderzeichen) oder kein Zeichen
#	Platzhalter für eine beliebige Ziffer (0 bis 9)

Tabelle 3.15

Die folgende Tabelle zeigt einige Beispielemuster und Zeichenketten, bei denen der *Like*-Operator zum Ergebnis *True* oder *False* führt:

Beispiele zum *Like*-Operator

<Muster>	<i>True</i>	<i>False</i>
<i>M*er</i>	Maier, Meier, Mer	Mayr
<i>o*1</i>	001,00001,010101	101, 000
<i>A???A</i>	AbbbA, AaBcA	bbbbA, AA, ABA
<i>a##a</i>	a11a, a67a,	A0a, A7A, AA

Tabelle 3.16



Zeichenkettenfunktionen

Access stellt eine Reihe von Funktionen zur Verfügung, mit denen Zeichenketten manipuliert werden können. Die folgende Tabelle gibt Ihnen eine Übersicht:

Wichtige Zeichenkettenfunktionen

Funktion	Beschreibung
<i>Left</i> (<String>, <Length>)	Liefert den linken Teil einer Zeichenkette.
<i>Right</i> (<String>, <Length>)	Liefert den rechten Teil einer Zeichenkette.
<i>Mid</i> (<String>, <Start>, <Length>)	Gibt einen Teil der Zeichenkette zurück.
<i>UCase</i> (<String>)	Wandelt die Zeichenkette in Großschrift um.
<i>LCase</i> (<String>)	Wandelt die Zeichenkette in Kleinschrift um.
<i>LTrim</i> (<String>)	Gibt eine Zeichenkette zurück, bei der alle führenden Leerzeichen abgeschnitten sind.
<i>RTrim</i> (<String>)	Gibt eine Zeichenkette zurück, bei der alle nachfolgenden Leerzeichen abgeschnitten sind.
<i>Trim</i> (<String>)	Schneidet alle führenden und nachfolgenden Leerzeichen ab.
<i>Len</i> (<Zeichenkette>)	Ermittelt die Anzahl der Zeichen der Zeichenkette. Im Gegensatz zu den anderen Funktionen dürfen bei dieser Funktion keine benannten Parameter verwendet werden.
<i>Chr</i> (<CharCode>)	Gibt ein einzelnes Zeichen zurück, das dem Zeichencode der ANSI-Zeichentabelle entspricht.
<i>Asc</i> (<String>)	Ermittelt den ANSI-Code des Zeichens.
<i>InStr</i> (<Parameterliste>)	Erlaubt das Durchsuchen einer Zeichenkette nach einem bestimmten Text. Den Aufbau der <Parameterliste> entnehmen Sie dem entsprechenden Abschnitt.

Tabelle 3.17



Left-, Right- und Mid-Funktion

Die Funktionen *Left* und *Right* werden dazu verwendet, einen bestimmten Teil einer Zeichenkette zu ermitteln. Die *Left*-Funktion gibt hierbei den Anfang, die *Right*-Funktion das Ende einer Zeichenkette zurück. Die Anzahl der Zeichen, die auf diese Weise abgeschnitten werden sollen, muss jeweils als numerischer Wert unter *<Length>* angegeben werden.

Die Funktion *Mid* bietet die Möglichkeit, auch Teile inmitten einer Zeichenkette auszuschnneiden. Hierzu muss unter *<Start>* die Position des Zeichens angegeben werden, bei dem Sie mit dem Ausschneiden beginnen möchten. Das erste Zeichen der Zeichenkette hat hierbei die Position 1, das zweite die Position 2 usw. Das Argument *<Length>* gibt wieder die Anzahl der Zeichen an, welche die Funktion angefangen von der Startposition zurückgeben soll.

Die folgende Tabelle enthält einige Beispiele zu den Funktionen. Sie können diese wie gewohnt mit voranstehendem *Print*-Befehl im Direktfenster ausprobieren.

Beispiele zu den Funktionen Left, Right und Mid

Ausdruck	Ergebnis
Left("Willi Meier", 5)	"Willi"
Left("Willi Meier", 1)	"W"
Right("Willi Meier", 4)	"eier"
Mid("Willi Meier", 3, 2)	"ll"
Mid("Willi Meier", 8, 1)	"e"
Mid("Willi Meier", 7)	"Meier"
Left(String:="Meier", Length:=1)	"W"

Tabelle 3.18

Wenn Sie für *<Start>* und *<Length>* ungültige Werte angeben, meldet Access trotzdem keinen Fehler. Access interpretiert Ihre Angaben in diesem Fall möglichst sinnvoll und gibt ein entsprechendes Ergebnis zurück. Wenn Sie beispielsweise die *Left*-Funktion auf eine 20 Zeichen lange Zeichenkette anwenden und unter *<Length>* einen Wert größer als 20 angeben, gibt Access die komplette Zeichenkette zurück. Fehler treten nur dann auf, wenn Sie negative Zahlenwerte angeben.



Hinweis

Access stellt unter dem Namen *Mid* sowohl eine Funktion als auch eine Anweisung zur Verfügung. Die *Mid*-Funktion haben Sie bereits kennen gelernt. Die *Mid*-Anweisung dient dagegen dazu, in einer Zeichenkette gezielt ein oder mehrere Zeichen auszutauschen. Um mit dieser Funktion sinnvoll arbeiten zu können, sind Variablen erforderlich. Die folgende Prozedur gibt Ihnen einen Vorgeschmack:



Hinweis





```
Sub Testprozedur3()  
  
Dim Text As String  
  
    Text = "Willi Meier"  
    Mid(Text, 8, 1) = "a"  
    MsgBox Text  
  
End Sub
```

Die Prozedur definiert eine Variable mit dem Namen *Text*, in der die Zeichenkette *Willi Meier* gespeichert wird. Anschließend wird mithilfe der *Mid*-Anweisung der erste Buchstabe *e* gegen ein *a* ausgetauscht. Das Ergebnis *Willi Maier* wird in einem Meldungsfenster ausgegeben.

UCase-, und LCase-Funktionen

Die beiden Funktionen *UCase* und *LCase* haben die Aufgabe, einen Text komplett in Groß- bzw. Kleinschreibung umzuwandeln. Die Zeichenkette, die umgewandelt werden soll, muss als Argument angegeben werden. Das Ergebnis der Funktionen ist eine neue Zeichenkette.

LTrim-, Rtrim-, und Trim-Funktionen

Mithilfe der Funktionen *LTrim* und *RTrim* können Sie voranstehende oder nachfolgende Leerzeichen von einer Zeichenkette abschneiden. Die Funktion *Trim* übernimmt gleich beide Aufgaben der Funktionen *LTrim* und *RTrim* in einem Arbeitsschritt. Die beiden Funktionen werden insbesondere zur Bearbeitung von Benutzereingaben verwendet. Hier kommt es beispielsweise oft vor, dass vor dem Namen aus Versehen ein Leerzeichen eingegeben wird. Mithilfe der *Trim*-Funktionen können Sie dieses problemlos abschneiden. Ein weiterer Einsatzbereich liegt z. B. im Zusammenfügen von Namen, die in einer Tabelle in den Datenfeldern *Vorname* und *Nachname* gespeichert sind. Den vollständigen Namen erhalten Sie dann mithilfe des folgenden Ausdrucks:

```
Trim (Vorname & " " & Nachname)
```

Wenn nun die Variable *Vorname* keinen Inhalt besitzt – wie beispielsweise bei Firmen üblich –, stellt Access dem Nachnamen trotzdem ein Leerzeichen voran. Durch den Einsatz der *Trim*-Funktion wird dieses wieder abgeschnitten.

Len-Funktion

Die *Len*-Funktion ermittelt die Anzahl der Zeichen, die sich in einer Zeichenkette befinden. Hierbei werden auch Leerzeichen mitgezählt, die sich am Anfang oder Ende der Zeichenkette befinden. Die *Len*-Funktion eignet sich besonders gut zur Überprüfung, ob eine Zeichenkette überhaupt einen Inhalt besitzt. Solche Zeichenketten werden auch als *leere Zeichenketten* bezeichnet und innerhalb von VBA



durch zwei direkt aufeinander folgende Anführungszeichen angegeben ("""). Die *Len*-Funktion gibt in diesem Fall das Ergebnis 0 zurück. Die folgende Tabelle enthält einige Beispiele zur *Len*-Funktion:

Beispiele zur Funktion *Len*

Ausdruck	Ergebnis
<i>Len</i> ("Willi Meier")	11
<i>Len</i> ("")	0
<i>Len</i> (" Willi Meier ")	13
<i>Len</i> (Trim(" Willi Meier "))	11

Tabelle 3.19

Chr- und Asc-Funktionen

Die Funktionen *Chr* und *Asc* haben besondere Bedeutung beim Umgang mit der ANSI-Zeichensatztabelle, die Sie im Anhang dieses Buches finden. In dieser Tabelle sind alle Zeichen aufgeführt, die innerhalb einer Zeichenkette verwendet werden können. Jedem Zeichen ist hierbei eine Zahl zugeordnet, die auch als *Zeichencode* bezeichnet wird.

Um den Zeichencode eines Zeichens von VBA aus zu ermitteln, kann die *Asc*-Funktion eingesetzt werden. Diese Funktion erwartet eine Zeichenkette als Parameter. Das Ergebnis ist der Zeichencode des ersten Zeichens dieser Zeichenkette.

Das Gegenstück zur *Asc*-Funktion ist die *Chr*-Funktion, die zu einem bestimmten Zeichencode das passende Zeichen ermittelt. Das Ergebnis der Funktion ist eine Zeichenkette, die aus nur einem Zeichen besteht.

Die folgende Tabelle enthält einige Beispiele zu den beiden Funktionen. Beachten Sie, dass Kleinbuchstaben einen anderen Zeichencode als die entsprechenden Großbuchstaben haben!

Beispiele zu den Funktionen *Asc* und *Chr*

Ausdruck	Ergebnis
<i>Asc</i> ("a")	97
<i>Asc</i> ("A")	65
<i>Asc</i> ("Auto")	65

Beispiele zu den Funktionen Asc und Chr (Forts.)

Ausdruck	Ergebnis
Asc("i")	49
Chr(97)	a
Chr(32)	" " (Leerzeichen)
Chr(34)	" (Anführungszeichen)

Tabelle 3.20

Sonderzeichen

Die Zeichencodes 1 bis 31 repräsentieren Sonderzeichen, die nicht direkt als einzelne Zeichen ausgegeben werden können. Einige dieser Zeichen sind für den praktischen Einsatz wichtig. Die folgende Tabelle enthält alle vier Zeichencodes unterhalb des Wertes 32, die für die Verwendung innerhalb von VBA geeignet sind:

Zeichencodes von Sonderzeichen

Code	Beschreibung
8	<i>Backspace</i> -Zeichen. Wird dieses Zeichen ausgegeben, löscht Access automatisch das links davon liegende Zeichen. Dieses Zeichen hat die gleichen Auswirkungen wie die Betätigung der  -Taste.
9	<i>Tab</i> -Zeichen. Fügt bei der Ausgabe der Zeichenkette einen Tabulatorvor-schub ein.
10	<i>LineFeed</i> -Zeichen. Fügt bei der Ausgabe der Zeichenkette einen Zeilen-wechsel ein.
13	<i>Carriage Return</i> -Zeichen. Markiert das Ende eines Absatzes.

Tabelle 3.21

InStr-Funktion

Die *InStr*-Funktion erlaubt die Suche nach einem bestimmten Text innerhalb einer Zeichenkette. Die komplette Syntax der Funktion besitzt folgenden Aufbau:

`InStr ([<Start>], <String1>, <String2> [, <Compare>])`

Die Zeichenkette, die durchsucht werden soll, muss unter *<String1>* angegeben werden. Die Zeichenkette, die Sie suchen möchten, geben Sie dagegen unter *<String2>* an. Wahlweise können Sie unter *<Start>* eine Zeichenposition angeben, bei der die Suche begonnen werden soll.



Die Suche wird grundsätzlich von links nach rechts durchgeführt. Wird der Suchtext gefunden, gibt die Funktion die Position des ersten Zeichens als numerischen Wert aus. Sollte der gesuchte Text mehrmals in der Zeichenkette auftauchen, gibt die Funktion die Position der ersten übereinstimmenden Textstelle an. Konnte der Suchtext nicht gefunden werden, ist das Ergebnis der Funktion der Wert 0.

Mit dem optionalen Argument *<Compare>* bestimmen Sie, ob Access beim Vergleich die Groß- und Kleinschreibung beachtet. Geben Sie hier entweder den Wert 0 (die Groß- und Kleinschreibung wird beachtet) oder den Wert 1 (die Groß- und Kleinschreibung wird nicht beachtet) an. Wenn Sie das Argument auslassen, verwendet Access die Standardvorgabe 0.

Die Verwendung benannter Parameter ist bei der *InStr*-Funktion nicht zulässig. Außerdem dürfen Sie das Argument *<Start>* nicht auslassen, wenn Sie das Argument *<Compare>* angeben.



Hinweis

Die folgende Tabelle enthält einige Beispiele:

Beispiele zur *InStr*-Funktion

Ausdruck	Ergebnis
<i>InStr</i> (1, "Dies ist ein Text", "ein")	10
<i>InStr</i> ("Dies ist ein Text", "ein")	10
<i>InStr</i> ("Dies ist ein Text", "t")	8
<i>InStr</i> (9, "Dies ist ein Text", "t")	14
<i>InStr</i> ("Dies ist ein Text", "Hallo")	0
<i>InStr</i> ("Dies ist ein Text", " ")	5
<i>Debug.Print InStr</i> (1, "aaaaaAaa", "A", 0)	6
<i>Debug.Print InStr</i> (1, "aaaaaAaa", "A", 1)	1

Tabelle 3.22

Die *InStr*-Funktion kann beispielsweise dazu verwendet werden, einen kompletten Namen wie "Willi Müller" in einen Vornamen und einen Nachnamen zu zerlegen. Dazu muss zunächst mithilfe der *InStr*-Funktion die Position des Leerzeichens ermittelt werden. Anschließend kann die Zeichenkette mit den Funktionen *Left* und *Mid Right* zerlegt werden. Die folgende Prozedur zeigt die entsprechende Vorgehensweise. Auch hierbei ist wieder ein kleiner Vorgriff auf den Abschnitt *Variablen* erforderlich:





```

Sub Testprozedur4()

Dim Vorname As String
Dim Nachname As String
Dim KompletterName As String
Dim LeerzeichenPosition As Integer

KompletterName = "Willi Müller"
LeerzeichenPosition = InStr(KompletterName, " ")
Vorname = Left(KompletterName, _
    LeerzeichenPosition - 1)
Nachname = Right(KompletterName, _
    Len(KompletterName) _
    - LeerzeichenPosition)
Nachname = Mid(KompletterName, _
    LeerzeichenPosition + 1)

MsgBox Vorname
MsgBox Nachname

End Sub

```

Hinweis



Bei der obigen Prozedur ergeben sich Probleme, wenn der Name aus mehreren einzelnen Wörtern besteht und damit mehrere Leerzeichen beinhaltet (z. B. "Dieter von Statzen"). Ob das Ergebnis trotzdem noch brauchbar ist, hängt von dem Eingabeformat der Namen und dem Verwendungszweck ab.

Ein weiteres Problem ergibt sich, wenn der komplette Name überhaupt kein Leerzeichen enthält. In diesem Fall gibt die *InStr*-Funktion den Wert 0 zurück. Da von dieser Position in der folgenden Programmzeile der Wert 1 subtrahiert wird, besitzt der Ausdruck *LeerzeichenPosition - 1* den Wert -1. Dieser Wert wird aber von der *Left*-Funktion nicht akzeptiert und führt daher zu einem Fehler. Um das Problem zu lösen, müsste das Ergebnis 0 der *InStr*-Funktion mithilfe einer *If*-Abfrage abgefangen werden, die Sie bereits aus dem zweiten Kapitel dieses Buches kennen.

Datums- und Zeitausdrücke

Access stellt spezielle Funktionen zur Unterstützung von Datums- und Zeitwerten zur Verfügung. Ein Datum oder eine Zeit wird von Access intern als ein numerischer Ausdruck behandelt. Dazu findet eine Dezimalzahl Verwendung, deren Zahlenwert vor dem Komma als Datum und nach dem Komma als Uhrzeit interpretiert wird. Dieses Format wird auch als *serielles Datumsformat* bezeichnet.

Aufbau des seriellen Datumsformats

Der Teil vor dem Komma gibt die Anzahl der Tage an, die seit dem 31. Dezember des Jahres 1899 verstrichen sind. Tage vor diesem Datum werden mit negativen Werten dargestellt. Der Teil nach dem Komma gibt die Anzahl der Sekunden seit Mitternacht an. Aus diesen beiden Angaben kann Access ein eindeutiges Datum und eine eindeutige Uhrzeit ermitteln.



Bei der Arbeit mit Datumsausdrücken tritt das serielle Datumsformat allerdings selten zutage. Sowohl bei der Eingabe eines Datums als auch bei der Ausgabe führt Access automatisch einer Konvertierung zwischen dem seriellen Datumsformat und der gewohnten Schreibweise (z. B. 13.8.01) durch.

Ein festes Datum oder eine feste Uhrzeit kann innerhalb von VBA ähnlich wie eine Zeichenkette angegeben werden. Anstelle von Anführungszeichen muss der komplette Ausdruck allerdings von zwei Nummernzeichen (#) umgeben sein. Außerdem muss das Datum im amerikanischen Format in der Form Monat/Tag/Jahr angegeben werden. Bei Datumsangaben wären beispielsweise folgende Ausdrücke gültig, die alle das Datum 30.1.2001 zum Ergebnis haben:

Datumsausdrücke

Datums- und Zeitangaben

Ausdruck	Ausdruck
#Jan 30 01#	#Jan. 30 2001#
#1/30/2001#	#January 30 2001#
#1-30-2001#	#January 30 01#

Tabelle 3.23

Sie können diese Beispiele ebenfalls im Direktfenster ausprobieren. Stellen Sie dazu dem Ausdruck den *Print*-Befehl voran (z. B. "Print #1/30/2001#"). Bei der Ausgabe des Datums verwendet Access die vertraute Datumsschreibweise (tt.mm.jj) gemäß den Ländereinstellungen der Windows-Systemsteuerung.

Wenn Sie ein Datum innerhalb einer Prozedur angeben, wandelt Access es direkt nach der Eingabe in das Format m/t/j um. Die Jahreszahlen 30 bis 99 werden als Jahre 1900 bis 1999, die Jahreszahlen 0 bis 29 als Jahre 2000 bis 2029 interpretiert. Wenn Sie auf diese Weise ein Jahr innerhalb einer Prozedur angeben, ergänzt Access noch während der Eingabe das aktuelle Jahr. Beachten Sie, dass dieses nur bei der Eingabe von Programmzeilen in eine Prozedur und nicht im Testfenster der Fall ist.

Monatsnamen werden in der kurzen (z. B. *Jan*) oder der vollständigen (z. B. *January*) Schreibweise akzeptiert. Auch hier wandelt Access die Monatsnamen während der Eingabe in einen Zahlenwert um.

Angabe von Monatsnamen

Eine Zeitangabe muss ebenfalls innerhalb von Nummernzeichen erfolgen. Das Format entspricht hierbei der Form

Zeitangaben

#<Stunde>:<Minute>[:<Sekunde>]#

wobei die Angabe der Sekunden optional ist. Gültige Zeitangaben sind beispielsweise #10:00# oder #20:30:10#. Nach der Eingabe wandelt Access die Zeitangabe in folgendes Format um:



#hh:mm:ss {AM|PM} #

Darüber hinaus kann eine Zeit in Kombination mit einem Datum angegeben werden. Die Zeit muss hierbei entweder vor oder hinter dem Datum angeordnet werden. Die folgende Tabelle enthält einige Beispiele:

Datums- und Zeitausdrücke

Ausdruck

#Sept/22/01 20:00#

#20:00#

#20:00:00 9/22/01#

Tabelle 3.24

Hinweis



Das aktuelle Datum Ihres Rechners erhalten Sie mithilfe der *Date*-Funktion. Auf ähnliche Weise gibt die *Time*-Funktion die aktuelle Uhrzeit zurück. Eine Kombination aus aktuellem Datum und aktueller Uhrzeit liefert die *Now*-Funktion. Die drei Funktionen sind Ihnen vielleicht bereits unter ihren deutschen Namen *Datum*, *Zeit* und *Jetzt* bekannt. Die folgenden Beispiele können Sie im Testfenster eingeben:

```
Print Date
Print Time
Print Now
```

Ändern des Systemdatums und der Systemzeit

Mit *Date* und *Time* können Sie allerdings auch den umgekehrten Weg gehen und das Systemdatum bzw. die Systemzeit verändern. Um dies auszuprobieren, geben Sie die Anweisung

```
date = #10/01/2001#
```

ein. Wenn Sie nun die Systemsteuerung aufrufen, sehen Sie, dass diese Anweisung direkten Einfluss auf Ihre Windows-Konfiguration – nämlich das Systemdatum – genommen hat. Mit der *Time*-Anweisung können Sie entsprechend die Systemzeit verändern.

Datumsoperationen und Datumsfunktionen

Addieren von Tagen

Da sich hinter Datums- und Zeitangaben ganz normale numerische Ausdrücke verbergen, können auch alle numerischen Operatoren wie + oder * angewendet werden. Wenn Sie beispielsweise das Datum ermitteln möchten, an dem eine Rechnung zu zahlen ist, brauchen Sie zu dem Datumswert einfach nur die Rechnungsfrist in Tagen zu addieren. So führt beispielsweise die Berechnung

```
#7/20/01# + 30
```



zu dem Ergebnis 19.08.01. Schaltjahre und sonstige Besonderheiten des Kalenders werden hierbei natürlich genau berücksichtigt. Sie können sich von dem Ergebnis im Testfenster überzeugen, indem Sie hier folgende Anweisung eingeben:

```
Print #7/20/01# + 30
```

Neben dem `+`-Operator kann auch der `-`-Operator eingesetzt werden. Bei allen anderen numerischen Operatoren wie `*` oder `Mod` löst Access allerdings das Datumsformat auf und zeigt das Ergebnis als Zahl an. Der Grund dafür ist, dass mit diesen Operatoren keine sinnvollen Datumsberechnungen möglich sind.

Die *DateAdd*-Funktion

In der Praxis ist es oft notwendig, zu einem Datumswert nicht nur eine bestimmte Anzahl von Tagen, sondern z. B. Wochen oder Jahre zu addieren. Zu diesem Zweck stellt Access die *DateAdd*-Funktion zur Verfügung:

```
DateAdd(<Interval>, <Number>, <Date>)
```

Das Argument *<Date>* gibt das Datum an, von dem aus die Berechnung erfolgen soll. Mit *<Interval>* bestimmen Sie, welche Zeitintervalle Sie addieren bzw. subtrahieren möchten. Dies können beispielsweise Tage, Wochen oder Quartale sein. Dazu geben Sie unter *<Interval>* eine der folgenden Zeichenketten an:

Angaben für das Argument *<Interval>*

<Interval>	Bedeutung
yyy	Jahr
q	Quartal
m	Monat
y	Kalendertag
d	Tag
w	Wochentag
ww	Woche
h	Stunde
n	Minute
s	Sekunde

Tabelle 3.25



Wie Sie sehen, enthält die Tabelle auch drei Intervalle, mit denen sich Zeitberechnungen durchführen lassen. Sie können dazu unter *<Date>* entweder eine Kombination aus Datum und Uhrzeit oder ausschließlich eine Uhrzeit angeben. Wenn Sie lediglich ein Datum angeben und eine Zeitberechnung durchführen, geht Access von Mitternacht (00:00) des angegebenen Datums aus.

Hinweis



Beachten Sie, dass das Intervall *Minuten* über den Buchstaben *n* angegeben wird. Dies ist erforderlich, da der Buchstabe *m* bereits für das Intervall *Monat* reserviert ist.

Das Argument *<Number>* bestimmt die Anzahl der Intervalle, die Sie addieren möchten. Wenn Sie hier einen negativen Wert angeben, führt Access eine Subtraktion durch. Die folgende Tabelle enthält einige Beispiele:

Beispiele zur *DateAdd*-Funktion

Ausdruck	Ergebnis
<i>DateAdd</i> ("d", 1, #1/1/01#)	02.01.01
<i>DateAdd</i> ("ww", 1, #1/1/01#)	08.01.01
<i>DateAdd</i> ("yyyy", 2, #1/1/01#)	01.01.03
<i>DateAdd</i> ("yyyy", -2, #1/1/01#)	01.01.99
<i>DateAdd</i> ("n", -100, #12:00#)	10:20:00
<i>DateAdd</i> ("h", -100, #1/1/02#)	27.12.01 20:00:00
<i>DateAdd</i> ("s", 100000, #1/1/01 12:00#)	02.01.01 15:46:40
<i>DateAdd</i> ("d", 8, Date)	Abhängig vom aktuellen Datum

Tabelle 3.26

Die *DateDiff*-Funktion

Das Gegenstück zur *DateAdd*-Funktion ist die *DateDiff*-Funktion. Sie ermittelt die Anzahl der Intervalle, die zwischen zwei Tagen liegen:

```
DateDiff(<Interval>, <Date1>, <Date2>  
[ <FirstDayOfWeek>] [, <FirstWeekOfYear>])
```

Unter *<Interval>* geben Sie an, in welchem Intervall Sie den Abstand messen möchten. Die Angabe muss in der gleichen Form erfolgen wie vorher bei der *DateAdd*-Funktion. *<Date1>* und *<Date2>* geben die beiden Datumswerte an. Das Ergebnis der Funktion ist die Anzahl der Intervalle. Sollte *<Date2>* vor *<Date1>* liegen, ist das Ergebnis negativ.



Das Argument *<FirstDayOfWeek>* wirkt sich aus, wenn Sie als Intervall Wochen (ww) ausgewählt haben. Bei der Standardeinstellung – die dann aktiv ist, wenn Sie das Argument auslassen – addiert Access immer dann eine neue Woche, sobald der Sonntag verstrichen ist. Über das Argument *<FirstDayOfWeek>* können Sie einen anderen Tag als Wochenanfang definieren (z.B. Montag), wodurch das Ergebnis in einigen Fällen beeinflusst werden kann. Für das Argument geben Sie einen Zahlenwert an, wobei 1 Sonntag, 2 Montag usw. bedeutet.

Das Argument *<FirstWeekOfYear>* hat eine ähnliche Bedeutung. Hiermit bestimmen Sie, wann für Access die erste Woche eines Jahres beginnt. Auch dabei müssen Sie einen Zahlenwert angeben, den Sie der folgenden Tabelle entnehmen:

Einstellungen für *<FirstWeekOfYear>*

Wert	Bedeutung
0	Verwendet die Systemeinstellung.
1	Die erste Woche beginnt mit dem 1. Januar des Jahres. Diese Einstellung ist die Standardeinstellung, falls Sie das Argument <i><FirstWeekOfYear></i> auslassen.
2	Legt als erste Woche des Jahres diejenige Woche fest, die mindestens vier Tage besitzt.
3	Legt die erste vollständige Woche als erste Woche des Jahres fest.

Tabelle 3.27

Die folgende Tabelle enthält einige Beispiele:

Beispiele zur *DateDiff*-Funktion

Ausdruck	Ergebnis
<i>DateDiff</i> ("d", #1/1/01#, #3/1/01#)	59
<i>DateDiff</i> ("d", #3/1/01#, #1/1/01#)	-59
<i>DateDiff</i> ("s", #1/1/01#, #1/2/01#)	86400
<i>DateDiff</i> ("d", Date, Date +8)	8
<i>DateDiff</i> ("ww", #1/1/01#, #2/1/01#)	4
<i>DateDiff</i> ("ww", #1/1/01#, #2/1/01#, 3)	5

Tabelle 3.28

Die *DatePart*-Funktion

Was noch fehlt, ist eine Funktion zur Ermittlung eines bestimmten Teils eines Datums. Dazu stellt Access die *DatePart*-Funktion zur Verfügung. Mit ihrer Hilfe lässt sich beispielsweise der Monat aus einem Datum herauslösen oder feststellen, in welche Kalenderwoche ein bestimmtes Datum fällt. Die Funktion besitzt folgende Syntax:

```
DatePart(<Interval>, <Date> [,<FirstDayOfWeek>]
[,<FirstWeekOfYear>])
```

Die vier Argumente stimmen mit denen der beiden vorhergehenden Funktionen überein. Es folgen einige Beispiele:

Beispiele zur *DatePart*-Funktion

Ausdruck	Ergebnis
<i>DatePart</i> ("d", #1/1/01#)	1
<i>DatePart</i> ("d", #5/3/01#)	3
<i>DatePart</i> ("m", #5/3/01#)	5
<i>DatePart</i> ("yyyy", #5/3/01#)	2001
<i>DatePart</i> ("ww", #7/1/01#)	27 (Kalenderwoche!)
<i>DatePart</i> ("q", #7/1/01#)	3 (Quartal!)
<i>DatePart</i> ("q", Date)	Das aktuelle Quartal (abhängig vom Tagesdatum)

Tabelle 3.29

Weitere Datumsfunktionen

Neben den beiden vorgestellten Funktionen stellt Access noch einige andere Datums- und Zeitfunktionen zur Verfügung. Eine Übersicht über alle Funktionen gibt Ihnen die folgende Tabelle.

Datums- und Zeitfunktionen

Funktion	Beschreibung
<i>DateAdd</i> (<Interval>, <Number>, <Date>)	Addiert ein Intervall zu einem bestimmten Tag.

Datums- und Zeitfunktionen (Forts.)

Funktion	Beschreibung
<i>DateDiff</i> (<Interval>, <Date1>, <Date2> [, <FirstDayOfWeek>] [, <FirstWeekOfYear>])	Ermittelt die Differenz zwischen zwei Tagen.
<i>DatePart</i> (<Interval>, <Date> [, <FirstDayOfWeek>] [, <FirstWeekOfYear>])	Ermittelt einen bestimmten Teil eines Datums.
<i>Year</i> (<Date>)	Ermittelt das Jahr eines Datums.
<i>Month</i> (<Date>)	Ermittelt den Monat eines Datums.
<i>Day</i> (<Date>)	Ermittelt den Tag eines Datums.
<i>Weekday</i> (<Date>, <FirstDayOfWeek>)	Ermittelt den Wochentag eines Datums. Das Ergebnis ist eine Zahl, wobei 1 dem Sonntag, 2 dem Montag usw. entspricht.
<i>Hour</i> (<Time>)	Ermittelt die Stunde einer Uhrzeit.
<i>Minute</i> (<Time>)	Ermittelt die Minute einer Uhrzeit.
<i>Second</i> (<Time>)	Ermittelt die Sekunde einer Uhrzeit.
<i>DateSerial</i> (<Year>, <Month>, <Day>)	Wandelt die drei numerischen Angaben für Jahr, Monat und Tag in ein komplettes Datum um.
<i>DateValue</i> (<Date>)	Wandelt den Inhalt einer Zeichenkette in ein Datum um. Das Datum kann innerhalb der Zeichenkette im deutschen Format angegeben werden. Deutsche Monatsnamen sind ebenfalls zulässig.

Tabelle 3.30

Das Ergebnis der Funktionen *Year*, *Month*, *Day*, *Weekday*, *Hour*, *Minute* und *Second* könnte auch mithilfe der *DatePart*-Funktion erzielt werden. Um beispielsweise das Ergebnis der *Day*-Funktion zu erhalten, können Sie folgenden Ausdruck verwenden:

```
DatePart ("d", <Datum>)
```

Die sieben Funktionen dienen also nur einer einfacheren Bedienung und führen zur besseren Lesbarkeit des Quelltextes.

Die letzte Funktion *DateValue* eignet sich dazu, eine vom Benutzer getätigte Eingabe in ein serielles Datum umzuwandeln und so innerhalb von Access nutzbar zu machen.



Hinweis



Die folgende Tabelle enthält Beispiele zu den Funktionen:

Beispiele zu den Datums- und Zeitfunktionen

Ausdruck	Ergebnis
<i>Day</i> (#8/23/1968#)	23
<i>DatePart</i> ("d", #8/23/1968#)	23
<i>Hour</i> (#8/23/1968#)	0
<i>Hour</i> (#12:30#)	12
<i>Minute</i> (#12:30#)	30
<i>DateSerial</i> (2001, 3, 5)	05.03.01
<i>DateValue</i> ("1. Januar 2002")	01.01.02

Tabelle 3.31

Variablen

In den vorhergehenden Abschnitten wurden Variablen bereits an der ein oder anderen Stelle erwähnt. Eine Variable lässt sich mit einem Behälter vergleichen, der einen bestimmten Inhalt hat. Dieser Inhalt kann eine Zahl oder eine Zeichenkette sein. Der Zugriff auf den Inhalt einer Variablen erfolgt über den Namen der Variablen, den Sie selbst festlegen können. In der Regel werden Namen verwendet, die die Art des Inhalts beschreiben. Wenn Sie beispielsweise den Namen einer Person in einer Variablen speichern möchten, würden Sie der Variablen den Namen *Name* oder *Personenname* geben. Die Regeln zur Bildung eines Variablennamens entnehmen Sie der folgenden Auflistung:

Namens-
konventionen von
Variablen

- Der Name muss mit einem Buchstaben beginnen.
- Der Name darf nur Buchstaben, Zahlen und das Unterstreichungszeichen enthalten. Leer-, Satz- oder Sonderzeichen sind nicht erlaubt.
- Die Länge darf 255 Zeichen nicht überschreiten.
- Der Name sollte nicht mit einem reservierten Schlüsselwort übereinstimmen (z. B. *If*).



Die Groß- und Kleinschreibung von Variablenamen findet keine Beachtung. Access achtet aber während der Eingabe eines Variablenamens stets darauf, dass an allen Stellen, an denen die Variable angegeben ist, die gleiche Schreibweise verwendet wird. Gegebenenfalls ändert Access die Schreibweise gemäß Ihrer Deklaration.

Deklaration

Bevor Sie eine Variable in einer Prozedur verwenden können, müssen Sie diese üblicherweise erst anmelden. Dieser Vorgang wird auch als *Deklaration* bezeichnet. Bei der Deklaration müssen Sie festlegen, welchen Inhalt die Variable später aufnehmen soll. Dies ist vergleichbar mit den Felddatentypen, die Sie beim Anlegen von Access-Datentabellen zuweisen.

Für Zeichenketten ist der Datentyp *String* zuständig. Er ermöglicht das Speichern von Texten mit einer Maximallänge von ca. zwei Milliarden Zeichen. Die eigentliche Deklaration wird mithilfe der *Dim*-Anweisung vorgenommen:

```
Dim <Variablenamen> As <Datentyp>
```

Um beispielsweise eine Variable mit dem Namen *Nachname* für die Aufnahme von Texten vorzubereiten, geben Sie innerhalb einer Prozedur einfach die folgende Anweisung an:

```
Dim Nachname As String
```

In einigen Spezialfällen kann es vorteilhaft sein, die maximale Anzahl der Zeichen innerhalb einer Zeichenkette zu beschränken. In diesem Fall muss die maximale Anzahl der Zeichen in Kombination mit dem *-Zeichen hinter dem Datentypschlüsselwort *String* angegeben werden. Solche Zeichenketten können maximal 65.535 Zeichen beinhalten. Das folgende Beispiel deklariert eine Zeichenkettenvariable, die maximal 100 Zeichen aufnehmen kann:

```
Dim Nachname As String * 100
```

Wenn Sie in einer Prozedur einmal mehrere Variablen benötigen, können Sie die *Dim*-Anweisung – jeweils in einer eigenen Zeile – mehrmals hintereinander angeben. Alternativ ist es auch möglich, mehrere Variablen mithilfe einer einzelnen *Dim*-Anweisung zu deklarieren. Dazu muss das Paar aus Variablenname und Datentyp mehrmals – jeweils durch Kommas voneinander getrennt – hinter der *Dim*-Anweisung angegeben werden. Die folgende Anweisung deklariert z. B. zwei *String*-Variablen:

```
Dim Nachname As String, Vorname As String
```

Zusätzlich zur Deklaration nimmt die *Dim*-Anweisung auch eine Initialisierung der Variablen vor, indem ihr der Standardinhalt zugewiesen wird. Bei Zeichenketten ist dies eine leere Zeichenkette.

String-Datentyp



Hinweis

Initialisierung von Variablen





Die *Dim*-Anweisung gehört zu den wenigen Anweisungen, die Sie nicht im Testfenster ausprobieren können, sondern die nur im Kontext einer Prozedur eingesetzt werden darf:

```
Sub Testprozedur()  
  
Dim Nachname As String  
  
    [<Anweisungen>]  
  
End Sub
```

Hinweis



Obwohl die *Dim*-Anweisung an jeder Stelle der Prozedur angegeben werden kann, sollten die Deklarationen aller Variablen einer Prozedur direkt am Anfang vorgenommen werden. So ist auf den ersten Blick ersichtlich, welche Variablen in der Prozedur verwendet werden. Unter anderem lassen sich so Fehler leichter ausfindig machen.

Der Zuweisungsoperator =



Um einer Variablen einen neuen Inhalt zuzuweisen, muss der Zuweisungsoperator = verwendet werden, der nicht mit dem Vergleichsoperator = verwechselt werden sollte. Access erkennt an dem Kontext, in dem Sie das Gleichheitszeichen einsetzen, ob es sich um den Zuweisungsoperator oder den Vergleichsoperator handelt. Die Zuweisung eines Inhalts erfolgt gemäß der folgenden Syntax:

```
<Variablenname> = <Ausdruck>
```

Hinweis



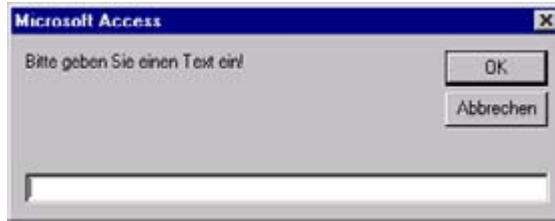
Bei der Zuweisung muss darauf geachtet werden, dass die Variable das Ergebnis von *<Ausdruck>* auch wirklich aufnehmen kann! Wenn es sich bei der Variablen beispielsweise um eine numerische Variable handelt, der Ausdruck aber einen String zum Ergebnis hat, tritt ein Fehler auf.

Die folgende Beispielprozedur zeigt die Anwendung einer Variablen mit dem Namen *Text*. In der Prozedur wird eine Funktion mit dem Namen *InputBox* verwendet, die ein Eingabefenster anzeigt. Der Benutzer hat hier die Gelegenheit, einen beliebigen Text anzugeben und mit *OK* zu bestätigen. Als Parameter wird der Funktion eine Zeichenkette übergeben, die innerhalb des Eingabefensters als erläuternder Text erscheint. Das Ergebnis der Funktion ist der eingegebene Text.

Neben der *OK*-Schaltfläche enthält das Dialogfenster der *InputBox*-Funktion zusätzlich eine *Abbrechen*-Schaltfläche. Wenn der Benutzer diese anklickt, gibt die Funktion eine leere Zeichenkette zurück.

```
Sub Testprozedur5()  
  
Dim Text As String  
  
    Text = InputBox("Bitte geben Sie einen Text ein!")  
  
    MsgBox "Ihre Eingabe: " & Text  
  
End Sub
```





○ Abb. 3.17

Die *InputBox*-Funktion

In dem Beispiel wird die Variable *Text* mithilfe der *Dim*-Anweisung mit dem Datentyp *String* deklariert. Die neue Variable ist damit definiert und kann nun einen Text aufnehmen. In der zweiten Programmzeile wird ein Text durch eine Eingabe vom Benutzer abgefragt und in der Variablen *Text* gespeichert. Mit dem Inhalt dieser Variablen – die bei der Erstellung der Prozedur noch nicht bekannt ist – kann nun gearbeitet werden. In diesem Fall wird der Text in der dritten Programmzeile mit der Zeichenkette *Ihre Eingabe:* verknüpft und über ein Meldungsfenster ausgegeben.



○ Abb. 3.18

Das Ergebnis der Prozedur

Wenn Sie diese Prozedur oder alle folgenden Beispielprozeduren ausprobieren möchten, legen Sie einfach ein neues Modul an und geben die Prozedur ein. Anschließend starten Sie diese vom Testfenster aus, indem Sie den Prozedurnamen eingeben und am Ende der Zeile die -Taste drücken. Alternativ starten Sie die Prozedur über das *Weiter*-Symbol in der Symbolleiste.



Hinweis

Neben der vorgestellten Art und Weise zur Deklaration einer Variablen bietet die *Dim*-Anweisung noch eine weitere Variante mithilfe von Datentypkennzeichen. Hierbei wird dem Namen der Variablen ein Typkennzeichen angehängt, das Access den Datentyp mitteilt. Für Zeichenketten ist hierbei das Typkennzeichen *\$* zuständig. Die Deklaration einer Zeichenkettenvariablen kann also mithilfe der folgenden beiden Anweisungen erfolgen, die vom Ergebnis identisch sind:

```
Dim Text As String  
Dim Text$
```

Welche der beiden Varianten Sie vorziehen, bleibt Ihnen überlassen. Die Funktion der Variablen beeinflussen Sie damit nicht. In vielen Fällen wird die zweite Variante weniger gerne benutzt, da sie die Lesbarkeit des Programmcodes verschlechtert. Außerdem sollte bei der weiteren Verwendung der Variablen innerhalb der Prozedur stets das Typkennzeichen mit angegeben werden. Bei Verwendung von Typkennzeichen hat die vorhergehende Beispielprozedur den folgenden Aufbau:





```
Sub Testprozedur6()  
  
Dim Text$  
  
Text$ = InputBox("Bitte geben Sie einen Text ein!")  
  
MsgBox "Ihre Eingabe: " & Text$  
  
End Sub
```

Implizite und explizite Deklaration

Die Deklaration über die *Dim*-Anweisung – ob mit oder ohne Typkennzeichen – wird auch als *explizite Deklaration* bezeichnet, da Sie vor der Benutzung der Variablen sowohl den Namen der Variablen als auch den Datentyp festlegen.

Darüber hinaus unterstützt Access auch die *implizite Deklaration*, die aus Rücksicht auf ältere Basic-Dialekte vorhanden ist. Hierbei können Sie eine Variable direkt und ohne Anmeldung verwenden. Die Testprozedur könnte also folgendermaßen aussehen:

```
Sub Testprozedur7()  
  
Text$ = InputBox("Bitte geben Sie einen Text ein!")  
  
MsgBox "Ihre Eingabe: " & Text$  
  
End Sub
```

In diesem Fall wird die *Dim*-Anweisung einfach ausgelassen. Wenn Access nun bei der Abarbeitung der Prozedur auf eine unbekannte Variable trifft – in diesem Fall gleich in der ersten Zeile – deklariert es die Variable automatisch. Den Datentyp ermittelt Access hierbei anhand des Typkennzeichens.

Das Problem bei der impliziten Deklaration ist, dass Access einige Fehler nicht selbstständig ausfindig machen kann. Wenn Ihnen beispielsweise in der zweiten Anweisung ein Tippfehler unterlaufen wäre und Sie anstelle des Variablennamens *Text\$* den Namen *Test\$* angegeben hätten, würde Access von zwei getrennten Variablen ausgehen. Wenn Sie diese fehlerhafte Prozedur ausführen, liest Access die Eingabe des Benutzers in die Variable *Text\$* ein. In der zweiten Anweisung wird dann allerdings der Inhalt der Variablen *Test\$* ausgegeben, der in diesem Fall eine leere Zeichenkette ist. In dem Meldungsfenster würde also lediglich der Text "Ihre Eingabe: " erscheinen.

Da solche Schreibfehler bei langen Prozeduren sehr schlecht ausfindig zu machen sind, ist die Möglichkeit zur impliziten Deklaration üblicherweise deaktiviert und damit nicht zulässig. Jedes Modul hat zu diesem Zweck im Deklarationsteil die Anweisung *Option Explicit*, die die explizite Deklaration erzwingt. Die Anweisung wirkt sich nur auf das aktuelle Modul aus. Sie können also für jedes Modul getrennt bestimmen, ob Sie die implizite Deklaration erlauben.

Die *Option Explicit*-Anweisung



Bei Modulen, die Sie neu anlegen, fügt Access selbstständig die *Option Explicit*-Anweisung ein. Wenn Sie dies unterbinden möchten, löschen Sie die Option *Variablendeklaration erforderlich* des *Optionen*-Dialogs. Sie finden diese Option im Dialogfenster *Extras > Optionen* im Register *Editor*.

Nummerische Datentypen

Im vorhergehenden Abschnitt haben Sie den grundlegenden Umgang mit Variablen anhand des Datentyps *String* kennen gelernt. Variablen dieses Typs haben allerdings den Nachteil, dass sie keine Zahlenwerte speichern können.

Zum Speichern von Zahlen stellt Access gleich mehrere Datentypen zur Verfügung. Die folgende Tabelle enthält eine Auflistung aller numerischen Datentypen:

Nummerische Datentypen

Datentyp	Typkennzeichen	Beschreibung
Byte	keines	Ganze Zahlen im Bereich von 0 bis 255 (1 Byte Speicherbedarf).
Integer	%	Ganze Zahlen im Bereich von -32.768 bis 32.767 (2 Byte Speicherbedarf).
Long	&	Ganze Zahlen im Bereich von -2.147.483.648 bis 2.147.483.647 (4 Byte Speicherbedarf)
Single	!	Dezimalbrüche im Bereich von -3,402823E38 bis -1,401298E-45 für negative Werte und 1,401298E-45 bis 3,402823E38 für positive Werte (4 Byte Speicherbedarf).
Double	#	Dezimalbrüche im Bereich von -1,79769313486232E308 bis -4,94065645841247E-324 für negative Werte und 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte (8 Byte Speicherbedarf).
Currency	@	Festkommazahlen im Bereich von -922.337.203.685.477,5808 bis 922.337.203.685.477,5807 (8 Byte Speicherbedarf).

Tabelle 3.32





Die Datentypen unterscheiden sich einmal in der oberen und unteren Grenze und in der Genauigkeit nach dem Komma. Dies wirkt sich unter anderem auf den Speicherbedarf aus. Im Gegensatz zu einer *Integer*-Variablen benötigt eine *Double*-Variable gleich den vierfachen Speicherplatz – nämlich 8 Byte. Neben dem höheren Speicherbedarf macht sich dies auch in der Geschwindigkeit von Berechnungen bemerkbar, die z.B. mit *Integer*-Variablen schneller als mit *Double*-Variablen durchgeführt werden können.

Wertebereich und Genauigkeit

Ein weiterer Unterschied besteht in dem Wertebereich und der Genauigkeit, mit der Zahlen gespeichert werden können. Eine *Integer*-Variable kann beispielsweise nur ganze Zahlen im Bereich von -32.768 bis 32.767 speichern. Wenn Sie der Variablen einen Wert außerhalb dieses Bereichs zuweisen, tritt ein Fehler auf. Eine *Double*-Variable besitzt dagegen einen wesentlich größeren Zahlenumfang und bietet zudem noch die Möglichkeit zum Speichern von Dezimalbrüchen.

Um sowohl die Ausführungsgeschwindigkeit als auch den Speicherbedarf zu optimieren, sollten Sie in der Praxis immer diejenigen Datentypen auswählen, welche die geringsten Anforderungen stellen und gleichzeitig noch Ihren Bedürfnissen genügen. In der obigen Tabelle sind die einzelnen Datentypen nach den Anforderungen an Ihren Rechner sortiert. Der oberste Datentyp *Byte* hat also die geringsten Anforderungen, bietet dafür allerdings nur eingeschränkte Genauigkeit und einen stark begrenzten Wertebereich.

Der Datentyp *Currency* hat eine besondere Bedeutung, da er zum Speichern von Währungen optimiert ist. Der Vorteil dieses Datentyps im Gegensatz zum *Double*-Datentyp besteht darin, dass Berechnungen mit besonders hoher Genauigkeit ausgeführt werden.

Wenn numerische Variablen untereinander verwendet werden, führt Access in vielen Fällen eine automatische Anpassung des Wertebereichs durch. Es ist beispielsweise zulässig, einer Variablen des Datentyps *Byte* einen Wert zuzuweisen, der in einer Variablen des Typs *Integer* gespeichert wird. Voraussetzung dafür ist, dass der Inhalt der *Integer*-Variablen im Wertebereich des Datentyps *Byte* – also zwischen 0 und 255 – liegt. Das folgende Beispiel führt dementsprechend nicht zu einem Fehler:

```
Dim IntegerZahl As Integer
Dim ByteZahl As Byte
```

```
IntegerZahl = 100
ByteZahl = IntegerZahl
```

Dagegen führt das folgende Beispiel zu dem Fehler *Überlauf*, weil der Wertebereich des Datentyps *Byte* überschritten wird:

```
Dim IntegerZahl As Integer
Dim ByteZahl As Byte
```



```
IntegerZahl = 300  
ByteZahl = IntegerZahl
```

Nicht so streng ist Access bei der Zuweisung von Dezimalbrüchen an einen ganzzahligen Datentyp (z.B. *Integer* oder *Long*). In diesem Fall nimmt Access automatisch eine Rundung vor und weist das Ergebnis der Variablen zu. Die folgende Prozedur zeigt dies:

```
Sub Testprozedur8()  
  
Dim EineZahl As Integer  
  
    EineZahl = 3.85  
    Debug.Print EineZahl  
  
End Sub
```

In der Prozedur wird zu Beginn eine Variable mit dem Namen *EineZahl* mit dem Datentyp *Integer* deklariert. In der zweiten Zeile wird ihr der Wert 3.85 zugewiesen. Bei der Programmausführung bemerkt Access nun, dass die Variable die Zahl nicht vollständig speichern kann. Es wird daher eine Rundung vorgenommen und schließlich der Wert 4 gespeichert. Die Anweisung *Debug.Print* gibt das Ergebnis zur Kontrolle im Direktfenster aus.

Andersherum stellt es natürlich kein Problem dar, beispielsweise einer *Double*-Variablen eine ganze Zahl wie 4 zuzuweisen.

Nach der Deklaration werden numerische Variablen automatisch mit dem Wert 0 versehen! Dieser Vorgang wird auch als *Initialisierung* bezeichnet. Der Wert 0 ist in diesem Fall der *Initialisierungswert*.



Hinweis

Der Datentyp *Boolean*

Der Datentyp *Boolean* ermöglicht es, die beiden Zustände *True* und *False* zu speichern. Einer Variablen, die Sie mit diesem Datentyp deklariert haben, können Sie also alle logischen Ausdrücke zuweisen. Das folgende Beispiel deklariert eine Variable mit dem Datentyp *Boolean*:

```
Dim Antwort As Boolean
```

Der Speicherbedarf beträgt 2 Byte. Direkt nach der Deklaration einer *Boolean*-Variablen hat diese den Initialisierungswert *False*.

Der Datentyp *Boolean* ist mit allen numerischen Datentypen kompatibel. Wenn Sie beispielsweise einen *Boolean*-Ausdruck einer Variablen des Datentyps *Integer* zuweisen, wandelt Access *True* in -1 und *False* in 0 um. Gleiches gilt auch für alle anderen numerischen Datentypen. Eine Ausnahme stellt jedoch der Datentyp *Byte* dar. Hier wandelt Access *True* in den Wert 255 um. *False* wird dagegen wieder in den Wert 0 umgewandelt.





Der umgekehrte Weg, nämlich die Umwandlung eines numerischen Datentyps in den Datentyp *Boolean*, ist ebenfalls möglich. Access interpretiert hierbei den Wert 0 als *False* und alle anderen Werte als *True*.

Der Datentyp *Date*

Speziell auf das Speichern von Datums- und Zeitwerten ist der Datentyp *Date* zugeschnitten. Da Datums- und Zeitausdrücke intern im seriellen Datumsformat abgelegt werden, lassen sich prinzipiell auch *Double*-Variablen zum Speichern von Datumswerten verwenden. Hierbei geht allerdings der Datumscharakter verloren, wie das folgende Beispiel zeigt:

```
Sub DatumDouble()  
  
Dim i As Double  
  
    i = #1/1/96#  
  
    Debug.Print i  
  
End Sub
```

Als Ergebnis dieser Prozedur wird im Testfenster der Zahlenwert 35065 ausgegeben. Wenn Sie dagegen die Variable *i* mit dem Datentyp *Date* deklarieren, ist das Ergebnis 01.01.96:

```
Sub DatumDate()  
  
Dim i As Date  
  
    i = #1/1/96#  
  
    Debug.Print i  
  
End Sub
```

Der Datentyp *Variant*

Ein besonders flexibler Datentyp ist der *Variant*-Datentyp. Wie der Name schon vermuten lässt, kann er numerische Werte, Zeichenketten, logische Ausdrücke und Datumswerte speichern. Der *Variant*-Datentyp ist damit ein Universaldatentyp für alle Einsatzbereiche. Der Nachteil besteht darin, dass eine *Variant*-Variable einen Speicherbedarf von mindestens 16 Byte besitzt und bei mathematischen Berechnungen oder Zeichenkettenoperationen etwas langsamer ist. Eine Berechnung dauert beispielsweise mit *Variant*-Variablen etwa doppelt so lange wie mit *Integer*-Variablen. Da in der Praxis oft nur einzelne Berechnungen durchgeführt werden, tritt der Zeitunterschied allerdings kaum zutage.



Um eine Variable mit dem Datentyp *Variant* zu deklarieren, gehen Sie wie gewohnt vor:

Deklaration

```
Dim EineVariable As Variant
```

Im Unterschied zu den anderen Datentypen besitzt der *Variant*-Datentyp kein Typkennzeichen. Wenn Sie also die Anweisung

```
Dim EineVariable
```

angeben, dekariert Access diese automatisch mit dem Datentyp *Variant*. Dies ist auch bei der impliziten Deklaration der Fall. Findet Access eine noch nicht bekannte Variable ohne Typkennzeichen, wird sie mit dem Datentyp *Variant* dekariert.

Wenn Sie eine *Variant*-Variable in numerischen Berechnungen oder in Zeichenkettenoperationen einsetzen, interpretiert Access den Inhalt automatisch in der gerade benötigten Form. Bei Bedarf werden beispielsweise Zeichenketten in numerische Werte umgewandelt. Der umgekehrte Weg ist ebenfalls möglich. Das folgende Beispiel kann daher ohne Probleme abgearbeitet werden:

```
Sub Testprozedur13()  
  
Dim a As Variant  
Dim b As Variant  
Dim Ergebnis As Variant  
  
a = 5  
b = "7"  
Ergebnis = a * b  
  
MsgBox Ergebnis  
  
End Sub
```

Der Variablen *a* wird ein numerischer Wert, der Variablen *b* eine Zeichenkette zugewiesen. Anschließend erfolgt eine Multiplikation, bei der Access die Zeichenkette 7 in einen numerischen Wert umwandeln muss. Das Ergebnis wird in der gleichnamigen Variablen gespeichert. Bei dem Aufruf der *MsgBox*-Anweisung muss das Ergebnis nun wieder in eine Zeichenkette umgewandelt werden, da diese Anweisung nur Zeichenketten akzeptiert. Alle diese Konvertierungen werden von Access im Hintergrund und automatisch erledigt, so dass Sie sich hierum nicht kümmern müssen.

Bei jeder Zuweisung eines Inhalts an die *Variant*-Variable protokolliert Access unsichtbar den aktuellen Datentyp mit. Mithilfe der *VarType*-Funktion kann dieser Datentyp ermittelt werden:

Inhalt von *Variant*-Variablen

```
VarType(<Variant-Variable>)
```



Als Parameter muss lediglich die *Variant*-Variable (genauer: der Name der Variablen) angegeben werden, deren Datentyp Sie untersuchen möchten. Die Funktion gibt einen der folgenden Werte zurück:

Ergebnisse der *VarType*-Funktion

Konstante	Wert	Datentyp > Bedeutung
<i>vbEmpty</i>	0	Leer. Der Variablen wurde noch kein Wert zugewiesen (siehe unten).
<i>vbNULL</i>	1	<i>NULL</i> . Die Variable besitzt keinen Inhalt (siehe unten).
<i>vbInteger</i>	2	Integer
<i>vbLong</i>	3	Long
<i>vbSingle</i>	4	Single
<i>vbDouble</i>	5	Double
<i>vbCurrency</i>	6	Currency
<i>vbDate</i>	7	Date
<i>vbString</i>	8	String (Zeichenkette)
<i>vbBoolean</i>	11	Boolescher Wert
<i>vbDecimal</i>	14	Potenzzahlen zur Basis 10
<i>vbByte</i>	17	Byte

Tabelle 3.33

Zu jedem Wert stellt Access eine passende Konstante zur Verfügung, die Sie anstelle des Wertes einsetzen können.

Die folgende Prozedur weist einer *Variant*-Variablen einige Inhalte zu und gibt anschließend das Ergebnis der *VarType*-Funktion im Testfenster aus:

```
Sub Testprozedur14()

Dim EineVariable As Variant

    "VarType vbEmpty
    Debug.Print VarType(EineVariable)

    "VarType vbString
    EineVariable = "Dies ist ein Text"
    Debug.Print VarType(EineVariable)
```





```
"VarType vbInteger
EineVariable = 10
Debug.Print VarType(EineVariable)

"VarType vbDouble
EineVariable = 22.43
Debug.Print VarType(EineVariable)

"VarType vbDate
EineVariable = #01.01.96#
Debug.Print VarType(EineVariable)
```

End Sub

Die Ergebnisse sind jeweils als Kommentare im Programmcode der Prozedur vermerkt.

Eine Besonderheit stellen die beiden Inhalte *Leer* und *NULL* dar, die ausschließlich von *Variant*-Variablen unterstützt werden. Der Inhalt *Leer* (Varianttyp 0) ist der Initialisierungswert von *Variant*-Variablen. Mit diesem Inhalt werden *Variant*-Variablen direkt nach der Deklaration gefüllt. Eine weitere Besonderheit stellt der Inhalt *NULL* dar, der nicht mit dem Zahlenwert 0 verwechselt werden darf. Dieser Inhalt bedeutet, dass die *Variant*-Variable keinen Inhalt besitzt. Im Gegensatz zu dem Inhalt *Leer* können Sie einer *Variant*-Variablen an einer beliebigen Stelle innerhalb einer Prozedur den Inhalt *NULL* zuweisen:

```
<Variablenname> = NULL
```

Um zu überprüfen, ob eine *Variant*-Variable den Inhalt *NULL* oder *Leer* besitzt, ist neben der *VarType*-Funktion auch der Einsatz der beiden Funktionen *IsEmpty* und *IsNull* möglich.

```
IsNull (<Variablenname>)
IsEmpty (<Variablenname>)
```

Die beiden Funktionen haben entweder *True* oder *False* zum Ergebnis. Im *True*-Fall ist das Ergebnis *NULL* bzw. *Leer*.

Variant-Variablen werden häufig dazu verwendet, um den Inhalt eines Steuerelements oder eines Tabellenfeldes auszulesen, da hierbei nicht auf den jeweiligen Felddatentyp Rücksicht genommen werden muss.

Besonders wichtig ist außerdem, dass Felder und Steuerelemente, in die keine Eingabe vorgenommen wurde, stets den Inhalt *NULL* besitzen. Würden Sie versuchen, diesen Inhalt einer Variablen des Typs *String* oder *Integer* zuzuweisen, würde dies zu einem Fehler führen. Bei Variablen des Datentyps *Variant* kann dies nicht passieren. Das folgende Beispiel liest den Inhalt des Feldes *Name* aus dem geöffneten Formular *Adressen* aus und überprüft, ob es einen Inhalt hat:

Leer und NULL

Verwendung von Variant-Variablen





```
Sub Testprozedur15()  
  
Dim Inhalt As Variant  
  
    Inhalt = Forms!Adressen!Name  
  
    If IsNULL(Inhalt) Then  
  
        MsgBox "Das Feld hat keinen Inhalt."  
  
    Else  
  
        MsgBox "Das Feld hat einen Inhalt."  
  
    End If  
  
End Sub
```

Hinweis



Informationen über die Bezugnahme auf Formulare und Steuerelemente erhalten Sie im *Kapitel 4 Programmieren mit Objekten*.

Typumwandlung

Einige Datentypen sind untereinander nicht kompatibel. Dazu gehören beispielsweise die numerischen Datentypen und der Datentyp *String*. Es ist beispielsweise nicht zulässig, einer numerischen Variablen direkt eine Zeichenkette zuzuweisen.

Um dieses Problem zu lösen, stellt Access spezielle Typumwandlungsfunktionen zur Verfügung. Die wichtigsten Funktionen sind hierbei *Str* und *Val*:

```
Str(<Number>)  
Val(<String>)
```

Str-Funktion



Die *Str*-Funktion wandelt einen beliebigen numerischen Ausdruck in eine Zeichenkette um. Diese können Sie dann beispielsweise in einer Variablen des Typs *String* speichern. Das folgende Beispiel verwendet die *Str*-Funktion, um den Wert einer Zahl in einem Meldungsfenster anzuzeigen:

```
Sub Testprozedur9()  
  
Dim EineZahl As Integer  
Dim EinText As String  
  
    EineZahl = 23  
  
    EinText = Str(EineZahl)  
  
    MsgBox EinText  
  
End Sub
```



Der umgekehrte Weg, nämlich die Umwandlung eines Textes in einen numerischen Wert, ist mithilfe der *Val*-Funktion möglich:

Val-Funktion

```
Sub Testprozedur10()  
  
Dim EinText As String  
Dim EineZahl As Integer  
  
    EinText = InputBox("Bitte geben Sie eine Zahl ein!")  
  
    EineZahl = Val(EinText)  
  
    Debug.Print EineZahl  
  
End Sub
```

Beachten Sie bei diesem Beispiel, dass die *InputBox*-Funktion immer eine Zeichenkette zum Ergebnis hat. Dies trifft auch dann zu, wenn Sie lediglich Ziffern eingeben. Erst mithilfe der *Val*-Funktion wird aus der Zeichenkette ein Zahlenwert.

Die Umwandlung einer Zeichenkette in eine Zahl ist nur dann möglich, wenn die Zeichenkette einen Inhalt hat, der sinnvoll als Zahl interpretiert werden kann. Die Funktion wertet die Zeichenkette von links nach rechts aus. Sobald sie hierbei auf einen Buchstaben oder ein Sonderzeichen trifft, wird die Konvertierung abgebrochen und nur der vorhergehende Teil der Zeichenkette berücksichtigt. Wenn die Funktion keine Konvertierung vornehmen kann, ist das Ergebnis 0.

Die folgende Tabelle enthält einige Beispiele zu den beiden Funktionen:

Beispiele zu den Funktionen *Val* und *Str*

Ausdruck	Ergebnis
<i>Val</i> ("40595 Düsseldorf")	40595
<i>Val</i> ("2000")	2000
<i>Val</i> ("Düsseldorf, 40595")	0
<i>Val</i> ("80A20")	80
<i>Val</i> (" 80A20")	80
<i>Val</i> ("20.433")	20,433
<i>Val</i> ("20,433")	20



Beispiele zu den Funktionen *Val* und *Str* (Forts.)

Ausdruck	Ergebnis
<i>Str</i> (20)	" 20"
<i>Str</i> (-20)	"-20"
<i>Trim</i> (<i>Str</i> (20))	"20"

Tabelle 3.34

In dem folgenden Beispiel werden die beiden Funktionen *Val* und *Str* zusammen verwendet. Der Benutzer erhält die Möglichkeit zur Eingabe zweier Zahlen. Diese werden miteinander multipliziert und schließlich in einem Meldungsfenster ausgegeben:

```
Sub Testprozedur11()  
  
Dim Text1 As String  
Dim Text2 As String  
Dim Zahl1 As Integer  
Dim Zahl2 As Integer  
Dim Ergebnis As String  
  
    Text1 = InputBox("Bitte geben Sie die erste Zahl _  
                    ein!")  
    Text2 = InputBox("Bitte geben Sie die zweite Zahl _  
                    ein!")  
  
    Zahl1 = Val(Text1)  
    Zahl2 = Val(Text2)  
  
    Ergebnis = Str(Zahl1 * Zahl2)  
  
    MsgBox "Das Ergebnis ist " & Ergebnis  
  
End Sub
```

 Hinweis

Sollten Sie einen Wert eingeben, der nicht im Wertebereich von *Integer*-Variablen liegt, tritt während der Ausführung der Prozedur der Fehler *Überlauf* auf. Sie können dieses Problem beheben, indem Sie andere Datentypen wählen oder ungültige Eingaben mittels einer *If...Then*-Anweisung abfangen.

Die folgende Variante ist nicht nur kürzer, sondern umgeht das Problem elegant, indem das Ergebnis nicht in numerischen Variablen gespeichert wird. Stattdessen wird die Zeichenkette erst zur Berechnung in eine Zahl umgewandelt. Ein weiterer Vorteil dieser Strategie ist, dass nun auch mit Dezimalbrüchen gerechnet werden kann.



```
Sub Testprozedur12()  
  
Dim Text1 As String  
Dim Text2 As String  
  
    Text1 = InputBox("Bitte geben Sie die erste Zahl _  
        ein!")  
    Text2 = InputBox("Bitte geben Sie die zweite Zahl _  
        ein!")  
  
    MsgBox "Das Ergebnis ist " & Str(Val(Text1) *  
        Val(Text2))  
  
End Sub
```

Die weitere Typumwandlungsfunktion mit dem Namen *CVDate* hat die Aufgabe, eine Zeichenkette in ein Datum des Datentyps *Date* umzuwandeln:

 *CVDate*-Funktion

CVDate(<Expression>)

Unter <Expression> können Sie eine Zeichenkette angeben, die ein beliebiges Datum oder eine beliebige Zeit enthält. Die Datumsangabe muss hierbei gemäß den Einstellungen der Windows-Systemsteuerung erfolgen. Wenn Sie die deutschen Ländereinstellungen ausgewählt haben, kann die Zeichenkette also ein Datum im deutschen Format (z. B. tt.mm.jj) enthalten. Die Verwendung deutscher Monatsnamen ist ebenfalls zulässig.

Wenn die Funktion die Zeichenkette nicht als Datum interpretieren kann, tritt ein Laufzeitfehler auf. Vor der Ausführung der Funktion sollten Sie daher überprüfen, ob eine Konvertierung überhaupt möglich ist. Dies bewerkstelligen Sie mithilfe der Funktion *IsDate*:

IsDate(<Expression>)

Diese Funktion gibt den logischen Wert *True* zurück, wenn die unter <Expression> angegebene Zeichenkette als Datum interpretiert werden kann. Die folgende Prozedur verwendet die Funktion, um eine Benutzereingabe zu überprüfen:

```
Sub DatumseingabeÜberprüfen()  
  
Dim Eingabe As String  
Dim EinDatum As Date  
  
    Eingabe = InputBox("Bitte geben Sie ein Datum ein!")  
  
    If Len(Eingabe) > 0 Then  
  
        If IsDate(Eingabe) Then  
  
            Beep
```



```
MsgBox "Die Eingabe ist in Ordnung!"
EinDatum = CDate(Eingabe)

Else

    Beep
    MsgBox "Sie haben kein gültiges Datum eingegeben!"

End If

End If

End Sub
```

Typumwandlungsfunktionen

Neben den drei im vorhergehenden Abschnitt vorgestellten Funktionen zur Typumwandlung stellt Access noch eine Reihe weiterer Typumwandlungsfunktionen zur Verfügung. Ein Beispiel hierfür ist die *CDBl*-Funktion. Mit dieser Funktion können Sie einen beliebigen Ausdruck in einen *Double*-Ausdruck umwandeln. Der Vorteil gegenüber der *Val*-Funktion besteht darin, dass der Zieldatentyp – in diesem Fall *Double* – genau festgelegt ist. Es kann also beispielsweise ein Ausdruck des Datentyps *Integer* in einen *Double*-Ausdruck umgewandelt werden, um eine höhere Genauigkeit bei Berechnungen zu erzielen. Die folgende Tabelle enthält alle wichtigen Typumwandlungsfunktionen:

Typumwandlungsfunktionen

Datentyp	Umwandlung in Datentyp
<i>CBool</i>	Boolean
<i>CByte</i>	Byte
<i>CCur</i>	Currency
<i>CDate</i>	Date
<i>CDBl</i>	Double
<i>CDec</i>	Decimal
<i>CLng</i>	Long
<i>CSng</i>	Single
<i>CStr</i>	String
<i>CVar</i>	Variant

Tabelle 3.35



Der Datentyp *Decimal* ist ein spezieller Datentyp für skalierte Ganzzahlen und Zahlen mit 28 Kommastellen und kann derzeit nur innerhalb des *Variant*-Datentyps verwendet werden.

Geltungsbereich und Lebensdauer von Variablen

Bis jetzt haben Sie nur Variablen kennen gelernt, die innerhalb einer Prozedur deklariert wurden. Solche Variablen können nur innerhalb dieser Prozedur verwendet werden und heißen daher auch *lokale Variablen*. Der allgemeine Geltungsbereich einer Variablen wird auch als *Sichtbarkeit* bezeichnet. Lokale Variablen sind nur innerhalb der jeweiligen Prozedur sichtbar. Andere Prozeduren können diese Variablen also nicht nutzen.

Auswirkungen hat dies auch auf die Namensvergabe von Variablen. Aufgrund der eingeschränkten Sichtbarkeit ist es beispielsweise ohne Probleme möglich, in zwei unterschiedlichen Prozeduren zwei Variablen mit den gleichen Namen zu deklarieren.

Ein weiteres Kriterium zur Einordnung von Variablen ist die Lebensdauer. Die Lebensdauer lokaler Variablen beginnt mit der Deklaration. Sobald eine Prozedur beendet ist, wird die Variable inklusive ihres Inhalts aus dem Speicher gelöscht, was dem Ende der Lebensdauer entspricht. Sobald die Prozedur ein zweites Mal gestartet wird, erfolgt eine neue Deklaration und damit auch eine neue Initialisierung. Die Variable hat damit wieder den Initialisierungsinhalt (z. B. 0, *NULL* oder eine leere Zeichenkette). Dies wird an dem folgenden Beispiel schnell deutlich:

```
Sub Testprozedur16()  
  
    Dim EinWert As Integer  
  
    EinWert = EinWert + 1  
    Debug.Print EinWert  
  
End Sub
```

Die Prozedur deklariert eine Variable mit dem Namen *EinWert* und erhöht den Inhalt dieser Variablen anschließend um den Wert 1. Da die *Dim*-Anweisung der Variablen bei der Initialisierung den Wert 0 zuweist, gibt die Prozedur den Wert 1 im Direktfenster aus. Wenn Sie die Prozedur ein weiteres Mal starten, erfolgt wieder eine Initialisierung mit dem Wert 0 und die Prozedur gibt erneut den Wert 1 aus.

Eine Möglichkeit, die Lebensdauer einer Variablen zu erhöhen, sind statische Variablen. Diese Variablen verlieren im Gegensatz zu "normalen" Variablen ihren Inhalt nicht automatisch beim Beenden der Prozedur. Wenn die Prozedur erneut gestartet wird, initialisiert Access die Variable mit dem Wert, den sie zuletzt besaß. Die Lebensdauer wird damit über die Lebensdauer der Prozedur ausgedehnt.

Lebensdauer

Statische Variablen





Um eine statische Variable zu deklarieren, verwenden Sie anstelle der *Dim*-Anweisung die Anweisung *Static*, die von der Verwendung mit der *Dim*-Anweisung übereinstimmt:

```
Static <Variablenname> As <Datentyp>
```

Das folgende Beispiel stimmt weitestgehend mit dem der vorhergehenden Beispielprozedur überein. Der einzige Unterschied besteht darin, dass die Variable *EinWert* nun als statische Variable deklariert wird, die ihren Inhalt auch nach Beendigung der Prozedur beibehält:

```
Sub Testprozedur17()  
  
Static EinWert As Integer  
  
    EinWert = EinWert + 1  
    Debug.Print EinWert  
  
End Sub
```

Wenn Sie diese Prozedur das erste Mal starten, gibt sie den Wert 1 aus. Bei jedem weiteren Start erhöht sich das Ergebnis nun aber jeweils um den Wert 1.

Hinweis



Access stellt eine Möglichkeit zur Verfügung, alle innerhalb einer Prozedur deklarierten Variablen automatisch als statische Variablen zu deklarieren. In diesem Fall muss der Anweisung *Sub* im Kopf der Prozedur das Schlüsselwort *Static* vorangestellt werden. Die folgende Prozedur hat daher die gleiche Funktion wie die vorangehende:

```
Static Sub Testprozedur17()  
  
Dim EinWert As Integer  
  
    EinWert = EinWert + 1  
    Debug.Print EinWert  
  
End Sub
```

Private und öffentliche Variablen

Mithilfe von statischen Variablen können Sie zwar die Lebensdauer, nicht aber die Sichtbarkeit vergrößern. Oft ist es aber notwendig, dass mehrere Prozeduren eines Moduls oder auch einer kompletten Datenbank eine Variable gemeinsam benutzen können. Zu diesem Zweck stellt Access globale Variablen zur Verfügung.

Globale Variablen werden grundsätzlich im Deklarationsbereich eines Moduls definiert. Ähnlich wie Prozeduren lassen sich Variablen hier entweder als private oder als öffentliche Variablen deklarieren.





Private Variablen haben einen Geltungsbereich, der auf das aktuelle Modul beschränkt ist. Bei der Deklaration einer privaten Variablen gehen Sie genau wie bei der Deklaration einer lokalen Variablen vor. Anstelle der *Dim*-Anweisung verwenden Sie jedoch die *Private*-Anweisung:



Private Variablen

```
Private <Variablenname> As <Datentyp>
```

Das folgende Beispiel deklariert eine private *String*-Variable mit dem Namen *EinText*:

```
Private EinText As String
```

Wichtig ist hierbei, dass Sie die Anweisung im Deklarationsbereich des Moduls angeben müssen, in dem Sie die Variable nutzen möchten. Sie können die neue Variable nun von allen Prozeduren innerhalb des aktuellen Moduls aus nutzen. Ein Beispiel hierfür sind die folgenden beiden Prozeduren:

```
Sub MeldungDefinieren()  
  
    EinText = InputBox("Bitte geben Sie  
    einen Text ein!")  
End Sub  
Sub MeldungAusgeben()  
  
    MsgBox EinText  
  
End Sub
```

In beiden Prozeduren wird die Variable *EinText* verwendet. Die erste Prozedur füllt – wenn sie ausgeführt wird – den Inhalt der Variablen *EinText* mit einem Text, den der Benutzer eingibt. Wird anschließend die zweite Prozedur gestartet, erscheint dieser Text in einem Meldungsfenster. Voraussetzung für das Funktionieren ist, dass sich die beiden Prozeduren zusammen mit der privaten Variablen in einem Modul befinden.

In einigen wenigen Fällen ist es erforderlich, dass Prozeduren in verschiedenen Modulen auf die gleiche Variable zugreifen können. Zu diesem Zweck lassen sich innerhalb eines Moduls öffentliche Variablen deklarieren, die anderen Modulen zugänglich sind. Die Deklaration einer öffentlichen Variable erfolgt wieder im Deklarationsbereich eines Moduls. Anstelle der *Private*-Anweisung muss allerdings die *Public*-Anweisung eingesetzt werden:



Öffentliche Variablen

```
Public <Variablenname> As <Datentyp>
```

Das folgende Beispiel deklariert die Variable *EinText* als öffentliche Variable:

```
Public EinText As String
```





Die beiden Prozeduren *MeldungDefinieren* und *MeldungAusgeben* würden nun auch funktionieren, wenn sie sich in unterschiedlichen Modulen befinden.

Hinweis



Die Lebensdauer von privaten und öffentlichen Variablen beginnt mit dem Öffnen und endet mit dem Schließen der Datenbank! Wenn Sie einer solchen Variablen also einmal einen Inhalt zugewiesen haben, geht dieser erst wieder beim Beenden der Datenbank verloren. Besonders bei der Fehlersuche ist es jedoch manchmal erforderlich, den Inhalt aller Variablen zu löschen. Zu diesem Zweck wählen Sie den Menüpunkt *Ausführen* > *Zurücksetzen* aus. Alle privaten und öffentlichen Variablen werden hierbei mit ihren *NULL*-Werten initialisiert.

Namenskonflikte und Lokalität

Die eingeschränkste Sichtbarkeit haben lokale Variablen, die innerhalb einer Prozedur definiert werden. Sie können nur innerhalb dieser Prozedur verwendet werden. Bei der Deklaration einer neuen Variablen kann es vorkommen, dass Sie dieser einen Namen geben, den bereits eine private Variable des Moduls oder eine öffentliche Variable hat. Access meldet in diesem Fall allerdings keinen Fehler. Wenn Sie von der Prozedur aus auf die Variable zugreifen, verwendet Access automatisch die Variable, die Sie innerhalb der Prozedur definiert haben. Dieses Prinzip wird auch als *Lokalität* bezeichnet. Lokale Variablen haben vor allen anderen Variablen Vorrang, wenn ein Namenskonflikt auftritt.

Konflikte können außerdem auftreten, wenn eine private und eine öffentliche Prozedur denselben Namen tragen. Hier löst Access den Konflikt auch auf die gleiche Weise wie bei Prozeduren: Erfolgt der Zugriff von dem Modul aus, in dem die private Variable definiert wurde, greift Access auf diese Variable zu. Alle anderen Module greifen dagegen auf die öffentliche Variable zu.

Ebenfalls ist es möglich, in zwei verschiedenen Modulen zwei öffentliche Variablen mit demselben Namen zu verwenden. Auch hier gilt das Lokalitätsprinzip. Wenn der Zugriff allerdings von einem dritten Modul aus erfolgt, kann Access ihn nicht eindeutig zuordnen. In diesem Fall muss der Variablenname zusammen mit dem Modulnamen angegeben werden:

```
[<Modulname>].<Variablenname>
```

Wenn beispielsweise in den beiden Modulen *Modul A* und *Modul B* zwei öffentliche Variablen mit dem Namen *EinText* deklariert werden, müssen Sie beim Zugriff von einem dritten Modul aus einen der folgenden Verweise verwenden:

```
[Modul A].EinText  
[Modul B].EinText
```

Die Angabe der eckigen Klammern ist hierbei nur dann erforderlich, wenn der Modulname Leer- oder Sonderzeichen enthält.





Das Sichtbarkeits- und Lokalitätsprinzip von privaten und öffentlichen Variablen ist mit dem von Prozeduren vollkommen identisch. Ausführlichere Informationen dazu entnehmen Sie dem Abschnitt *Geltungsbereiche von Prozeduren* dieses Kapitels.



Hinweis

In der Programmierpraxis sollten Sie vermeiden, lokalen, privaten und öffentlichen Variablen denselben Namen zu geben. Namenskonflikte werden so von vornherein vermieden.

Benutzerdefinierte Datentypen

Datentypen wie *Integer* oder *String* werden auch als *Grunddatentypen* bezeichnet, da sie von Access vordefiniert sind. Darüber hinaus ist es mit Access möglich, eigene Datentypen zu entwerfen. Hierbei geht es allerdings weniger darum, den Wertebereich auszudehnen oder eine noch höhere Genauigkeit zu erzielen, sondern um das Definieren von zusammengesetzten Datentypen, die auf den Grunddatentypen basieren. Das folgende Beispiel definiert einen solchen Datentyp:

```
Private Type PersonenTyp
    Name As String
    Alter As Integer
    Telefon As String
End Type
```

Der benutzerdefinierte Datentyp beginnt zunächst einmal mit dem Schlüsselwort *Type* und einem Namen – in diesem Fall *PersonenTyp*. Anschließend folgt eine Liste der einzelnen Elemente des Datentyps. In dem Beispiel werden die drei Elemente *Name*, *Alter* und *Telefon* mit den entsprechenden Datentypen aufgelistet. Die Definition endet mit dem Schlüsselwort *End Type*.

Zwischen den *Type*- und *End Type*-Anweisungen können beliebig viele Elemente angegeben werden. Die einzelnen Elemente können entweder mit den Grunddatentypen oder anderen benutzerdefinierten Datentypen deklariert werden. Der Name der einzelnen Elemente und der Datentypname unterliegen denselben Beschränkungen wie Variablen- oder Prozedurnamen.

Die Definition eines benutzerdefinierten Datentyps muss im Deklarationsbereich eines Moduls erfolgen. Über die beiden Schlüsselwörter *Public* oder *Private*, die vor der *Type*-Anweisung angegeben werden können, legen Sie den Geltungsbereich des neuen Datentyps fest. Wenn Sie weder *Private* noch *Public* angeben, definiert Access den neuen Datentyp automatisch als *Public*.

Mit dem neuen Datentyp *PersonenTyp* kann jetzt eine Variable – beispielsweise mithilfe der *Dim*-Anweisung – deklariert werden:

```
Dim Person As PersonenTyp
```





Um auf ein Element der Variablen zugreifen zu können, muss der Variablenname, gefolgt von einem Punkt und dem Elementnamen angegeben werden. In dem folgenden Beispiel werden den Elementen der Variablen *Person* einige Daten zugewiesen:

```
Sub Testprozedur18()  
  
Dim Person As PersonenTyp  
  
    Person.Alter = 5  
    Person.Name = "Alfred Müller"  
    Person.Telefon = "0013224/245/3244-0"  
  
End Sub
```

Arrays

Ein Array ist ein Datenfeld, das aus einer Reihe von tabellarisch angeordneten Elementen besteht. Arrays werden immer dann eingesetzt, wenn eine große Anzahl von Daten mit gleichen Datentypen gespeichert werden soll. In der Access-Programmierpraxis werden – im Gegensatz zu anderen Basic-Dialekten – Arrays nur relativ selten eingesetzt. Die Funktion von Arrays lässt sich nämlich in vielen Fällen wesentlich leichter mithilfe von Datentabellen nachbilden.

Trotzdem besitzen Arrays ihre Daseinsberechtigung. Wenn Sie beispielsweise eine Reihe von Zahlen kurzzeitig speichern möchten, ermöglichen Arrays eine höhere Abarbeitungsgeschwindigkeit, als dies bei Tabellen möglich wäre. Außerdem haben Tabellen den Nachteil, dass nach umfangreichen Löschvorgängen eine Reorganisation des Datenbestands notwendig ist. Arrays werden dagegen im Hauptspeicher verwaltet und haben diesen Nachteil nicht.

Aufbau eines Arrays

Ein Array besteht aus einer Folge von Elementen, die stets den gleichen Datentyp besitzen. Jedes einzelne Element ist hierbei mit einer Variablen vergleichbar und kann einen Inhalt aufnehmen (z. B. eine Zahl oder einen Text). Um den Zugriff auf die einzelnen Elemente zu ermöglichen, sind diese nummeriert. Die Nummer eines Elements wird auch als *Index* bezeichnet.

Abb. 3.19

Aufbau eines
eindimensionalen
Arrays



Bei der Deklaration eines Arrays müssen Sie festlegen, wie viele Elemente das Array besitzen soll. Wenn Sie beispielsweise sieben Zahlen speichern möchten, benötigen Sie sieben Elemente. Weiterhin bestimmen Sie, mit welchem Index das erste Element angesprochen werden kann. In der Regel wird hierzu der Index 0 oder 1 ausgewählt. Das folgende Beispiel deklariert die Variable *TestArray* als Array mit dem Indexbereich von 1 bis 100:

```
Dim TestArray (1 To 100) As Integer
```

Die Variable *TestArray* kann nun 100 Werte vom Datentyp *Integer* speichern, wobei das erste Element den Index 1 besitzt. Anstelle des Datentyps *Integer* hätte auch jeder andere Grunddatentyp oder ein benutzerdefinierter Datentyp angegeben werden können. Letzteres ist besonders hilfreich, wenn Sie pro Element mehrere Daten wie z. B. einen Personennamen und eine Telefonnummer speichern möchten.

Um auf ein Element des Arrays zuzugreifen, muss dem Variablennamen ein numerischer Index folgen, der in runden Klammern angegeben wird. Der Index kann ein beliebiger ganzzahliger numerischer Ausdruck sein, also auch eine Berechnung, die auf anderen Variableninhalten basiert. In dem folgenden Beispiel werden die ersten drei Elemente mit Zahlenwerten gefüllt:

```
TestArray(1) = 25  
TestArray(2) = 10  
TestArray(2 + 1) = TestArray(2) + 1
```

Access lässt es zu, dass bei der Deklaration des Arrays die Untergrenze und das Schlüsselwort *To* des Indexbereichs ausgelassen werden. In diesem Fall verwendet Access die Standarduntergrenze 0. Das folgende Array hat also dementsprechend 101 Elemente (1 Element für den Index 0 und 100 weitere Elemente für die Indexbereiche 1 bis 100):

```
Dim TestArray (100) As Integer
```

Die Standarduntergrenze von Datenfeldern kann mithilfe der *Option Base*-Anweisung von 0 auf 1 geändert werden. Dazu geben Sie im Deklarationsbereich eines Moduls die folgende Anweisung an:

```
Option Base 1
```

Die Anweisung wirkt sich nur auf das aktuelle Modul aus.

Anstelle der Zahlenangaben können die Indexgrenzen des Arrays auch mithilfe von Konstanten festgelegt werden:

```
Const MaxAnzahl = 200  
Dim TextArray (1 To MaxAnzahl)
```



Hinweis





Deklaration von Arrays

Die Deklaration eines Arrays kann sowohl auf Prozedurebene – mithilfe der *Dim*-Anweisung – als auch im Deklarationsteil eines Moduls mithilfe der *Private*- und *Public*-Anweisung erfolgen. Hierdurch legen Sie den Geltungsbereich genau wie bei normalen Variablen fest.

Die Deklaration statischer Arrays ist auf Prozedurebene ebenfalls möglich. Anstelle der *Dim*-Anweisung muss hierzu die *Static*-Anweisung verwendet werden:

```
Static TestArray (1 To 100) As Integer
```

Mehrdimensionale Arrays

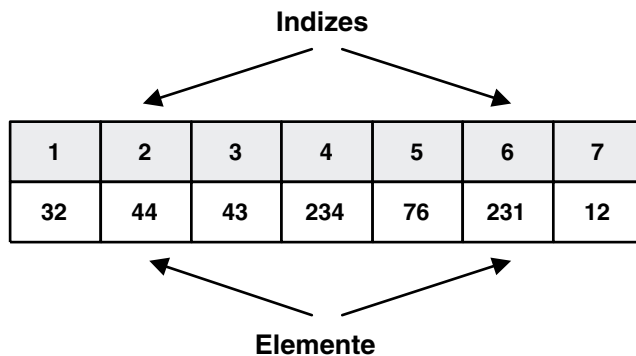
Neben eindimensionalen Arrays erlaubt VBA auch die Deklaration mehrdimensionaler Arrays. Ein mehrdimensionales Array besteht nicht nur aus einer eindimensionalen Liste, sondern aus einer mehrdimensionalen Matrix. Die folgende Anweisung deklariert ein zweidimensionales Array mit 100 Zeilen und 100 Spalten:

```
Dim TestArray (1 To 100, 1 To 100) As Integer
```

Dieses zweidimensionale Array kann nun insgesamt 10.000 Werte vom Datentyp *Integer* aufnehmen. Grafisch lässt sich dieses Datenfeld als zweidimensionale Matrix darstellen, wie die folgende Abbildung zeigt.

Abb. 3.20

Aufbau eines
zweidimensionalen
Arrays



Der Zugriff erfolgt auf ähnliche Weise wie bei eindimensionalen Arrays. Ein einzelnes Element hat nun jedoch nicht nur einen einzelnen Index, sondern zwei oder mehrere Indizes. In dem obigen Beispiel muss ein Wert für die Spalte und ein weiterer Wert für die Zeile angegeben werden. Die einzelnen Indizes müssen hierbei durch Kommas voneinander getrennt werden. Die folgende Anweisung weist dem Element am Kreuzungspunkt der vierten Spalte und der zweiten Zeile den Wert 5 zu:

```
TestArray (4,3) = 5
```

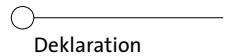
In VBA lassen sich Arrays mit maximal 60 Dimensionen deklarieren.



Dynamische Arrays

Bei allen bisherigen Arrays muss die Anzahl der benötigten Elemente bei der Erstellung des Programms bekannt sein. Da die Angaben im Programmcode vorgenommen werden, lassen sich die Dimensionsgrenzen des Arrays während der Programmausführung nicht ändern. Oftmals ist die benötigte Größe des Arrays bei der Erstellung des Programms nicht bekannt. Um dieses Problem zu lösen, stellt Access dynamische Arrays zur Verfügung, deren Größe sich während der Programmausführung ändern lässt.

Die Deklaration eines dynamischen Arrays kann wie bei statischen Arrays auf Prozedurebene über die *Dim*-Anweisung oder im Deklarationsteil eines Moduls über die Anweisungen *Private* und *Public* erfolgen. Bei der Deklaration dürfen jedoch keine Indexbereiche angegeben werden:



Deklaration

```
Dim Datenfeld() As Integer
```

Bevor das Array nun verwendet werden kann, muss die Anzahl der Dimensionen und deren Indexbereiche ein erstes Mal innerhalb einer Prozedur festgelegt werden. Hierfür ist die Anweisung *ReDim* zuständig, die vom Aufbau der *Dim*-Anweisung ähnelt. In einer Prozedur kann nun beispielsweise die folgende Anweisung angegeben werden:

```
ReDim Datenfeld (1 To 5, 1 To 100)
```

Die Dimensionen und Indexgrenzen für das globale Array *Datenfeld* sind damit festgelegt, und auf die einzelnen Elemente des Arrays kann zugegriffen werden. Um die Indexbereiche des Arrays nachträglich zu ändern, muss die *ReDim*-Anweisung erneut ausgeführt werden:

```
ReDim Datenfeld (1 To 5, 1 To 50)
```

Die Indexgrenzen können im Gegensatz zur *Dim*-Anweisung bei der *ReDim*-Anweisung auch über Variablen festgelegt werden. So kann die Größe an die aktuellen Bedürfnisse angepasst werden:



Hinweis

```
Sub Testprozedur20()
```

```
Dim x As Integer, y As Integer
```

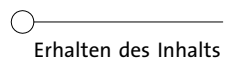
```
    x = 5
```

```
    y = 1
```

```
    ReDim Datenfeld(1 To x, y To 50)
```

```
End Sub
```

Die Indexgrenzen lassen sich beliebig vergrößern oder verkleinern. Auch die Anzahl der Dimensionen kann nachträglich verändert werden. Zu beachten ist allerdings, dass bei einer Veränderung des Datenfeldes die Inhalte aller Elemente des Arrays verloren gehen! Dies kann teilweise verhindert werden, indem zusätzlich das Schlüsselwort *Preserve* angegeben wird:



Erhalten des Inhalts



ReDim Preserve Datenfeld (1 To 10, 1 To 100)

Beim Einsatz des *Preserve*-Schlüsselworts müssen folgende Regeln beachtet werden:

- Die Anzahl der Dimensionen des Arrays darf nicht verändert werden.
- Änderungen dürfen nur an der letzten Dimension vorgenommen werden.

Wird das Array vergrößert, bleiben die Inhalte aller Elemente erhalten. Die neu hinzugekommenen Elemente werden mit dem für den Datentyp vorgesehenen Initialisierungswert gefüllt (z.B. 0, *NULL* oder eine leere Zeichenkette). Wird das Array dagegen verkleinert, fallen einige Elemente zusammen mit ihren Inhalten weg. Die Inhalte der übrigen Elemente bleiben dagegen erhalten.

Tipp

Möchten Sie die Inhalte aller Elemente des Arrays löschen und dadurch den belegten Speicherplatz freigeben, verwenden Sie die *Erase*-Anweisung:

Erase <Array>, <Array>, ...

Diese Funktion erwartet als Parameter ein oder mehrere dynamische Arrays. Wenn Sie das Array anschließend wieder verwenden möchten, muss vorher eine neue Dimensionierung mithilfe der *ReDim*-Anweisung erfolgen.

Die *Erase*-Anweisung kann auch bei nicht dynamischen Arrays eingesetzt werden. In diesem Fall werden die Inhalte aller Elemente mit den Initialisierungswerten versehen. Die Struktur des Arrays bleibt aber erhalten, und dementsprechend wird der belegte Speicherplatz auch nicht freigegeben.

LBound und *UBound*

Wenn Sie mit dynamischen Arrays arbeiten, ist es oft zweckmäßig, die obere und untere Grenze zu kennen. Zu diesem Zweck stellt Access die zwei Funktionen *LBound* und *UBound* (Abkürzungen für *Lower Bound* und *Upper Bound*) zur Verfügung. Beide Funktionen erwarten als Parameter den Namen des dynamischen Arrays. Die folgenden beiden Anweisungen ermitteln beispielsweise die Ober- und Untergrenze des Arrays *TestArray*:

```
Debug.Print LBound (TestArray)
Debug.Print UBound (TestArray)
```

Bei mehrdimensionalen Arrays müssen Sie zusätzlich die Dimension angeben, deren Grenze Sie ermitteln möchten. Die erste Dimension besitzt hierbei den Wert 1:

```
Debug.Print UBound (Datenfeld, 1)
Debug.Print UBound (Datenfeld, 2)
```

Hinweis

Wenn Sie bei mehrdimensionalen Arrays die Angabe der Dimension auslassen, ermittelt Access automatisch die Grenzen der ersten Dimension.



Variant-Arrays

Eine Alternative zu dynamischen Arrays stellen Variant-Arrays dar. Hierbei handelt es sich um Datenfelder, die innerhalb einer Variablen des Typs *Variant* abgelegt sind.

Um dieser Variablen ein Datenfeld zuzuweisen, muss die *Array*-Funktion verwendet werden. Das folgende Beispiel legt ein Variant-Array mit fünf Elementen an:

```
Sub VariantArray()  
  
Dim Var As Variant  
  
    Var = Array(1, 6, 2, 3, 1)  
  
End Sub
```

Die Anzahl der Elemente, die Sie der *Variant*-Variablen zuweisen, ist hierbei beliebig. Die Größe des Datenfeldes muss also nicht vorher angemeldet werden. Ein weiterer Vorteil gegenüber "normalen" Arrays besteht darin, dass die Datentypen der einzelnen Elemente unterschiedlich sein können, da die einzelnen Elemente stets den Datentyp *Variant* haben. Das folgende Beispiel zeigt dies:

```
Var = Array(1, "Dies ist ein Text", #1/1/95#, -3.322, 1)
```

Nicht zulässig sind lediglich benutzerdefinierte Datentypen sowie Zeichenketten, die eine feste Länge haben.

Der Zugriff auf ein bestimmtes Element erfolgt wie bei Arrays üblich über einen Index, der in Klammern hinter dem Namen der Variablen angegeben wird. Der Index des ersten Elements wird über die *Option Base*-Anweisung bestimmt. Wenn diese Anweisung nicht angegeben wird, weist Access dem ersten Element den Index 0 zu. Der folgende Ausdruck hat beispielsweise das dritte Element zum Ergebnis:

```
Var(2)
```

Die aktuelle Unter- und Obergrenze eines Variant-Arrays kann wie bei normalen Arrays über die Funktionen *UBound* und *LBound* ermittelt werden.

Um herauszufinden, ob in einer Variant-Variablen momentan ein Array gespeichert ist, können Sie die *IsArray*-Funktion verwenden:

```
IsArray(<VarName>)
```

Die Funktion gibt *True* zurück, wenn die unter <VarName> angegebene *Variant*-Variable ein Array enthält.



Array-Funktion





Konstanten

Eine Konstante ist im Prinzip nichts anderes als eine normale Variable, deren Wert während der Ausführung des Programms allerdings nicht verändert werden kann. Konstanten eignen sich daher besonders gut, wenn ein bestimmter fester Wert im Programm an mehreren Stellen auftaucht. Bei Änderungen braucht so nicht der komplette Programmcode überarbeitet, sondern nur der Wert der Konstanten geändert zu werden. Für die Definition einer Konstanten ist die Anweisung *Const* zuständig. Das folgende Beispiel definiert beispielsweise eine Konstante mit dem Namen *Pi*:

```
Const Pi = 3.14
```

In einer Prozedur kann nun anstelle des Wertes 3.14 die Konstante *Pi* angegeben werden:

```
Sub Testprozedur19()

Const Pi = 3.14
Dim Fläche As Single, Radius As Single

    Radius = 5
    Fläche = Radius ^ 2 * Pi

End Sub
```

Den Geltungsbereich von Konstanten legen Sie ähnlich wie bei Variablen mithilfe der Schlüsselwörter *Private* und *Public* fest. Eine der folgenden beiden Anweisungen kann beispielsweise im Deklarationsteil eines Moduls angegeben werden:

```
Private Const Pi = 3.14
Public Const Pi = 3.14
```

Namenskonflikte müssen genau wie bei Variablen oder Prozeduren durch Angabe des Modulnamens aufgelöst werden. Wenn Sie die *Const*-Anweisung im Deklarationsteil eines Moduls ohne *Private* oder *Public* angeben, wird die Konstante automatisch öffentlich deklariert.

Datentypen



Der Name der Konstanten unterliegt den gleichen Beschränkungen wie bei Variablennamen. Wie Sie vielleicht bemerkt haben, muss bei der Definition einer Konstanten kein Datentyp angegeben werden, da dieser von Access automatisch ermittelt wird. Dies ist auch dann der Fall, wenn – wie im folgenden Beispiel – eine Konstante als Zeichenkette oder Datum definiert wird:

```
Const Text = "Das ist ein Text"
Const Gründungstag = #1.10.90#
```



Neben einfachen Ausdrücken können auch berechnete Ausdrücke als Konstanten definiert werden. Die Berechnung darf hierbei auch auf anderen Konstanten basieren:

```
Const Stockwerke = 5
Const WohnungenProStockwerk = 23
Const Wohnungen = Stockwerke * WohnungenProStockwerk
```

Ist die Definition zweier öffentlicher Konstanten in verschiedenen Modulen gegenseitig voneinander abhängig, tritt beim Kompilieren des Programmcodes ein Fehler auf. Ansonsten gilt für Konstanten wie für Variablen das Lokalitätsprinzip.



Hinweis

Parametervariablen

Zum Austausch von Informationen zwischen mehreren Prozeduren haben Sie bis jetzt nur private bzw. öffentliche Variablen kennen gelernt, auf die mehrere Prozeduren zugreifen können. Die Arbeit mit solchen Variablen ist allerdings in der Regel nicht vorteilhaft. So kann beispielsweise der Inhalt einer öffentlichen Variablen von jeder Prozedur aus verändert werden. Um eine bessere Kontrolle über den Informationsaustausch zu erhalten, stellt Access eine Möglichkeit zur Verfügung, einer bestimmten Prozedur beim Aufruf Informationen zu übergeben. Die Informationen werden hierbei in Form von Parametern (auch als *Argumente* bezeichnet) ausgetauscht.

Die folgende Prozedur dient beispielsweise dazu, eine Meldung auf dem Bildschirm auszugeben. Sie haben eine solche Prozedur bereits in den vorhergehenden Abschnitten kennen gelernt. Die Besonderheit dieser Prozedur ist nun aber, dass die Meldung nicht fest vorgegeben ist, sondern beim Aufruf der Prozedur angegeben werden kann:

```
Sub MeldungAnzeigen2 (Meldungstext As String)
```

```
    Beep
    MsgBox Meldungstext
```

```
End Sub
```

Die Prozedur besitzt im Prozedurkopf die Deklaration einer Parametervariablen. Über diese Deklaration wird Access der Datentyp der Information mitgeteilt, welche die Prozedur beim Aufruf erwartet. In dem Beispiel wird also eine Zeichenkette erwartet. Zusätzlich zu dem Datentyp wird außerdem der Variablenname festgelegt, unter dem die Information innerhalb der Prozedur genutzt werden kann.

Um diese Prozedur vom Direktfenster zu starten, müssen Sie zusätzlich zum Prozedurnamen eine Zeichenkette mit dem gewünschten Meldungstext angeben:

```
MeldungAnzeigen2 "Dies ist eine Meldung!"
```





Alternativ können Sie zum Start auch die *Call*-Anweisung verwenden. In diesem Fall muss der Parameter allerdings eingeklammert werden:

```
Call MeldungAnzeigen2 ("Dies ist eine Meldung!")
```

Auch die Verwendung benannter Parameter ist wie bei den in Access eingebauten Anweisungen zulässig:

```
MeldungAnzeigen2 Meldungstext:="Dies ist eine Meldung!"  
Call MeldungAnzeigen2 (Meldungstext:="Dies ist eine  
Meldung!")
```

Wenn Sie die Prozedur von einer anderen Prozedur aus aufrufen möchten, können Sie zwischen den vier Alternativen wählen. Das folgende Beispiel verwendet die ersten beiden, um die Prozedur zweimal hintereinander mit verschiedenen Parametern aufzurufen:

```
Sub Testprozedur21()  
  
Dim Text As String  
  
    Text = InputBox("Bitte geben Sie einen Text ein!")  
  
    MeldungAnzeigen2 Text  
  
    Call MeldungAnzeigen2("Die Prozedur wird beendet!")  
  
End Sub
```

Eine Besonderheit der Prozedur ist, dass die Meldung vom Benutzer eingegeben und anschließend in der Variablen *Text* gespeichert wird. Diese Variable – bzw. der Inhalt dieser Variablen – wird dann im nächsten Schritt an die Prozedur *MeldungAnzeigen2* übergeben.

Bei der Übergabe eines Parameters an eine Prozedur ist stets darauf zu achten, dass jeweils die Datentypen übereinstimmen. Dies gilt nicht nur bei benutzerdefinierten Datentypen, sondern auch bei den Grunddatentypen wie *String* oder *Integer*. Einer Prozedur, die für einen bestimmten Parameter eine Zahl erwartet, kann also keine Zeichenkette übergeben werden. Um dieses Problem zu lösen, müssen Sie ggf. eine Typumwandlung durchführen. Die entsprechenden Funktionen (z. B. *Str* und *Val*) haben Sie bereits kennen gelernt.

Hinweis



Prozeduren, die Parametervariablen des Datentyps *Variant* besitzen, können beliebige Datentypen übergeben werden. Andersherum ist es ebenfalls möglich, den Inhalt einer *Variant*-Variablen an eine Prozedur zu übergeben, die einen anderen Datentyp (z. B. *Integer* oder *String*) erwartet.



Übergabe mehrerer Parameter

Häufig ist die Übergabe eines einzelnen Parameters an eine Prozedur nicht ausreichend. In diesem Fall können im Kopf der aufzurufenden Prozedur mehrere Parameter mit unterschiedlichen Datentypen definiert werden. Die einzelnen Deklarationen müssen hierbei in Form einer Liste angegeben werden, wobei die einzelnen Definitionen jeweils durch Kommas voneinander getrennt werden. Die folgende Beispielprozedur erwartet zwei *Integer*-Zahlen:

```
Sub SummeAusgeben (Zahl1 As Integer, Zahl2 As Integer)

    MsgBox Str(Zahl1 + Zahl2)

End Sub
```

Um diese Prozedur nun aufzurufen, müssen die Parameter in der richtigen Reihenfolge ebenfalls durch Kommas voneinander getrennt angegeben werden:

```
SummeAusgeben 5, 7
```

Bei der Verwendung benannter Parameter ist die Reihenfolge unerheblich. Wichtig ist nur, dass Sie alle Parameter angeben:

```
SummeAusgeben Zahl1:=5, Zahl2:=7
SummeAusgeben Zahl2:=7, Zahl1:=5
```

Eine weitere Möglichkeit, mehrere Informationen an eine Prozedur zu übergeben, ist die Verwendung von benutzerdefinierten Datentypen, die ebenfalls bei der Deklaration von Parametern verwendet werden dürfen.



Hinweis

Optionale Parameter

Beim Aufruf einer Prozedur mit Parametern ist es wichtig, dass alle Parameter, die von der Prozedur erwartet werden, auch wirklich angegeben sind. Eine Ausnahme stellen optionale Parameter dar, deren Angabe beim Aufruf der Prozedur freigestellt ist.

Um zu kennzeichnen, ob ein bestimmter Parameter ausgelassen werden darf, muss dem Parameter das Schlüsselwort *Optional* vorangestellt werden. Als Datentyp unterstützt Access hierbei jeden beliebigen Grunddatentyp.

Die folgende Prozedur hat drei Parameter, von denen der zweite und dritte optional ist:

```
Sub Testprozedur22(Wert1 As Variant, Optional Wert2 As
Variant, Optional Wert3 As Variant)
```





Der Aufruf kann nun wahlweise mit einem, mit zwei oder mit drei Parametern erfolgen:

```
Testprozedur 100  
Testprozedur 100, 200  
Testprozedur 100, 200, 50
```

Wenn ein bestimmter Parameter mit dem Attribut *Optional* versehen wurde, müssen alle folgenden Parameter ebenfalls optional sein, ansonsten kann Access die Parameter beim Aufruf nicht eindeutig zuordnen. Im Falle der Testprozedur bedeutet dies, dass der dritte Parameter mit dem Namen *Wert3* optional sein muss, da auch der zweite Parameter *Wert2* optional ist. Wird diese Regel nicht beachtet, meldet Access beim Kompilieren einen Fehler.

Da die Parameter *Wert2* und *Wert3* der Prozedur optional sind, ist es beispielsweise zulässig, beim Prozeduraufruf nur den ersten und dritten Parameter anzugeben. Damit Access die Parameter eindeutig zuordnen kann, darf der zweite Parameter beim Aufruf allerdings nicht einfach ausgelassen werden. Stattdessen muss für diesen Parameter ein Komma, aber kein Ausdruck angegeben werden. Das folgende Beispiel zeigt dies:

```
Testprozedur 100, , 50
```

Bei der Verwendung benannter Parameter ist dies nicht erforderlich. Hier genügt es, wenn Sie nur die gewünschten Parameter angeben:

```
Testprozedur Wert1:=100, Wert3:=50
```

IsMissing



Um festzustellen, ob ein bestimmter Parameter beim Aufruf der Prozedur angegeben wurde, kann die *IsMissing*-Funktion eingesetzt werden. Der folgende Ausdruck hat beispielsweise den Wert *True* zum Ergebnis, wenn der Parameter *Wert2* nicht angegeben wurde:

```
IsMissing(Wert2)
```

Die Funktion kann allerdings nur dann eingesetzt werden, wenn der optionale Parameter den Datentyp *Variant* hat. Bei optionalen Parametern anderer Datentypen hat die *IsMissing*-Funktion stets den Wert *False*.

Fehlende Parameter, die nicht den Datentyp *Variant* haben, initialisiert Access mit ihren jeweiligen *NULL*-Werten (z. B. 0 bei numerischen Variablen und "" bei Zeichenketten).

Parameter-Arrays

In einigen Fällen müssen einer Prozedur mehrere Parameter übergeben werden, deren Anzahl bei der Programmierung der Prozedur nicht festgelegt werden soll. In diesem Fall können Sie Parameter-Arrays einsetzen. Ein Parameter-Array ist im Grunde genommen nicht anderes als ein Variant-Array, das von Access beim Start



der Prozedur mit den übergebenen Parameterwerten gefüllt wird. Der folgenden *Sub*-Prozedur können beispielsweise mehrere Werte übergeben werden, deren Summe im Testfenster ausgegeben wird:

```
Sub SummeBerechnen(ParamArray Zahlen() As Variant)

Dim Summe As Variant
Dim i As Integer

    For i = LBound(Zahlen) To UBound(Zahlen)
        Summe = Summe + Zahlen(i)
    Next i

    Debug.Print Summe

End Sub
```

Innerhalb der Prozedur wird eine *For...Next*-Schleife verwendet, um die einzelnen Werte zu addieren. Die *For...Next*-Anweisung lernen Sie im Abschnitt *Kontrollstrukturen* dieses Kapitels noch kennen.



Hinweis

Der Aufruf der Prozedur kann nun folgendermaßen erfolgen:

```
SummeBerechnen 4, 5
SummeBerechnen 4, 5, 3, 5, 2
SummeBerechnen
```

Der Zugriff auf die einzelnen Elemente des Parameter-Arrays erfolgt auf die gleiche Weise wie bei Variant-Arrays. Hierzu muss hinter dem Namen des Parameter-Arrays ein Index innerhalb von numerischen Klammern angegeben werden:

```
<Variablenname>(<Index>)
```

Der erste übergebene Parameter hat stets den Index 0. Die Indexbereiche und damit die Anzahl der übergebenen Parameter können wie bei Variant-Arrays über die Funktionen *UBound* und *LBound* ermittelt werden.

Parameter-Arrays müssen grundsätzlich den Datentyp *Variant* haben und sind immer optional. Neben dem Parameter-Array können im Kopf der Prozedur noch weitere Parametervariablen definiert werden. Es darf allerdings immer nur ein Parameter-Array vorhanden sein. Außerdem muss dieses immer als letzter Parameter der Parameterliste angegeben werden.

Um festzustellen, ob für ein Parameter-Array mindestens ein Wert angegeben wurde, kann genau wie bei optionalen Parametern die *IsMissing*-Funktion verwendet werden. Der folgende Ausdruck hat den Wert *True* zum Ergebnis, wenn kein Wert angegeben wurde:

```
IsMissing(Zahlen())
```





Übergabe von Arrays

Auch die Übergabe von dynamischen Arrays an eine Prozedur ist möglich. Die folgende Prozedur hat beispielsweise die Aufgabe, die Inhalte aller Elemente eines Arrays im Testfenster aufzulisten. Die Prozedur erwartet dazu ein eindimensionales Array des Datentyps *Integer*:

```
Sub InhaltAusgeben (EinArray() As Integer)

    Dim i As Integer

    For i = LBound(EinArray) To UBound(EinArray)

        Debug.Print EinArray(i)

    Next i

End Sub
```

Hinweis



In der Prozedur wird wieder eine *For...Next*-Schleife benutzt, zu der Sie im Abschnitt *Kontrollstrukturen* weitere Informationen erhalten.

Der fertigen Prozedur können Sie nun ein beliebiges eindimensionales Array übergeben. Beim Aufruf der Prozedur muss dazu der Name des Arrays inklusive eines leeren Klammernpaars angegeben werden. Das folgende Beispiel definiert ein dynamisches Array mit dem Namen *TestDatenfeld*. Das Array wird mit einigen Zufallswerten zwischen 1 und 100 gefüllt und anschließend mithilfe der Prozedur *InhaltAusgeben* im Testfenster ausgegeben:

```
Sub Testprozedur23()

    Const Obergrenze = 30

    Dim i As Integer
    Dim Zufallszahl As Integer
    Dim TestDatenfeld() As Integer

    Randomize

    ReDim TestDatenfeld(1 To Obergrenze)

    For i = LBound(TestDatenfeld) To
        UBound(TestDatenfeld)

        Zufallszahl = Int((100 - 1 + 1) * Rnd + 1)
        TestDatenfeld(i) = Zufallszahl

    Next i

    InhaltAusgeben TestDatenfeld()

End Sub
```



Function-Prozeduren

Bis jetzt haben Sie nur Möglichkeiten kennen gelernt, Informationen in Form von Parametern an eine Prozedur zu übergeben. In der Praxis ist es aber auch erwünscht, Informationen – wie beispielsweise ein Rechenergebnis – von einer Prozedur zurückzuerhalten. Zu diesem Zweck stellt Access einen weiteren Typ von Prozeduren zur Verfügung – die *Function*-Prozeduren. Im Gegensatz zu den bisher vorgestellten Prozeduren beginnen *Function*-Prozeduren (oder kurz: Funktionen) nicht mit dem Schlüsselwort *Sub*, sondern mit dem Schlüsselwort *Function* und enden dementsprechend mit der Anweisung *End Function*:

```
{Private|Public} Function <Prozedurname>
  [(<Parameterliste>)] [As <Datentyp>]

  [<Anweisungen>]
```

End Function

Genau wie *Sub*-Prozeduren haben auch *Function*-Prozeduren eine Parameterliste, über die Informationen an die Prozedur übergeben werden. Zusätzlich wird hinter der Parameterliste jedoch noch ein weiterer Datentyp angegeben, der den Typ der zurückzugebenden Information angibt. Der Prozedurkopf

```
Function KreisflächeBerechnen (Radius As Double)
As Double
```

bedeutet beispielsweise, dass die Funktion einen Parameter des Typs *Double* erwartet und – nachdem die Prozedur beendet wurde – einen Wert des Typs *Double* zurückliefert. Die vollständige Prozedur zur Berechnung der Kreisfläche könnte folgenden Aufbau haben:

```
Function KreisflächeBerechnen (Radius As Double)
As Double
```

```
Const Pi = 3.14
Dim Fläche As Double
```

```
    Fläche = Radius ^ 2 * Pi
```

```
    KreisflächeBerechnen = Fläche
```

End Function

Innerhalb der Prozedur muss bestimmt werden, welches Ergebnis die Funktion bei ihrer Beendigung zurückgeben soll. Zu diesem Zweck muss das Ergebnis dem Namen der Funktion zugewiesen werden. In der Beispielprozedur wird dies in der vierten Zeile erledigt, indem dem Namen *KreisflächeBerechnen* der Inhalt der Variablen *Fläche* zugewiesen wird.





Hinweis



Sie können auch Funktionen entwickeln, die keinen Parameter erwarten und trotzdem einen Wert zurückgeben. In diesem Fall lassen Sie die *<Parameterliste>* einfach aus. Auch auf die Angabe des Arguments *As <Datentyp>* kann verzichtet werden. In diesem Fall deklariert Access das Ergebnis automatisch mit dem Datentyp *Variant*.

Einsatz von *Function*-Prozeduren



Die fertige Funktion kann nun innerhalb einer *Sub*-Prozedur oder einer anderen Funktion verwendet werden. Da sie ein Ergebnis zurückgibt, muss sie allerdings wie ein Ausdruck behandelt werden. In der Praxis bedeutet dies, dass Sie das Ergebnis z. B. einer Variablen zuweisen, als Parameter einer anderen Funktion verwenden oder innerhalb von Berechnungen einsetzen. Die folgenden Programmausschnitte zeigen einige Anwendungsbereiche:

```
Dim Kreisfläche As Double
Kreisfläche = KreisflächeBerechnen (3)
MsgBox Str(KreisflächeBerechnen (3))
Dim DoppelteKreisfläche As Double
DoppelteKreisfläche = KreisflächeBerechnen (3) *
KreisflächeBerechnen (3)
```

Wenn Sie eine Funktion im Direktfenster testen möchten, stellen Sie dem Namen einfach den *Print*-Befehl voran. Der folgende Ausdruck startet beispielsweise die Prozedur *KreisflächeBerechnen* vom Testfenster aus und gibt das Ergebnis aus:

```
Print KreisflächeBerechnen (5.25)
```

Hinweis



Die Verwendung benannter und optionaler Parameter ist bei Funktionen genau wie bei *Sub*-Prozeduren möglich.

Wenn Sie eine Funktion aufrufen möchten, die keine Parameter erwartet, müssen Sie trotzdem ein leeres Klammernpaar hinter dem Funktionsaufruf angeben. Ein Beispiel für eine Funktion ohne Parameter ist die folgende Prozedur:

```
Function HeuteInEinemMonat() As Variant

    HeuteInEinemMonat = DateAdd("m", 1, Date)

End Function
```

Diese Funktion ermittelt das Datum des Tages in genau einem Monat. Um sie aufzurufen, kann folgender Ausdruck verwendet werden:

```
Rechnungsfrist = HeuteInEinemMonat()
```

Der Start vom Direktfenster aus erfolgt auf die gleiche Art und Weise:

```
Print HeuteInEinemMonat()
```

Hinweis



Funktionen können in Access auch wie Anweisungen aufgerufen werden. Die Parameter der Funktion dürfen hierbei nicht in Klammern angegeben werden. Das Ergebnis der Funktion wird außerdem ignoriert.



Einsatz von Funktionen außerhalb von VBA

Wenn Sie eine öffentliche Funktion in einem Modul angelegt haben, ist sie nicht nur von VBA, sondern von der kompletten Datenbank aus nutzbar. Dazu gehören beispielsweise Makros, Eigenschaften von Formularen, Berichten oder Steuerelementen und auch Abfragen.

Wenn Sie beispielsweise in ein Textfeld eines Formulars standardmäßig das Datum in einem Monat eintragen lassen möchten, geben Sie für die *Standardwert*-Eigenschaft einfach den folgenden Ausdruck ein:

```
= HeuteInEinemMonat()
```

Auch die Verwendung innerhalb von Abfragen ist möglich. Der SQL-Text einer Abfrage könnte beispielsweise folgendermaßen aussehen:

```
SELECT * FROM Rechnungen WHERE Rechnungsfrist >=
HeuteInEinemMonat()
```

Selbst der Einsatz im Zusammenhang mit Ereignisseigenschaften ist möglich. Mehr dazu erfahren Sie im Abschnitt *Ereignisorientierte Programmierung* dieses Kapitels.

Call by reference und Call by value

Function-Prozeduren stellen eine komfortable Möglichkeit dar, eine Information von einer Prozedur zurückzuerhalten. Was ist aber, wenn Sie mehrere Informationen austauschen möchten?

Dieses Problem lässt sich auf einfache Weise lösen, indem Sie den Rückgabewert einer Funktion einfach mit einem benutzerdefinierten Datentyp deklarieren. Diese Vorgehensweise ist jedoch in vielen Fällen zu umständlich.

Eine andere Lösung für dieses Problem basiert auf der Art und Weise, wie Access Parameter an *Sub*- oder *Function*-Prozeduren übergibt. Um dies zu verstehen, ist etwas Wissen darüber erforderlich, wie Access Variablen intern handhabt.

Der Hauptspeicher Ihres Computers besteht aus einer Liste einzelner Speicherstellen. Jede Speicherstelle ist hierbei ein Byte groß. Da *Integer*-Zahlen einen Speicherbedarf von zwei Byte haben, sind dementsprechend zwei dieser Speicherstellen in der Lage, eine *Integer*-Zahl zu speichern. Jede dieser Speicherstellen besitzt eine Adresse, über die auf den Inhalt der Speicherstelle zugegriffen werden kann.

Wenn Sie nun innerhalb einer Prozedur eine *Integer*-Variable deklarieren, reserviert Access im Speicher Ihres Rechners genau zwei Speicherstellen. Die Adresse dieses Paares wird ebenfalls von Access vermerkt, damit später auf den Inhalt der Speicherstellen zugegriffen werden kann. Beim Aufruf einer anderen Prozedur mit Parametern übergibt Access nun nicht den Inhalt der Speicherstellen, sondern lediglich die Adresse des Speicherpaares. Die aufgerufene Prozedur greift also auf die gleiche

Speicherstellen





Speicherstelle zu wie die aufrufende. Das folgende Beispiel macht dies deutlich. Die erste Prozedur mit dem Namen *WertVerdoppeln* hat die Aufgabe, den als Parameter übergebenen Wert zu verdoppeln:

```
Sub WertVerdoppeln (Wert As Integer)
```

```
    Wert = Wert * 2
```

```
End Sub
```

Eine weitere Prozedur nutzt die *WertVerdoppeln*-Prozedur:

```
Sub Berechnung()
```

```
    Dim EineZahl As Integer
```

```
    EineZahl = 5
```

```
    Debug.Print EineZahl
```

```
    WertVerdoppeln EineZahl
```

```
    Debug.Print EineZahl
```

```
End Sub
```

In der zweiten Prozedur wird zunächst eine Variable mit dem Namen *EineZahl* deklariert, der anschließend der Wert 5 zugewiesen wird. In der dritten Zeile wird dieser Wert zur Kontrolle im Direktfenster ausgegeben.

Beim Aufruf der Prozedur *WertVerdoppeln* übergibt Access nun nicht den Wert 5, sondern die Speicherstelle, an der dieser Wert im Hauptspeicher abgelegt ist. Nach dem Start der Unterprozedur *WertVerdoppeln* wird die Adresse dieser Speicherstelle nun unter dem Namen *Wert* gehandhabt. Die beiden Variablen *Wert* und *EineZahl* "zeigen" also nun auf die gleiche Speicherstelle, die den Wert 5 enthält. Durch den Ausdruck *Wert = Wert * 2* wird der Inhalt dieser Speicherstelle verdoppelt und die Prozedur schließlich beendet. Als Nächstes führt Access die letzte Anweisung der Prozedur *Berechnung* aus, die den Inhalt der Speicherstelle im Testfenster ausgibt. Da dieser Inhalt aber durch die Prozedur *WertVerdoppeln* verändert wurde, ist dies nicht mehr der Wert 5, sondern der Wert 10.

Diese Technik, bei der anstelle des eigentlichen Wertes lediglich die Speicherstelle an eine Prozedur übergeben wird, bezeichnet man auch als *Call by reference*. Mit *Reference* ist hierbei ein Zeiger auf eine Speicherstelle bzw. die Adresse einer Speicherstelle gemeint. In der Praxis wird fast ausschließlich *Call by reference* verwendet. Es muss keine Kopie der Speicherstelle erzeugt werden, und die Subroutine wird deutlich schneller aktiviert.





1. Dimension

2. Dimension

	1	2	3	4	5	6	7
1	32	5	43	2	8	0	12
2	32	45	43	5	76	0	9
3	6	8	3	5	3	2	6
4	98	65	43	23	66	66	433
5	2	33	33	234	67	98	2

Elemente

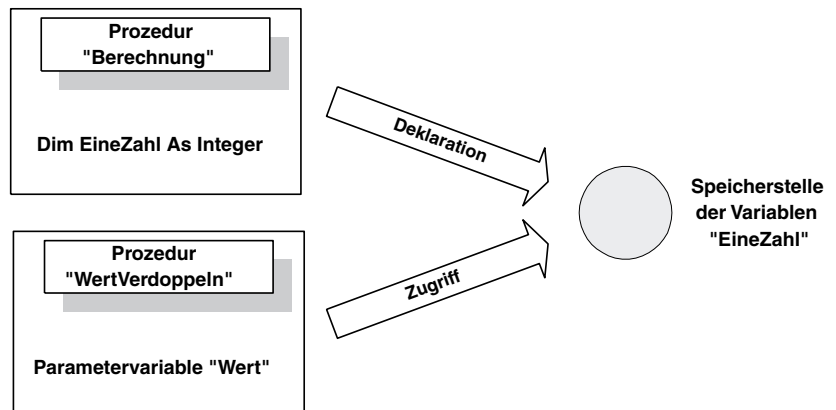
Abb. 3.21

Call by reference

Das Gegenstück dazu ist die Parameterübergabe per *Call by value*. Hierbei wird vor der Übergabe eine Kopie der Speicherstelle angelegt und schließlich die Adresse dieser Kopie übergeben. Sobald die Unterprozedur beendet ist, wird diese Kopie wieder gelöscht. Die Unterprozedur kann also keinen Einfluss auf Variablen der Hauptprozedur nehmen.

Abb. 3.22

Call by value



Um *Call by Value* mit Access zu erzwingen, muss in der Parameterliste der aufzurufenden Prozedur dem Namen der jeweiligen Variablen das Schlüsselwort *ByVal* vorangestellt werden. Da dies bei jedem einzelnen Parameter erfolgen kann, kön-

Das Schlüsselwort

ByVal





nen Sie separat festlegen, welcher Parameter per *Call by value* oder per *Call by reference* übergeben werden soll. Das folgende Beispiel zeigt die dementsprechend veränderte Prozedur *WertVerdoppeln*:

```
Sub WertVerdoppeln (ByVal Wert As Integer)

    Wert = Wert * 2

End Sub
```

Wenn Sie nun noch einmal die Prozedur *Berechnung* starten, gibt diese im Direktfenster zweimal hintereinander den Wert 5 aus. Die Prozedur *WertVerdoppeln* kann nun also den Wert der Variablen *EineZahl* nicht mehr verändern.

Das folgende Beispiel zeigt Ihnen eine Prozedur zur Berechnung des Kreisumfangs und der Kreisfläche anhand eines Kreisradius. Die Prozedur benötigt für diese Berechnung den Radius und gibt zwei Informationen zurück, nämlich den Kreisumfang und die Kreisfläche:

```
Sub KreisdatenBerechnen (Radius As Double, Umfang
As Double, Fläche As Double)

Const Pi = 3.14

    Fläche = 2 * Pi * Radius
    Umfang = Radius ^ 2 * Pi

End Sub
```

Um die Prozedur nun zu verwenden, stellen Sie zwei leere Variablen bereit, die den Inhalt der Berechnung aufnehmen:

```
Sub Kreis()

Dim MeinUmfang As Double
Dim MeineFläche As Double
Dim Radius As Double

    Radius = 5

    KreisdatenBerechnen Radius, MeinUmfang, MeineFläche

    Debug.Print MeinUmfang
    Debug.Print MeineFläche

End Sub
```





In der Praxis sollten Sie möglichst immer *Call by reference* verwenden. Da Access hierbei keine Kopie einer Speicherstelle anlegen muss, kann der Aufruf einer Unterprozedur schneller erfolgen! Aus diesem Grund wird in der obigen Prozedur bei dem Parameter *Radius* ebenfalls *Call by reference* angewendet, obwohl dieser Wert innerhalb der Prozedur nicht verändert wird. Nur bei Problemstellungen, in denen Sie eine unbeabsichtigte Veränderung eines Variableninhalts ausschließen möchten, sollten Sie *Call by value* vorziehen.



Hinweis

Kontrollstrukturen

Entscheidungs- und Schleifenstrukturen beeinflussen die Reihenfolge, in der die Anweisungen einer Prozedur ausgeführt werden. In diesem Abschnitt wird die Verwendung aller Kontrollstrukturen erläutert. Eine Ausnahme ist lediglich die *For Each...Next*-Schleife, die im Zusammenhang mit Objektauflistungen eingesetzt wird. Mehr dazu erfahren Sie im folgenden *Kapitel 4 Programmieren mit Objekten*.

If...Then...Else

Eines der wichtigsten Konstrukte ist in diesem Zusammenhang die *If*-Anweisung, die Sie bereits aus dem *Kapitel 2 Schnellstart VBA* kennen.

Mithilfe der *If*-Anweisung kann die Ausführung einer bestimmten Anweisung oder einer Reihe von Anweisungen von einer Bedingung abhängig gemacht werden. Als Bedingung eignet sich ein beliebiger Ausdruck des Datentyps *Boolean*, der das Ergebnis *True* oder *False* hat. Eine typische *If*-Anweisung sieht beispielsweise folgendermaßen aus:

```
If A = 5 Then Debug.Print "A hat den Wert 5"
```

Hat die Variable *A* den Inhalt 5, ist das Ergebnis des Ausdrucks *True*, andernfalls ist es *False*. Von diesem Ergebnis ist abhängig, ob die Anweisung hinter dem *Then*-Schlüsselwort ausgeführt wird (*True*) oder nicht (*False*).

Bei der oben vorgestellten Syntax ist immer nur die Ausführung einer einzelnen Anweisung von einer Bedingung abhängig. Im *Kapitel 2 Schnellstart VBA* haben Sie bereits eine Möglichkeit kennen gelernt, mehrere Anweisungen zu einem Block zusammenzufassen. In diesem Fall muss die folgende Syntax verwendet werden:

```
If <Bedingung> Then  
  
    <Anweisungsblock>  
  
End If
```





Access führt den Anweisungsblock – der aus mehreren hintereinander folgenden Anweisungen bestehen kann – nur dann aus, wenn die *<Bedingung>* den Wert *True* besitzt. Andernfalls wird die Ausführung der Prozedur nach der *End If*-Anweisung fortgesetzt. Es folgt ein Beispiel, bei dem der Benutzer eine Zahl eingibt. Ist diese Zahl größer als 10, erscheint eine Meldung:

```
Sub Testprozedur24()  
  
Dim Eingabe As String  
  
    Eingabe = InputBox("Geben Sie eine Zahl ein!")  
  
    If Val(Eingabe) > 10 Then  
  
        Beep  
        MsgBox "Die Zahl ist größer als 10!"  
  
    End If  
  
End Sub
```

Das *Else*-Schlüsselwort

In der Praxis tauchen oft Problemstellungen auf, bei denen im *True*-Fall der eine und im *False*-Fall ein anderer Anweisungsblock ausgeführt werden soll. Zu diesem Zweck kann die *If...Then*-Anweisung um das Schlüsselwort *Else* erweitert werden:

```
If <Bedingung> Then  
  
    <Anweisungsblock1> :Rem Anweisungen des Then-Teils  
  
Else  
  
    <Anweisungsblock2> :Rem Anweisungen des Else-Teils  
  
End If
```

Der *<Anweisungsblock1>* wird ausgeführt, wenn die Bedingung den Wert *True* hat. Andernfalls führt Access den *<Anweisungsblock2>* aus. Das folgende Beispiel erweitert die vorhergehende Textprozedur um einen entsprechenden *Else*-Teil:

```
Sub Testprozedur25()  
  
Dim Eingabe As String  
  
    Eingabe = InputBox("Geben Sie eine Zahl ein!")  
  
    If Val(Eingabe) > 10 Then  
  
        Beep  
        MsgBox "Die Zahl ist größer als 10!"  
  
    Else  
  
        Beep  
        MsgBox "Die Zahl ist kleiner als 10!"  
  
    End If  
  
End Sub
```





```
Else

    Beep
    MsgBox "Die Zahl ist kleiner oder gleich 10!"

End If

End Sub
```

Auch im Zusammenhang mit der *Else*-Anweisung gibt es wieder eine einzeilige Variante, die nur dann verwendet werden kann, wenn jeweils eine einzelne Anweisung ausgeführt werden soll:

```
If <Bedingung> Then <Anweisung1> Else <Anweisung2>
```

Da diese Variante nicht sehr übersichtlich ist, sollten Sie sie möglichst vermeiden. Dies ist ohne Probleme möglich, da ein Anweisungsblock natürlich auch lediglich aus einer einzelnen Anweisung bestehen kann.



Hinweis

Eine weitere Variante bringt die Anweisung *Elseif* ins Spiel. Sie ermöglicht es, im *False*-Fall eine weitere Bedingung zu überprüfen, die dann wiederum zur alternativen Ausführung zweier Anweisungsblöcke führt:



Elseif

```
If <Bedingung1> Then

    <Anweisungsblock1>

ElseIf <Bedingung2> Then

    <Anweisungsblock2>

Else

    <Anweisungsblock3>

End If
```

In der Praxis könnte dies folgendermaßen aussehen:

```
Sub Testprozedur26()

    Dim Eingabe As String

    Eingabe = InputBox("Geben Sie eine Zahl ein!")

    If Val(Eingabe) > 10 Then

        MsgBox "Die Zahl ist größer als 10!"

    ElseIf Val(Eingabe) < 5 Then
```





```
MsgBox "Die Zahl ist kleiner als 5"

Else

MsgBox "Die Zahl liegt zwischen 5 und 10 _
      (inklusive) "

End If

End Sub
```

Verschachtelte If-Anweisungen

Hinweis



If-Anweisungen lassen sich beliebig verschachteln. Es ist beispielsweise möglich – und in der Praxis auch oft erforderlich – innerhalb des *True*- oder *False*-Teils einer If-Anweisung eine weitere If-Anweisung zu verwenden.

In der Praxis taucht oft das Problem auf, dass Sie einer Variablen in Abhängigkeit von einer bestimmten Bedingung den einen von zwei Werten zuweisen möchten. Um dies zu vereinfachen, stellt Access die *IIF*-Funktion (deutsch *Wenn*) zur Verfügung. Die Funktion hat folgenden Aufbau:

```
<Ergebnis> = IIF(<Bedingung>, <True-Ausdruck>,
<False-Ausdruck>)
```

Je nachdem, ob die *<Bedingung>* *True* oder *False* zum Ergebnis hat, gibt die Funktion entweder *<True-Ausdruck>* oder *<False-Ausdruck>* als Ergebnis zurück. Dieses kann dann z. B. in einer Variablen gespeichert werden. Der Datentyp des Ergebnisses stimmt hierbei mit dem Datentyp des jeweiligen Ausdrucks überein.

Select...Case

Die *Select...Case*-Anweisung bietet mehr Flexibilität als die *If...Then...Else*-Anweisung, wenn es darum geht, auf mehrere verschiedene Ergebnisse eines Ausdrucks zu reagieren. Die Syntax der *Select...Case*-Anweisung lautet folgendermaßen:

```
Select Case <Testausdruck>

    [Case <Ausdruck1>
    [<Anweisungsblock1>] ]

    [Case <Ausdruck2>
    [<Anweisungsblock2>] ]

    [Case Else
    [<ElseAnweisungsblock>] ]

End Select
```



<Testausdruck> kann ein beliebiger numerischer Ausdruck oder ein Zeichenkettenausdruck sein. Zwischen den Anweisungen *Select Case* und *End Select* können mehrere Anweisungsblöcke angegeben werden, wobei jeder mithilfe der *Case*-Anweisung und mithilfe eines <Ausdruck> gekennzeichnet wird. Bei der Ausführung überprüft Access nun, welcher <Ausdruck> mit dem <Testausdruck> übereinstimmt. Der entsprechende Anweisungsblock wird schließlich ausgeführt. Sollte Access keinen passenden Block ausfindig machen können, wird – falls vorhanden – der <ElseAnweisungsblock> ausgeführt.

Das folgende Beispiel zeigt die praktische Verwendung der *Select Case*-Anweisung:

```
Sub Testprozedur27()  
  
Dim Eingabe As String  
  
    Eingabe = InputBox("Geben Sie eine Zahl ein!")  
  
    Select Case Val(Eingabe)  
  
        Case 1  
            MsgBox "Sie haben die Zahl 1 eingegeben!"  
  
        Case 2  
            MsgBox "Sie haben die Zahl 2 eingegeben!"  
  
        Case 3  
            MsgBox "Sie haben die Zahl 3 eingegeben!"  
  
        Case Else  
            MsgBox "Sie haben die Zahl ungleich 1, " _  
                & "2 oder 3 eingegeben!"  
  
    End Select  
  
End Sub
```

In diesem Beispiel wird vom Benutzer ein Zahlenwert eingegeben. Je nachdem, ob 1, 2 oder 3 eingegeben wird, führt Access den entsprechenden Anweisungsblock aus. Sollte eine andere Zahl eingegeben werden, führt Access die Anweisung hinter der *Case Else*-Anweisung aus.

Die Ausdrücke hinter den *Case*-Anweisungen können nicht nur aus einzelnen Zahlen oder Zeichenketten bestehen, sondern aus einer Liste mehrerer Ausdrücke, die jeweils durch ein Komma voneinander getrennt werden. Der zugehörige Anweisungsblock wird nur dann ausgeführt, wenn einer der Ausdrücke mit dem <Testausdruck> übereinstimmt. Das obige Beispiel könnte beispielsweise vor dem *Case Else*-Block durch die folgenden Zeilen erweitert werden:

```
Case 4,5,6  
    MsgBox " Sie haben die Zahl 4, 5 oder 6 eingegeben"
```

Ausdruckslisten





For...Next

Bei der *For...Next*-Anweisung handelt es sich um eine Schleifenstrukturanweisung, die es ermöglicht, einen Anweisungsblock mehrmals auszuführen. Die Anzahl der Wiederholungen lässt sich hierbei festlegen. Die allgemeine Syntax lautet folgendermaßen:

```
For <Zähler> = <Startwert> To <Endwert>
```

```
    [<Anweisungsblock>]
```

```
Next [<Zähler>]
```

<Zähler> ist eine normale Variable, die bei jedem Durchgang automatisch von Access um den Wert 1 erhöht wird. Access beginnt die Zählung hierbei mit dem unter <Startwert> angegebenen Wert und stoppt, sobald der <Endwert> erreicht ist. Für <Startwert> und <Endwert> können beliebige numerische Ausdrücke – also entweder direkt Zahlenwerte oder auch numerische Variablen – angegeben werden. In dem folgenden Beispiel wird eine Anweisung, die einen Text im Testfenster ausgibt, mehrmals wiederholt. Die Anzahl wird hierbei vorher vom Benutzer abgefragt:

```
Sub Testprozedur28()  
  
Dim Zähler As Integer  
Dim Anzahl As Integer  
  
    Anzahl = Val(InputBox("Bitte geben Sie die " _  
        & "Anzahl der Wiederholungen an!"))  
  
    For Zähler = 1 To Anzahl  
  
        Debug.Print "Das ist Durchlauf Nr." & Str(Zähler)  
  
    Next Zähler  
  
End Sub
```

Beim Eintritt in die *For*-Schleife setzt Access die Zählvariable *Zähler* zuerst auf den <Startwert> (hier 1) und führt alle Anweisungen bis zu dem Schlüsselwort *Next* aus. Danach erhöht Access die Zählvariable um den Wert 1. Wenn der Wert der Zählvariable kleiner ist als der <Endwert> (hier der Inhalt der Variablen *Anzahl*), durchläuft Access die Schleife erneut. Andernfalls wird die Programmausführung mit der ersten Anweisung hinter dem *Next*-Schlüsselwort fortgesetzt.

Das Ende der Schleife wird mit dem Schlüsselwort *Next* markiert, dem noch einmal der Name der Zählvariablen folgt. Durch die Angabe der Zählvariablen ist sofort ersichtlich, zu welchem Schleifenkopf die *Next*-Anweisung gehört. Access erlaubt es jedoch, die Zählvariable auszulassen. In diesem Fall bezieht sich die *Next*-Anweisung immer automatisch auf die zuletzt gestartete Schleife.

Hinweis



Hinter dem *<Endwert>* kann in Verbindung mit dem Schlüsselwort *Step* eine Schrittgröße angegeben werden:



```
For <Zähler> = <Startwert> To <Endwert>  
[Step <Schrittgröße>]
```

In diesem Fall erhöht Access den *<Zähler>* bei jedem Durchlauf der Schleife nicht um den Wert 1, sondern um den unter *<Schrittgröße>* angegebenen Wert. Der Schleifenkopf

```
For Zähler = 1 To 6 Step 2
```

führt also dazu, dass Access den Zähler in Zweierschritten hochzählt und die Schleife damit nur dreimal durchläuft. Für *<Schrittgröße>* können auch negative Werte angegeben werden. In diesem Fall zählt Access rückwärts. Hierbei ist allerdings darauf zu achten, dass *<Startwert>* einen größeren Wert als *<Endwert>* aufweist.

Um eine Schleife zu verlassen, bevor *<Endwert>* erreicht ist, stellt Access die *Exit For*-Anweisung zur Verfügung. Diese Anweisung bewirkt, dass die Programmausführung sofort mit der ersten Anweisung hinter dem *Next*-Schlüsselwort fortgesetzt wird.



Hinweis

In VBA ist es zulässig, mehrere *For*-Schleifen ineinander zu verschachteln, wie das folgende Beispiel zeigt:

```
Sub Testprozedur29()  
  
Dim Zähler1 As Integer, Zähler2 As Integer  
  
For Zähler1 = 1 To 5  
  
    Debug.Print "Äußere Schleife; Durchlauf " _  
        & Str$(Zähler1)  
  
    For Zähler2 = 1 To 10  
  
        Debug.Print "Innere Schleife; Durchlauf " _  
            & Str$(Zähler2)  
  
    Next Zähler2  
  
Next Zähler1  
  
End Sub
```





Kopfgesteuerte Schleife

Do...Loop

Die *Do...Loop*-Anweisung erfüllt eine ähnliche Aufgabe wie die *For...Next*-Anweisung. Die Anzahl der Wiederholungen wird jedoch nicht beim Eintritt der Schleife festgelegt, sondern ist von einer Bedingung abhängig. Die allgemeine Syntax der *Do...Loop*-Anweisung lässt sich in zwei Varianten zerlegen. Die erste hat folgende Syntax:

```
Do [{While|Until} <Bedingung>]

    [<Anweisungsblock>]
```

Loop

Das Argument *<Bedingung>* muss wie das gleichnamige Argument der *If...Then...Else*-Anweisung ein Ausdruck des Typs *Boolean* sein, der einen der logischen Werte *True* oder *False* zum Ergebnis hat. Die entsprechenden numerischen Werte sind ebenfalls zulässig.

Der Kopf der *Do*-Anweisung lässt die folgenden beiden Alternativen zu:

```
Do While <Bedingung>
Do Until <Bedingung>
```

Wenn Sie die Alternative *Do While* verwenden, wiederholt Access den *<Anweisungsblock>* so lange, wie die *<Bedingung>* den Wert *True* zum Ergebnis hat. Die *<Bedingung>* wird hierbei beim Eintritt in die Schleife und nach jedem Durchlauf erneut ausgewertet. Sollte sie den Wert *False* zum Ergebnis haben, wird die Schleife verlassen und die Programmausführung hinter der *Loop*-Anweisung fortgesetzt.

Wenn Sie dagegen *Do Until* verwenden, wiederholt Access den *<Anweisungsblock>* so lange, wie *<Bedingung>* den Wert *False* zum Ergebnis hat. Erst wenn das Ergebnis *True* ist, wird die Schleife verlassen. Die beiden folgenden Schleifenköpfe haben also genau die gleiche Wirkung:

```
Do While <Bedingung>
Do Until Not <Bedingung>
```

Hinweis



Sollte die Bedingung bereits beim Eintritt in die *Do While*-Schleife das Ergebnis *False* haben, wird die Schleife komplett übersprungen und damit keine Anweisung des Anweisungsblocks ausgeführt. Das Gleiche gilt bei einer *Do Until*-Schleife, falls die Bedingung bereits beim Eintritt den Wert *True* besitzt.

Fußgesteuerte Schleife

Die zweite Variante der *Do...Loop*-Anweisung hat im Prinzip denselben Aufbau, die Wiederholungsbedingung befindet sich jedoch nicht am Anfang, sondern am Ende der Schleife. Dadurch wird die Schleife im Gegensatz zu der ersten Variante unabhängig von der *<Bedingung>* mindestens einmal durchlaufen:



Do

[<Anweisungsblock>]

Loop [{While|Until} <Bedingung>]

Das folgende Beispiel fordert den Benutzer zur Eingabe einer Zahl zwischen 1 und 10 auf. Liegt die Eingabe außerhalb dieses Bereichs, muss sie wiederholt werden:

```
Sub Testprozedur30()
```

```
Dim Eingabe As String
```

```
Do
```

```
    Eingabe = InputBox("Bitte geben Sie eine Zahl _  
                      zwischen 1 und 10 ein!")
```

```
Loop Until Val(Eingabe) >= 1 And Val(Eingabe) <= 10
```

```
End Sub
```

Bei *Do...Loop*-Schleifen ist darauf zu achten, dass die Schleife auch wirklich beendet werden kann. Voraussetzung dafür ist, dass die <Bedingung> innerhalb der Schleife verändert wird. Bei der folgenden Schleife ist dies zwar der Fall, die Ausführung bricht jedoch trotzdem nicht ab. Solche Schleifen werden daher auch als *Endlosschleifen* bezeichnet:

Endlosschleifen

```
Sub LoopTest()
```

```
Dim x As Integer
```

```
    x = 0
```

```
Do While x >= 0
```

```
    x = x + 1: Rem x wird um den Wert 1 erhöht
```

```
Loop
```

```
End Sub
```

Access unterbricht die Schleife nach kurzer Zeit durch einen Laufzeitfehler, da der Wert der Variablen *x* den Wertebereich einer *Integer*-Variablen überschreitet.



Exit Do

Manchmal ist es sinnvoll, eine Schleife direkt und unabhängig von der <Bedingung> zu verlassen. Zu diesem Zweck kann im Anweisungsblock die Anweisung *Exit Do* angegeben werden. Die folgende Prozedur ist eine Erweiterung der vorhergehenden Prozedur. Hier wird zusätzlich überprüft, ob der Benutzer die Eingabe abgebrochen hat. In diesem Fall liefert die *InputBox*-Funktion eine leere Zeichenkette zurück, und die Schleife wird verlassen:

```
Sub Testprozedur31()  
  
Dim Eingabe As String  
  
Do  
  
    Eingabe = InputBox("Bitte geben Sie eine Zahl " _  
        & "zwischen 1 und 10 ein!")  
  
    If Eingabe = "" Then  
  
        Beep  
        MsgBox "Abbruch!"  
        Exit Do  
  
    End If  
  
    Loop Until Val(Eingabe) >= 1 And Val(Eingabe) <= 10  
  
End Sub
```

Verwendet wird die *Exit Do*-Anweisung außerdem noch bei Schleifen folgender Art:

```
Sub LoopTest2()  
  
Dim x As Integer  
  
x = 0  
  
Do  
  
    x = x + 1: Rem x wird um den Wert 1 erhöht  
  
    If x = 100 Then Exit Do  
  
    Debug.Print x  
  
Loop  
  
End Sub
```



In diesem Beispiel wurde die Bedingung einfach ausgelassen. Access wiederholt die Schleife in diesem Fall so lange, bis sie mithilfe der *Exit Do*-Anweisung verlassen wird.

While...Wend

Eine *While...Wend*-Schleife ist nichts anderes als eine *Do...Loop*-Schleife mit eingeschränktem Funktionsumfang. Die Syntax hat folgende Struktur:

```
While <Bedingung>

    [<Anweisungsblock>]
Wend
```

Auch hier wird die Abarbeitung des Anweisungsblocks so lange wiederholt, bis die Bedingung den Wert *False* ausweist. Die gleiche Funktion kann über die *Do...Loop*-Anweisung erzielt werden:

```
Do While <Bedingung>

    [<Anweisungsblock>]
Loop
```

Zeilensprungmarken

Ein etwas veraltetes Mittel zur Strukturierung von Programmcode sind Sprungmarken. Sie stammen noch aus der Zeit, als Basic-Programmiersprachen noch keine Prozeduren unterstützten.

Eine Sprungmarke ist nichts anderes als eine Markierung, die eine bestimmte Stelle innerhalb einer Prozedur kennzeichnet. Diese Markierung besteht aus einem Namen, dem ein Doppelpunkt folgt. Der folgende Ausdruck markiert beispielsweise eine Sprungmarke mit dem Namen *ZahlEingeben*:

ZahlEingeben:

Der Name darf aus maximal 200 Zeichen bestehen und mit keinem reservierten Schlüsselwort von Access übereinstimmen! Außerdem muss der Name der Sprungmarke für die aktuelle Prozedur eindeutig sein. Es ist jedoch zulässig, dass sich der Name einer Sprungmarke innerhalb eines Moduls wiederholt.

Mithilfe der *GoTo*-Anweisung kann nun von einer anderen Stelle der Prozedur aus zu dieser Marke gesprungen werden, und Access setzt die Abarbeitung des Programms direkt hinter der Sprungmarke fort.

Die *GoTo*-Anweisung erwartet als Parameter den Namen einer Sprungmarke. Hierbei ist es notwendig, dass sich die Sprungmarke innerhalb der aktuellen Prozedur befindet. Sprünge in andere Prozeduren sind nicht möglich. Das folgende Beispiel





verwendet die *GoTo*-Anweisung und eine Sprungmarke, um die Eingabe einer Zahl zwischen 1 und 5 zu erzwingen:

```
Sub Testprozedur32()  
  
Dim Eingabe As String  
  
ZahlEingeben:  
  
    Eingabe = InputBox("Bitte geben Sie eine Zahl _  
                        zwischen 1 und 10 ein!")  
  
    If Val(Eingabe) < 1 Or Val(Eingabe) > 10 Then  
  
        Beep  
        MsgBox "Ihre Eingabe liegt außerhalb des _  
                gültigen Bereichs!"  
        GoTo ZahlEingeben  
  
    End If  
  
End Sub
```

Hinweis



Sprungmarken und die *GoTo*-Anweisung sollten Sie bei der Programmierung auf jeden Fall vermeiden, da sie zu sehr unübersichtlichem Programmcode führen. Mithilfe der Schleifenstrukturen wie *Do...Loop* oder *For...Next* lässt sich dies fast immer bewerkstelligen. Der einzige Bereich, in dem Sprungmarken weiterhin verwendet werden müssen, ist die Behandlung von Laufzeitfehlern. Mehr dazu erfahren Sie im folgenden Abschnitt *Fehlersuche und Fehlerbehandlung*.

Fehlersuche und Fehlerbehandlung

Bei der Programmierung mit Access können grundsätzlich zwei Arten von Fehlern auftreten: Kompilierfehler und Laufzeitfehler. Kompilierfehler werden von Access bereits während der Kompilierung erkannt. Ein typischer Kompilierfehler ist beispielsweise, dass Sie eine Variable verwenden, die vorher nicht deklariert wurde.

Laufzeitfehler treten dagegen erst während der Ausführung des Programms auf. Sie führen dazu, dass Access die Programmausführung unterbricht und die fehlerhafte Programmstelle im Modulfenster anzeigt. Ein typischer Laufzeitfehler kann beispielsweise in der folgenden Prozedur auftreten:

```
Sub Testprozedur33()  
  
Dim Zahl1 As Double  
Dim Zahl2 As Double  
Dim Ergebnis As Double
```





```
Zahl1 = Val(InputBox("Bitte die erste Zahl _  
eingegeben!"))  
Zahl2 = Val(InputBox("Bitte die zweite Zahl _  
eingegeben!"))  
  
Ergebnis = Zahl1 / Zahl2  
  
MsgBox Str(Ergebnis)
```

End Sub

Die Prozedur fordert den Benutzer zur Eingabe zweier Zahlen auf. Anschließend wird die erste Zahl durch die zweite dividiert. Ein Laufzeitfehler kann nun auftreten, wenn der Benutzer als zweite Zahl eine *NULL* eingibt. Bei der Berechnung erfolgt dann eine Division durch *NULL*, die den Laufzeitfehler auslöst.

Eine andere Ursache für einen Laufzeitfehler kann z.B. die Überschreitung eines Wertebereichs einer Variablen sein oder der Zugriff auf eine Datei, die nicht existiert. Mit Access lassen sich solche Fehler abfangen und dann innerhalb des Programms verarbeiten und im Idealfall auch beheben.

Wie bereits im vorhergehenden Abschnitt erwähnt, basiert die Technik der Fehlerbehandlung auf Sprungmarken. Mithilfe der *On Error GoTo*-Anweisung kann eine Sprungmarke bestimmt werden, die Access beim Auftreten eines Laufzeitfehlers anspricht. Der Programmcode hinter dieser Sprungmarke wird auch als *Fehlerbehandlungsroutine* bezeichnet. Damit die Behandlung von Laufzeitfehlern direkt zu Beginn der Prozedur aktiviert wird, sollten Sie die Anweisung möglichst bereits am Anfang der Prozedur angeben.

 **On Error-Anweisung**

Die folgende Prozedur fängt die Division durch *NULL* ab und gibt eine entsprechende Meldung aus:

```
Sub Testprozedur34()  
  
Dim Zahl1 As Double  
Dim Zahl2 As Double  
Dim Ergebnis As Double  
  
On Error GoTo Err  
  
Zahl1 = Val(InputBox("Bitte die erste Zahl _  
eingegeben!"))  
Zahl2 = Val(InputBox("Bitte die zweite Zahl _  
eingegeben!"))  
  
Ergebnis = Zahl1 / Zahl2  
  
MsgBox Str(Ergebnis)  
  
Exit Sub
```





Err:

```
MsgBox "In der Prozedur "Testprozedur34" " _  
      & "ist ein Fehler aufgetreten!"
```

```
Exit Sub
```

```
End Sub
```

Sobald nun ein Fehler auftritt, verzweigt Access zu der Sprungmarke *Err* und zeigt das Meldungsfenster mit der Fehlermeldung an. Sollte aber kein Fehler auftreten – was vermutlich die Regel sein dürfte – kann das Ergebnis der Berechnung ohne Probleme angezeigt werden. Direkt danach führt Access die Anweisung *Exit Sub* aus, die zum Verlassen der Prozedur führt. Diese Anweisung ist unbedingt notwendig, da Access sonst die Programmausführung über die Sprungmarke *Err* hinaus fortsetzen und damit auch noch die Fehlermeldung anzeigen würde. Eine *Sub*-Prozedur mit Fehlerbehandlungsroutine hat damit in der Regel den folgenden Aufbau:

```
Sub <Prozedurname> (<Parameterliste>)
```

```
On Error Goto <Sprungmarke>
```

```
[<Anweisung>]
```

```
Exit Sub
```

```
<Sprungmarke>:
```

```
<Anweisungen zur Fehlerbehandlung>
```

```
Exit Sub
```

```
End Sub
```

Wird nach dem Auftreten eines Laufzeitfehlers zu einer bestimmten Sprungmarke verzweigt, führt Access die hier angegebene Fehlerbehandlungsroutine aus. Nachdem dies erledigt ist, muss Access mitgeteilt werden, ob es die Prozedur verlassen oder die Abarbeitung mit der fehlerhaften Programmstelle fortsetzen soll. Letzteres ist beispielsweise dann sinnvoll, wenn Sie den Fehler in der Fehlerbehandlungsroutine beheben konnten.

Resume-Anweisung

Zum Verlassen der Prozedur geben Sie – wie in dem vorherigen Beispiel – einfach die *Exit Sub*-Anweisung an. Um die Prozedur dagegen fortzusetzen, muss die *Resume*-Anweisung verwendet werden. Hierbei sind die folgenden drei Varianten möglich:



Resume-Anweisung

Argument	Beschreibung
<i>Resume [o]</i>	Das Programm wird mit der Anweisung fortgesetzt, die den Fehler verursacht hat. Die fehlerhafte Anweisung wird also wiederholt. Der Einsatz dieser Anweisung ist nur dann sinnvoll, wenn der Fehler in der Fehlerbehandlungsroutine behoben werden konnte. Die Angabe der Zahl o ist optional, hat keine Bedeutung.
<i>Resume Next</i>	Das Programm wird nach der Anweisung fortgesetzt, die den Fehler verursacht hat. Die fehlerhafte Anweisung wird also nicht erneut ausgeführt.
<i>Resume <Sprungmarke></i>	Die Programmausführung wird an der <Sprungmarke> fortgesetzt.

Tabelle 3.36

Mithilfe der *On Error GoTo*-Anweisung haben Sie die Möglichkeit, auf einen Fehler zu reagieren. Um dies aber optimal erledigen zu können, müssen Sie Informationen über die Art des Fehlers haben. Access kennt eine Vielzahl von Laufzeitfehlern, wobei jedem dieser Fehler eine Nummer – die Laufzeitfehlernummer – zugewiesen ist. Mithilfe des Objekts *Err* können Sie die Nummer des zuletzt aufgetretenen Fehlers ermitteln. Dazu muss auf die *Number*-Eigenschaft des Objekts zugegriffen werden. Das folgende Beispiel hat die Nummer des zuletzt aufgetretenen Fehlers zum Ergebnis:

```
Err.Number
```

Eine Beschreibung des Fehlers in Form einer Zeichenkette erhalten Sie mithilfe der *Description*-Eigenschaft:

```
Err.Description
```

Access besitzt eine Vielzahl unterschiedlicher Fehlermeldungen, deren Auflistung den Rahmen dieses Buches sprengen würde. Bei der Programmierung ist es jedoch oft erforderlich, auf einen bestimmten Fehler in einer genau definierten Weise zu reagieren. Sie können zu diesem Zweck den Inhalt des Ausdrucks *Err.Number* mit einer bestimmten Fehlernummer vergleichen.

Die Frage ist nun, welcher Fehler welche Laufzeitfehlernummer hat. Um die Fehlernummer eines bestimmten Laufzeitfehlers herauszufinden, können Sie beispielsweise innerhalb der Fehlerbehandlungsroutine die Anweisung

```
Debug.Print Err
```

Ermittlung der
Fehlernummern





zur Ausgabe der Fehlernummer angeben und anschließend die wahrscheinlich auftretenden Laufzeitfehler provozieren. Die Fehlernummern können Sie nun notieren. Auf diese Weise können Sie auf alle Laufzeitfehler, mit deren Auftreten Sie rechnen, gezielt reagieren. Bei allen übrigen, nicht geplanten Fehlern beenden Sie die Prozedur oder ignorieren den Fehler.

Der folgende Programmausschnitt zeigt den Aufbau der erweiterten Fehlerbehandlungsroutine der obigen Beispielprozedur. Vorher wurde für den Laufzeitfehler *Division durch NULL* die Fehlernummer 11 ermittelt:

```
Err:

    If Err.Number = 11 Then

        MsgBox "Eine Division durch 0 ist nicht
        zulässig!"

    Else

        MsgBox Err.Description

    End If

    Exit Sub

End Sub
```

Die Fehlerbehandlungsroutine einer Prozedur wird beim Verlassen dieser Prozedur automatisch deaktiviert. Soll die Deaktivierung jedoch noch während der Ausführung der Prozedur erfolgen, muss die folgende Anweisung an einer beliebigen Stelle innerhalb der Prozedur angegeben werden:

```
On Error GoTo 0
```

Hinweis



Üblicherweise werden Fehlerbehandlungsroutinen bereits bei der Entwicklung eines Programms verwendet. Während der Testphase ist es aber vorteilhaft, wenn Access die Fehler nicht abfängt, sondern die übliche Meldung ausgibt und den Debug-Modus aktiviert. Eine gute Lösung für dieses Problem besteht darin, dass Sie eine öffentliche Konstante mit dem Namen *cDebugModus* anlegen. Während der Testphase des Programms weisen Sie dieser den Wert *True* zu. Überall dort, wo Sie eine *On Error GoTo*-Anweisung verwenden möchten, geben Sie nun die folgende Programmzeile an:

```
If cDebugModus = False Then On Error GoTo <Sprungmarke>
```

Die *On Error GoTo*-Anweisung ist damit nur dann aktiv, wenn Sie den Test beendet haben und der Konstanten *cDebugModus* den Wert *False* zuweisen.



Unerwartete Laufzeitfehler

Unerwartete Laufzeitfehler sind Laufzeitfehler, die in einer Prozedur ohne Fehlerbehandlungsroutine auftreten. Anstatt die Ausführung der Prozedur beim Auftreten eines Laufzeitfehlers zu unterbrechen, durchsucht Access zunächst die aufrufende Prozedur nach einer Fehlerbehandlungsroutine. Wird Access fündig, führt es die hier vorhandene Fehlerbehandlungsroutine aus. Sollte die Suche hier ebenfalls erfolglos sein, durchsucht Access wiederum die nächste aufrufende Prozedur usw. Erst wenn Access in diesem Aufrufpfad keine Fehlerbehandlungsroutine findet, wird ein Laufzeitfehler ausgegeben.

Beim Auftreten eines Fehlers in einer Prozedur kann es also vorkommen, dass der Fehler in einer anderen Prozedur als derjenigen Prozedur behandelt wird, in welcher der Fehler auftrat. Wird nun die *Resume*-Anweisung verwendet, stellt sich die Frage, wo Access mit der Ausführung des Programmcodes fortsetzt.

Die Anweisung *Resume [o]* bewirkt in diesem Fall, dass die komplette Prozedur, die den Fehler an die aktuelle Prozedur weitergibt, neu gestartet wird. Dies ist nur dann sinnvoll, wenn die Ursache des Fehlers in der Fehlerbehandlungsroutine behoben werden konnte. Die *Resume Next*-Anweisung bewirkt dagegen, dass Access die komplette Prozedur überspringt.

Fehlersuche

Zum Aufspüren von Fehlern stellt Ihnen Access mit dem Debugger ein leistungsfähiges Werkzeug zur Verfügung. In einem Programm können drei verschiedene Arten von Fehlern auftreten: Kompilierfehler, Laufzeitfehler und Fehler in der Programmlogik.

Kompilierfehler entstehen, sobald Access den Programmcode in ein für den Computer verständliches Format übersetzt. Access überprüft hierbei, ob die Syntax aller Befehle und Ausdrücke richtig ist. Bei einem Kompilierfehler werden Sie auf die fehlerhafte Stelle hingewiesen.

○ —————
Kompilierfehler

Laufzeitfehler sind Fehler, die sich erst bei der Programmausführung bemerkbar machen. Ein Laufzeitfehler kann beispielsweise dann auftreten, wenn Sie eine Division durch *NULL* vornehmen. Auch bei Laufzeitfehlern werden Sie auf die fehlerhafte Stelle hingewiesen.

○ —————
Laufzeitfehler

Die letzte Art von Fehlern sind Fehler in der Programmlogik. Diese Fehler sind schwer zu lokalisieren, da die genaue Ursache oftmals nicht bekannt ist und sich die Fehler an den verschiedensten Stellen auswirken können. Ein typisches Beispiel ist eine falsche Rechenformel, deren falsches Ergebnis sich über mehrere Schritte hinweg auswirkt. Die Aufgabe des Debuggers ist es, solche Fehler zu lokalisieren.

○ —————
Logische Fehler





Der Unterbrechungsmodus

Access aktiviert den Unterbrechungsmodus, sobald innerhalb einer gestarteten Prozedur ein Laufzeitfehler auftritt. Das Programm zeigt in diesem Fall ein Dialogfenster mit einer entsprechenden Fehlermeldung an.

Abb. 3.23

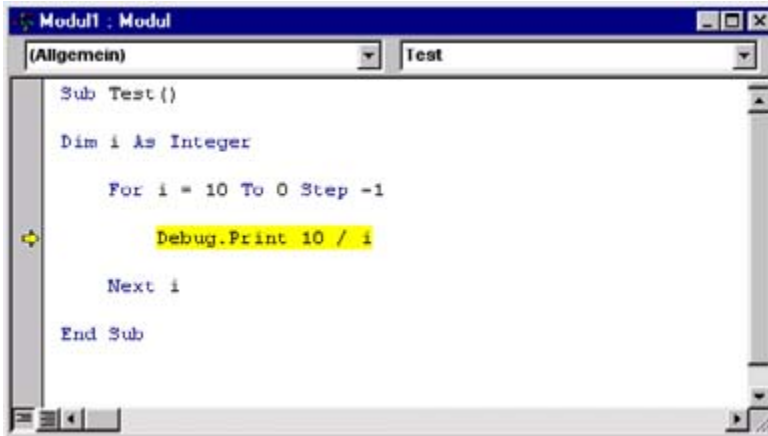
Eine Fehlermeldung



Das Dialogfenster enthält eine Schaltfläche mit der Beschriftung *Beenden*, die die Programmausführung vollständig beendet. Um allerdings den Unterbrechungsmodus zu aktivieren, klicken Sie auf die Schaltfläche *Testen*. Access aktiviert daraufhin den Unterbrechungsmodus und markiert die fehlerhafte Programmzeile im Modulfenster mit einem gelben Pfeil sowie einem gelben Hintergrund.

Abb. 3.24

Die Markierung der fehlerhaften Programmzeile



Wenn Ihnen die Ursache des Fehlers bereits bekannt ist, können Sie die fehlerhafte Programmzeile direkt korrigieren. Nachdem dies erledigt ist, setzen Sie die Programmausführung fort, indem Sie den Menüpunkt *Ausführen* > *Weiter* auswählen oder das Symbol *Weiter* in der Symbolleiste anklicken. Access wiederholt hierbei die Anweisung, die den Fehler verursacht hat. In der Regel setzt Access die Programmausführung also nahtlos fort.



Alternativ haben Sie auch die Möglichkeit, die komplette Programmausführung mit *Ausführen* > *Zurücksetzen* zu beenden. Hierbei löscht Access die Inhalte aller Variablen. Für diese Funktion steht in der Symbolleiste auch ein entsprechendes Symbol zur Verfügung.



Haltepunkte

Oft lässt sich die Ursache eines Fehlers nur dann genau lokalisieren, wenn Sie die Abarbeitung eines Programms Schritt für Schritt durchführen und jede Anweisung überprüfen. Dies ist besonders im Zusammenhang mit logischen Fehlern eine zeitraubende, aber oftmals notwendige Vorgehensweise.

Bei logischen Fehlern ist es wichtig, den Unterbrechungsmodus an einer ganz bestimmten Programmzeile zu aktivieren. Dies bewerkstelligen Sie mit den so genannten *Haltepunkten*, die Sie innerhalb einer Prozedur setzen.

Um einen Haltepunkt an einer bestimmten Programmzeile zu aktivieren, positionieren Sie die Einfügemarke auf diese Zeile und öffnen das Kontextmenü. Hier wählen Sie den Befehl *Umschalten* > *Haltepunkt* aus. Alternativ klicken Sie einmal in die graue Kennzeichenleiste am linken Rand des Modulfensters oder auf das Symbol *Haltepunkt ein/aus* in der Symbolleiste. Das Menü *Testen* stellt ebenfalls einen entsprechenden Befehl zur Verfügung.



Setzen von
Haltepunkten



Access kennzeichnet den Haltepunkt durch eine rote Hintergrundfarbe sowie einen roten Punkt in der Kennzeichenleiste.

Wenn Access nun während der Programmausführung auf diesen Haltepunkt stößt, stoppt die Programmausführung noch vor der Abarbeitung der ersten Anweisung in dieser Zeile. Außerdem aktiviert Access den Unterbrechungsmodus und Sie können mit der Fehlersuche beginnen.

Innerhalb einer Datenbank dürfen Sie beliebig viele Haltepunkte setzen. Es gibt jedoch Programmzeilen, in denen keine Haltepunkte zulässig sind. Dazu gehören beispielsweise Zeilen mit Variablendeklarationen oder Kommentarzeilen.

Um einen Haltepunkt wieder zu entfernen, positionieren Sie die Einfügemarke auf die betreffende Programmzeile und wählen den Menüpunkt *Umschalten* > *Haltepunkt* erneut aus. Ein Mausklick auf den roten Punkt der Kennzeichenleiste oder auf das Symbol *Haltepunkt ein/aus* führt zum gleichen Ziel. Der Menüpunkt *Debuggen* > *Alle Haltepunkte löschen* entfernt alle Haltepunkte Ihrer Datenbank in einem Rutsch.



Haltepunkte
deaktivieren

Die Haltepunkte eines Moduls werden nicht automatisch gelöscht, wenn Sie das Modul schließen. Nur das Schließen der Datenbank entfernt alle Haltepunkte.



Hinweis





Schrittweise Programmausführung

Wenn Access den Unterbrechungsmodus durch einen Haltepunkt aktiviert, stoppt die Programmausführung vor der markierten Zeile. Über den Menüpunkt *Ausführen* > *Fortsetzen* setzen Sie die Programmausführung bis zum Ende bzw. bis zum nächsten Haltepunkt fort.



Soll Access lediglich die aktuelle Anweisung ausführen, klicken Sie auf das Symbol *Einzelschritt* in der Symbolleiste oder wählen den gleichnamigen Befehl aus dem Menü *Debuggen*. Nach der Abarbeitung der aktuellen Anweisung aktiviert Access wieder den Unterbrechungsmodus und markiert im Modulfenster die nächste Anweisung. Sie können so alle Anweisungen der Prozedur schrittweise ausführen.

Prozedurschritt



Sollte der Aufruf einer Prozedur markiert sein, springt Access beim nächsten Einzelschritt zur ersten Anweisung dieser Prozedur. Wenn Sie die Prozedur als Ganzes ausführen lassen möchten, klicken Sie einfach auf das Symbol *Prozedurschritt* in der Symbolleiste. Access springt in diesem Fall nicht zu der ersten Anweisung dieser Prozedur, sondern führt diese komplett aus und markiert die nächste auszuführende Anweisung der aktuellen Prozedur.

Nächste Anweisung festlegen

Access führt während der Einzelschrittausführung eines Programms stets die momentan gelb markierte Programmzeile aus. Das Programm bietet Ihnen jedoch die Möglichkeit, die Programmausführung an einem anderen Punkt fortzusetzen. Hierzu bewegen Sie die Einfügemarke auf die Anweisung der aktuellen Prozedur, an der die Programmausführung beim nächsten Schritt fortgesetzt werden soll, und wählen aus dem Kontextmenü oder dem Menü *Testen* den Befehl *Nächste Anweisung festlegen* aus. Alternativ können Sie auch den gelben Pfeil in der Kennzeichenleiste via Drag & Drop verschieben.

Hinweis



Beachten Sie, dass sich die nächste Anweisung auf jeden Fall in der aktuellen Prozedur befinden muss!

Wenn Sie während der Fehlersuche im Unterbrechungsmodus zu anderen Prozeduren wechseln, wissen Sie unter Umständen nicht mehr, an welchem Punkt das Programm unterbrochen wurde. Wählen Sie dann einfach den Menüpunkt *Nächste Anweisung anzeigen* aus dem Kontextmenü oder dem Menü *Debuggen* aus. Access springt daraufhin wieder zu der Anweisung, die als Nächstes ausgeführt wird.

Hinweis



Bei der Fehlersuche ist es oftmals wichtig zu wissen, von welcher Prozedur aus die aktuellen Prozeduren aufgerufen wurde. Eine Übersicht erhalten Sie im Dialogfenster *Aufrufeliste*, das Sie über den gleichnamigen Befehl des Menüs *Ansicht* öffnen. Alternativ klicken Sie auf die Editor-Schaltfläche im rechten oberen Bereich des Lokalfensters.



Das Direktfenster

Das wichtigste Element des Debuggers ist das Direktfenster. Über das Direktfenster können Sie direkt Einfluss auf den Programmcode aller Module nehmen. Dazu gehört beispielsweise das Starten bestimmter Prozeduren zu Testzwecken und das Abfragen und Bestimmen von Variableninhalten oder Eigenschaften.



Abb. 3.25

Das Direktfenster

Das Direktfenster aktivieren Sie über den Menüpunkt *Ansicht > Direktfenster* oder einen Klick auf das gleichnamige Symbol in der Symbolleiste.

In den Direktbereich können Sie jede beliebige VBA-Anweisung eingeben. Die Anweisung wird ausgeführt, sobald Sie am Ende der Zeile die -Taste drücken.

Über die Pfeiltasten bewegen Sie die Einfügemarke zu einer bereits eingegebenen Anweisung, um diese beispielsweise zu editieren oder erneut auszuführen. Die Zwischenablagefunktionen können Sie ebenfalls wie gewohnt nutzen.

- Öffnen des Direktfensters
- Der Direktbereich



Aufruf einer Prozedur

Vom Direktbereich aus können Sie auch eine einzelne Prozedur zu Testzwecken starten. Die Eingabe

Testprozedur

würde beispielsweise die *Sub*-Prozedur mit dem Namen *Testprozedur* starten, sobald Sie die -Taste am Ende der Zeile drücken. Existiert die Prozedur nicht, zeigt Access eine Fehlermeldung an.

Noch komfortabler starten Sie eine Prozedur, indem Sie die Einfügemarke auf eine beliebige Anweisung dieser Prozedur positionieren und anschließend den Menüpunkt *Ausführen > Fortsetzen* auswählen. Dies funktioniert allerdings nur bei parameterlosen Prozeduren.



Hinweis

Zum Test von *Function*-Prozeduren sollten Sie die *Print*-Anweisung verwenden, die das Ergebnis der Funktion im Direktfenster anzeigt. Erwartet eine Prozedur Parameter, können Sie diese natürlich auf die gewohnte Art und Weise mit angeben.

- Print-Methode



Wenn Sie eine private Prozedur aufrufen möchten, die sich in einem Standardmodul befindet, geben Sie den Namen der Prozedur zusammen mit einem Verweis auf das Modul an. Das folgende Beispiel führt die private *Sub*-Prozedur *MeldungAnzeigen* im Modul *Modul1* aus:

```
Modul1.MeldungAnzeigen
```

Das Ausführen einer Prozedur in einem Objektmodul – also einem Formular- oder Berichtsmodul – ist nur bei öffentlichen Prozeduren möglich. Zusätzlich ist es erforderlich, dass das betreffende Formular oder der Bericht geöffnet sein muss. Zur Ausführung geben Sie den Namen der Prozedur inklusive des vollständigen Objektverweises (z. B. *Forms!Kunden.KundenNummerAnzeigen*) an.

Ermitteln von Variableninhalten

Über die *Print*-Methode können Sie sich auch Inhalte von Variablen sowie von Eigenschaften verschiedener Objekte ausgeben lassen. Das erste der beiden folgenden Beispiele ermittelt den aktuellen Inhalt der Variablen *X*, das zweite den Inhalt des Steuerelements *Name* auf dem Formular *Angestellte*. Das entsprechende Formular muss dazu geöffnet sein:

```
Print X  
Print Forms![Angestellte]![Name]
```

Hinweis

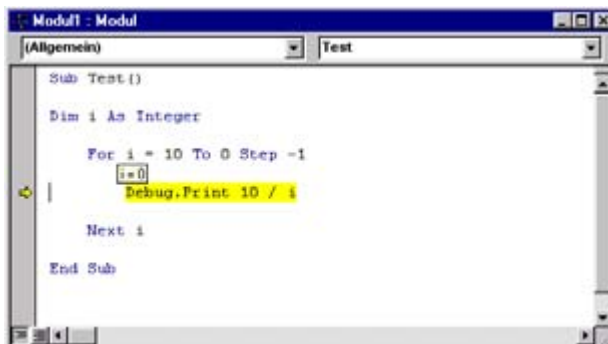


Noch einfacher werten Sie den Inhalt einer Variablen aus, indem Sie einfach den Mauszeiger darauf positionieren. Access zeigt daraufhin den aktuellen Inhalt in Form eines Tipptextes an. Voraussetzung dafür ist lediglich, dass sich Access im Unterbrechungsmodus befindet. Im Register *Editor* des *Optionen*-Dialogs haben Sie über die Option *Automatische Daten-Tips* die Möglichkeit, diese Funktion bei Bedarf zu deaktivieren.

Alternativ positionieren Sie die Einfügemarke auf die gewünschte Variable und wählen den Menüpunkt *Debuggen > Aktuellen Wert anzeigen* (+**F9**) aus. Diese Vorgehensweise ist besonders dann effektiv, wenn Sie vorzugsweise mit Ihrer Tastatur arbeiten.

Abb. 3.26

Anzeigen eines
Variableninhalts mit
Daten-Tipps



Über das Direktfenster können Sie jedoch nicht nur die Inhalte verschiedener Variablen oder Eigenschaften auslesen, sondern diese auch verändern. Hierzu geben Sie im Direktfenster auf die gleiche Art und Weise wie im Programmcode eine Zuweisung an. Das entsprechende Formular im zweiten Beispiel muss auch hier geöffnet sein:

```
X = 10
Forms![Angestellte]![Name] = "Meier"
```

Zuweisen von Werten

Zugriff auf den Direktbereich mit VBA

Auf den Direktbereich des Direktfensters können Sie auch vom Programmcode aus zugreifen. Der Direktbereich wird hierbei durch ein spezielles Objekt namens *Debug* vertreten. Das *Debug*-Objekt besitzt nur die eine Methode *Print*. Die Anweisung

```
Debug.Print "Das ist ein Test"
```

hätte beispielsweise die Ausgabe "Das ist ein Test" im Direktbereich zur Folge. Das Direktfenster braucht hierzu nicht geöffnet zu sein.

Das Lokal-Fenster

Das Lokal-Fenster informiert Sie über die Inhalte aller lokalen und globalen Variablen des aktuellen Moduls. Um dieses Fenster anzuzeigen, wählen Sie im *Ansicht*-Menü den Eintrag *Lokal-Fenster*.

Sobald der Unterbrechungsmodus aktiviert ist, erscheint im Lokal-Fenster eine Strukturansicht ähnlich dem Verzeichnisbaum Ihrer Festplatte. Der erste Knoten trägt stets den Namen des aktuellen Moduls, in dem der Unterbrechungsmodus aktiviert wurde. Eine Ausnahme bilden lediglich Klassenmodule, deren Knoten den Namen *Me* haben. Diesem Knoten folgen die einzelnen Variablen der aktuellen Prozedur.



Abb. 3.27

Der Lokalbereich im Einsatz



Die drei Spalten des Lokalbereichs haben folgende Bedeutungen:

Spalten des Lokalbereichs

Spalte	Bedeutung
<i>Ausdruck</i>	Der Name einer Variablen, eines Objekts oder eines anderen Elements des aktuellen Moduls bzw. der aktuellen Prozedur.
<i>Wert</i>	Der Wert einer Variablen bzw. einer Eigenschaft.
<i>Typ</i>	Der Typ einer Variablen oder eines Objekts.

Tabelle 3.37

Wie bereits erwähnt, repräsentiert der erste Knoten des Lokalbereichs stets das aktuelle Modul. Direkt neben dem Knoten zeigt Access ein +-Symbol an, mit dem Sie die nächste Hierarchieebene einblenden. Hier listet Access alle privaten und öffentlichen Variablen auf, die im Deklarationsbereich des Moduls deklariert sind.

Auswertung von Objekten

Neben normalen Variablen zeigt Access im Lokalbereich auch Objektvariablen an, die ebenfalls ein +-Symbol besitzen. Ein Beispiel für eine solche Variable ist beispielsweise die Objektvariable namens *MeineTabelle*, mit dem Datentyp *TableDef*. Wie Sie im *Kapitel 5 Datenzugriff mit VBA* noch erfahren werden, repräsentiert eine solche Variable eine Tabelle Ihrer Datenbank.

Wenn Sie die Hierarchie einer solchen Objektvariablen mittels +-Symbol erweitern, zeigt Access alle Eigenschaften des Objekts und deren Inhalte in einer Liste an. Sie erhalten somit einen Überblick über den aktuellen Status Ihres Objekts.

Sollte das Objekt Auflistungen enthalten, erscheinen diese ebenfalls mit einem +-Symbol. Ein *TableDef*-Objekt enthält beispielsweise mehrere *Field*-Objekte, die die einzelnen Datenfelder der Tabelle verkörpern. Diese *Field*-Objekte sind in einer Auflistung namens *Fields* angeordnet.

Mit einem Klick auf das +-Symbol dieser Auflistung zeigt Access die einzelnen *Field*-Objekte an, deren Eigenschaften Sie nun ebenfalls begutachten können. Auf diese Weise hangeln Sie sich Schritt für Schritt durch die einzelnen Auflistungen des Objekts und können so den genauen Status eines Objekts und damit Ihrer Prozedur feststellen.

Während der Unterbrechungsmodus aktiv ist, können Sie über das Überwachungsfenster den Inhalt von Variablen sowohl auslesen als auch verändern. Um die Inhalte mehrerer Variablen bzw. das Ergebnis von Ausdrücken während der Fehlersuche auf einfache Weise zu kontrollieren, können Sie diese automatisch überwachen lassen. Markieren Sie dazu im Modulfenster den gewünschten Ausdruck



und wählen Sie den Befehl *Überwachung hinzufügen* aus dem Kontextmenü oder dem Menü *Debuggen* aus. Access öffnet daraufhin das Dialogfenster *Überwachung hinzufügen*.

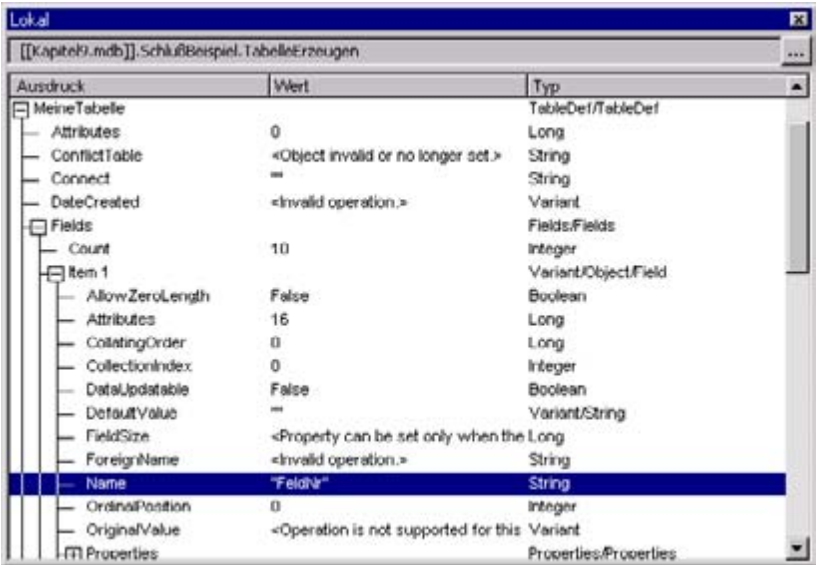


Abb. 3.28

Überwachen von Variablen und Ausdrücken

Abb. 3.29

Das Dialogfenster *Überwachung hinzufügen*



Im Feld *Ausdruck* hat Access bereits den markierten Ausdruck eingetragen, der überwacht werden soll. Unter *Kontext* legen Sie den Sichtbarkeitsbereich fest, in dem der zu überwachende Ausdruck vorhanden ist. Dies ist wichtig, falls der Ausdruck Variablen enthält, die in verschiedenen Sichtbarkeitsbereichen den gleichen Namen haben. Standardmäßig trägt Access das aktuelle Modul und die aktuelle Prozedur ein.

Überwachungsarten

In der untersten Sektion bestimmen Sie die Art der Überwachung. Die Einstellung *Überwachungsausdruck* bewirkt, dass Access lediglich das Ergebnis des eingegebenen Ausdrucks zur Begutachtung angezeigt. Dies ist beispielsweise dann sinnvoll, wenn Sie eine Prozedur schrittweise im Unterbrechungsmodus ausführen und das Ergebnis des Ausdrucks nach jedem Schritt kontrollieren möchten.

Die Option *Unterbrechen, wenn der Wert geändert wurde*, wird dagegen nicht im Unterbrechungsmodus, sondern bei der normalen Ausführung eines Programms eingesetzt. Access überprüft bei dieser Einstellung den angegebenen Ausdruck nach der Ausführung jeder einzelnen Anweisung. Sollte sich das Ergebnis ändern, aktiviert Access den Unterbrechungsmodus und zeigt die Anweisung, die die Änderung verursachte, im Modulfenster an.

Eine weitere Option mit dem Namen *Unterbrechen, wenn der Wert True ist* aktiviert den Unterbrechungsmodus, sobald der angegebene Ausdruck den logischen Wert *True* zum Ergebnis hat. Wenn Sie beispielsweise eine Prozedur unterbrechen möchten, sobald die Variable *Zähler* den Wert *10* aufweist, markieren Sie diese Option und geben unter *Ausdruck* Folgendes an:

Zähler = 10

Nachdem Sie das Dialogfenster mit *OK* geschlossen haben, öffnet Access das Überwachungsfenster und fügt den Ausdruck dem Überwachungsbereich hinzu.

Abb. 3.30

Das Überwachungsfenster



Im Überwachungsfenster listet Access alle Ausdrücke auf, die Sie mithilfe des Dialogfensters *Überwachung hinzufügen* hinzugefügt haben. Wenn Sie eine Überwachung bearbeiten möchten, markieren Sie sie und wählen den Menüpunkt *Überwachung bearbeiten* aus dem Kontextmenü oder dem Menü *Debuggen* aus. Hier finden Sie außerdem zwei Menüpunkte zur Bearbeitung und zum Löschen einer Überwachung.

Arbeiten mit Lesezeichen

Bei der Fehlersuche ist es häufig erforderlich, zwischen den einzelnen Modulen und Prozeduren Ihrer Datenbank zu wechseln, um der Ursache des Problems auf den Grund zu gehen.

Hierbei unterstützen Sie Lesezeichen. Ein Lesezeichen ist nichts anderes als eine Markierung einer bestimmten Programmzeile in einem beliebigen Modul. Sie setzen ein Lesezeichen, indem Sie die Einfügemarke auf die gewünschte Programmzeile platzieren und den Menüpunkt *Bearbeiten > Lesezeichen > Lesezeichen setzen > zurücksetzen* auswählen. Access markiert das auf diese Weise gesetzte Lesezeichen in der Kennzeichenleiste mit einem hellblauen Rechteck.

Wenn Sie mehrere Lesezeichen gesetzt haben, wechseln Sie mit den beiden Befehlen *Nächstes Lesezeichen* und *Vorheriges Lesezeichen* zwischen den einzelnen Lesezeichen Ihrer Datenbank. Sie finden diese beiden Befehle ebenfalls im Menü *Bearbeiten > Lesezeichen*.

Ein weiterer Befehl namens *Alle Lesezeichen löschen* entfernt die Lesezeichen aus allen Modulen Ihrer Datenbank.



Abb. 3.31

Ein Lesezeichen

Kompilieren des Programmcodes

Bevor Access ein Programm ausführen kann, muss es kompiliert werden. Bei der Kompilierung übersetzt Access den Programmcode in ein Format, das von Access ausgeführt werden kann. Von diesem Vorgang merken Sie als Anwender kaum etwas, da Access die Kompilierung automatisch vornimmt, wenn Sie eine Prozedur des Programms ausführen. Über den Menüpunkt *Debuggen > Kompilieren von* kön-





nen Sie die Kompilierung auch starten, ohne eine Prozedur auszuführen. Dies wird in der Regel dazu genutzt, um den Programmcode auf syntaktische Fehler zu untersuchen.

Der Menüpunkt *Kompilieren von* berücksichtigt bei der Kompilierung nur Module, die momentan geladen sind. Dazu zählen beispielsweise nicht die Objektmodule von Formularen und Berichten, die geschlossen sind.

Hinweis



Bei der automatischen Kompilierung werden nur die Teile des Programms kompiliert, die zur Ausführung benötigt werden. Die automatische Kompilierung kann im Register *Allgemein* des *Optionen*-Dialog unter *Kompilierung* > *bei Bedarf* deaktiviert werden.

Speichern der kompilierten Datenbank



Um das Öffnen und die Arbeitsgeschwindigkeit einer Datenbank zu beschleunigen, sollten Sie diese vorher im kompilierten Zustand speichern. Dazu öffnen Sie die Datenbank exklusiv, führen den Menübefehl *Kompilieren von* aus und speichern die Datenbank wieder.

Der kompilierte Status Ihrer Datenbank geht verloren, sobald Sie Änderungen an den Datenbankobjekten vornehmen (z. B. wenn Sie ein Modul ergänzen oder ein Formular löschen) oder die Datenbank umbenennen. Sie sollten also anschließend eine erneute Kompilierung der kompletten Datenbank vornehmen.

Hinweis



Mit der *IsCompiled*-Eigenschaft des *Application*-Objekts können Sie feststellen, ob die komplette Datenbank kompiliert ist. Die Eigenschaft hat in diesem Fall den Wert *True*.

Compilerdirektiven

Mithilfe von Compilerdirektiven ist es möglich, nur bestimmte Programmzeilen kompilieren zu lassen. Diese Möglichkeit wird Access von dem OLE-Server VBA zur Verfügung gestellt, der auch in Visual Basic 4.0 integriert ist. Da es von Visual Basic 4.0 sowohl eine 16- also auch eine 32-Bit-Version gibt, ist es in einigen Fällen wichtig, dass bestimmter Programmcode unter beiden Versionen ausgeführt werden kann. Probleme bereiten hier vor allem API-Deklarationen, die mithilfe der *Declare*-Anweisung eingebunden werden. Da sich die API-Funktionen von Windows 3.x und Windows 95/NT (Win32) in vielen Fällen unterscheiden, kann mithilfe von Compilerdirektiven je nach Betriebssystem die passende API-Deklaration verwendet werden.

Bei Access XP tritt dieses Problem nicht auf, da hier ausschließlich eine 32-Bit-Version existiert. Compilerdirektiven können aber eingesetzt werden, um speziellen Test-Programmcode (auch als *Debug-Code* bezeichnet) nur während der Entwicklung einer Datenbank zu aktivieren. Zu diesem Zweck stellt Access folgendes Konstrukt zur Verfügung:





```
#If <Ausdruck> Then

    <Anweisungen>

[#ElseIf <Ausdruck> Then

    [<elseifAnweisungen>]]

[#Else

    [<elseAnweisungen>]]

#End If
```

Dieses Konstrukt gleicht der *If...Then...Else*-Anweisung, die Sie bereits in den vorhergehenden Abschnitten kennen gelernt haben. Der einzige Unterschied besteht darin, dass die einzeilige *If...Then...Else*-Variante nicht zulässig ist.

Die einfachste Variante des obigen Konstrukts sieht folgendermaßen aus:

```
#If <Ausdruck> Then

    <Anweisungen>

#End If
```

Unter *<Anweisungen>* können Sie einen beliebigen Programmcode angeben, den Access bei der Kompilierung nur dann berücksichtigt, wenn der *<Ausdruck>* den Wert *True* zum Ergebnis hat. Unter *<Ausdruck>* wird in der Regel der Name einer Konstanten angegeben. Es dürfen jedoch keine "normalen" Konstanten, sondern lediglich Compilerkonstanten verwendet werden. Solche Konstanten werden mithilfe der Anweisung *#Const* definiert:

```
#Const <Konstantenname> = <Ausdruck>
```

Der Datentyp der Konstanten wird hierbei genau wie bei normalen Konstanten von Access automatisch ermittelt.

Obwohl Sie die obige Anweisung innerhalb einer Prozedur angeben können, wird sie immer auf Modulebene ausgewertet. Sie sollten die Anweisung daher im Deklarationsteil eines Moduls platzieren. Außerdem sind Compilerkonstanten, die Sie über die *#Const*-Anweisung definieren, immer privat und gelten damit nur für das aktuelle Modul.

Das folgende Beispiel definiert eine Konstante mit dem Namen *kompDebugModus*:

```
#Const kompDebugModus = True
```





Den Inhalt dieser Konstanten können Sie nun auswerten, um im *True*-Fall speziellen Debug-Code auszuführen:

```
Sub MeldungAusgeben()  
  
    #If kompDebugModus = True Then  
  
        Debug.Print "Prozedur "MeldungAusgeben" gestartet!"  
  
    #End If  
  
    MsgBox "Dies ist eine Meldung."  
  
End Sub
```

Wenn Sie den Test Ihres Programms beendet haben, setzen Sie den Wert der Konstanten einfach auf den Wert *False*. Der Debug-Code wird dann weder kompiliert noch ausgeführt.

Globale Kompilerkonstanten

Kompilerkonstanten können nicht nur über die *#Const*-Anweisung, sondern auch über den *Optionen*-Dialog (Menüpunkt *Extras > Optionen*) von Access definiert werden. Im Register *Weitere* finden Sie ein Textfeld mit der Beschriftung *Bedingte Kompilierungsargumente*. In dieses Textfeld können Sie direkt die Definition einer Konstanten eingeben:

```
kompDebugModus = -1
```

Beachten Sie bei der Eingabe von booleschen Werten, daß Sie hier nicht die Schlüsselwörter *True* und *False* verwenden können. Statt dessen müssen Sie einen der beiden Werte -1 (für *True*) und 0 (für *False*) angeben. Wenn Sie mehrere Konstanten definieren möchten, unterteilen Sie die einzelnen Definitionen durch einen Doppelpunkt.

Die definierten Konstanten werden zusammen mit der aktuellen Datenbank gespeichert. Im Gegensatz zu Konstanten, die Sie über die *#Const*-Anweisung definieren, wirken sie sich datenbankweit für alle Module aus.

Vordefinierte Compilerkonstanten

Neben den Konstanten, die Sie mithilfe der *#Const*-Anweisung selbst definieren, stellt Access zwei vordefinierte Konstanten mit den Namen *Win16* und *Win32* zur Verfügung. Je nachdem, ob die aktuelle Anwendung eine 16- oder eine 32-Bit-Anwendung ist, hat entweder die Konstante *Win16* oder *Win32* den Wert *True*. Da Access XP eine 32-Bit-Anwendung ist, enthält *Win16* stets den Wert *False* und *Win32* den Wert *True*. Das folgende Beispiel gibt also den Text "Die Anwendung ist eine 32-Bit-Anwendung" aus:



```
Sub KompilerKonstanten()  
  
    #If Win32 Then  
  
        MsgBox "Die Anwendung ist eine 32-Bit-Anwendung"  
  
    #End If  
  
    #If Win16 Then  
  
        MsgBox "Die Anwendung ist eine 16-Bit-Anwendung"  
  
    #End If  
  
End Sub
```

Datenbankverweise

Wenn Sie regelmäßig Datenbanken entwickeln und hierbei VBA-Prozeduren einsetzen, werden Sie wahrscheinlich nach einiger Zeit Prozeduren entwickelt haben, die allgemeine Aufgaben erledigen. Solche Prozeduren eignen sich besonders gut zum Zusammenstellen einer Prozedurbibliothek, auf die Sie bei der Entwicklung neuer Datenbanken zurückgreifen können. Ein Problem ist hierbei allerdings, dass Sie die entsprechenden Module, in denen sich die Prozeduren befinden, in jede einzelne Datenbank kopieren müssen. Wenn Sie nachträglich an einer bestimmten Prozedur eine Optimierung vornehmen oder einen Fehler beseitigen, müssen Sie die betreffenden Prozeduren in allen Datenbanken austauschen.

Um dieses Problem zu lösen, stellt Access die Möglichkeit zur Verfügung, eine oder mehrere Datenbanken in die aktuelle Datenbank einzubinden. Solche Datenbanken werden auch als *referenzierte Datenbanken* bezeichnet, da Sie eine Referenz zu dieser Datenbank anlegen. Alle Prozeduren, Variablen und Konstanten der referenzierten Datenbank können von der aktuellen Datenbank aus genutzt werden, als wenn sich die entsprechenden Module in der aktuellen Datenbank befänden. Die Datenbank, in welche die referenzierten Datenbanken eingebunden werden, wird in diesem Fall als *Hauptdatenbank* bezeichnet.

Die Möglichkeit zum Einbinden von Datenbanken eignet sich auch besonders gut für ein Entwicklerteam, in dem mehrere Entwickler an einer einzelnen Datenbank-anwendung arbeiten. Jeder Entwickler könnte eine eigene Datenbankdatei bearbeiten, die später mit den anderen Datenbanken zu einer Anwendung verknüpft wird.

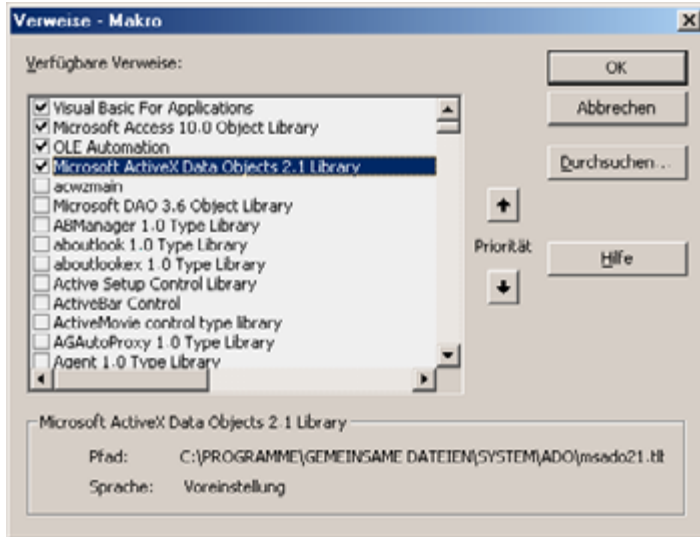
Herstellen eines Verweises

Zur Herstellung eines Verweises auf eine Datenbank stellt Access ein spezielles Dialogfenster mit dem Namen *Verweise* zur Verfügung, das Sie über den gleichnamigen Menüpunkt des Menüs *Extras* öffnen. Beachten Sie, dass dieser Menüpunkt nur im Modulfenster verfügbar ist.



Abb. 3.32

Das Dialogfenster
Verweise



OLE-Server

In dem Dialogfenster wird eine Liste *Verfügbare Verweise* mit mehreren Einträgen angezeigt. Bei diesen Einträgen handelt es sich fast ausschließlich um OLE-Server. Durch Markieren des Kontrollkästchens können Sie einen bestimmten Server aktivieren. Standardmäßig sind bei neuen Projekten bereits die folgenden Server aktiviert:

- Visual Basic For Applications
- Microsoft Access 10.0 Object Library
- OLE Automation
- Microsoft ActiveX Data Objects 2.1 Library

Wie Sie bereits in der Einleitung dieses Kapitels erfahren haben, machen diese OLE-Server (bzw. Objektbibliotheken) die Programmiersprache VBA aus. Neben diesen Einträgen enthält die Liste alle Objektbibliotheken, die auf Ihrem Rechner installiert sind. Wenn Sie beispielsweise das komplette Office-Paket von Microsoft installiert haben, finden Sie hier Einträge wie *Microsoft Excel 10.0 Objektbibliothek* oder *Microsoft Binder 10.0 Objektbibliothek*. Wenn Sie einen dieser Einträge mit einem Häkchen markieren, integriert Access die Objekte der Objektbibliothek in VBA. Wie Sie diese Objekte dann nutzen, erfahren Sie im *Kapitel 13 OLE und DDE*.

Einbinden von
Datenbanken

Neben Objektbibliotheken lassen sich über den *Verweise*-Dialog auch normale Datenbanken einbinden. Dazu klicken Sie auf die Schaltfläche *Durchsuchen*, um ein Dateiauswahlfenster zu öffnen.



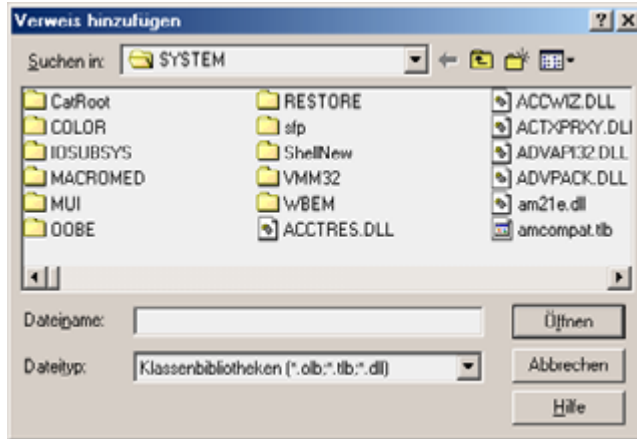


Abb. 3.33

Das Dateiauswahl-
fenster *Verweis
hinzufügen*

Da Sie eine Datenbank einbinden möchten, wählen Sie unter *Dateityp* den Eintrag *Microsoft Access-Datenbanken* aus. Anschließend suchen Sie in der Dateiliste nach der gewünschten Datenbank, markieren diese und klicken auf *OK*. Die neue Datenbank wird daraufhin der Verweiseliste hinzugefügt und automatisch mit einem Häkchen markiert. Sie ist damit erfolgreich eingebunden und kann nun genutzt werden.

Der Pfad zu der referenzierten Datenbank wird zusammen mit der aktuellen Datenbank gespeichert. Sie können also für jede Datenbank unterschiedliche Verweise definieren. Außerdem ist es zulässig, mehrere Datenbanken von einer Datenbank aus zu referenzieren. Auch gleichzeitige Verweise einer einzigen Datenbank von mehreren anderen Datenbanken aus sind zulässig.

Verschieben der referenzierten Datenbankdatei

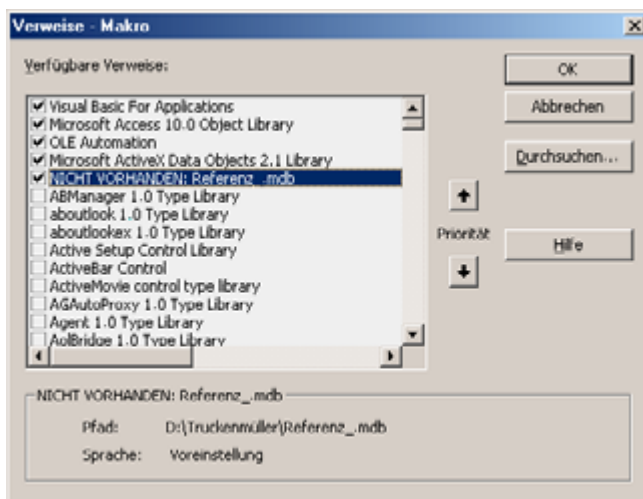
Während eine Datenbank von einer anderen Datenbank aus genutzt wird, ist es nicht möglich, diese zu verschieben, umzubenennen oder zu löschen. In allen Fällen meldet Windows einen entsprechenden Fehler. Dies gilt allerdings nur, solange die Hauptdatenbank – in die die referenzierte Datenbank eingebunden ist – auch geöffnet ist. Wenn Sie diese Datenbank dagegen schließen, können Sie alle oben genannten Änderungen an der Datei vornehmen.

Sollten Sie am Dateinamen oder am Pfad der referenzierten Datenbank Änderungen vorgenommen haben, kann Access sie beim nächsten Öffnen der Hauptdatenbank nicht mehr finden. Sobald beispielsweise auf eine Prozedur in einer referenzierten Datenbank zugegriffen wird, meldet Access einen Fehler und öffnet anschließend das Dialogfenster *Verweise*. Die nicht gefundene Datenbank wird mit dem Vermerk "NICHT VORHANDEN:" aufgeführt.



Abb. 3.34

Markierung fehlender
Referenzen



In diesem Fall müssen Sie die Verbindung "von Hand" wiederherstellen. Markieren Sie dazu die nicht gefundene Datenbank im Dialogfenster *Verweise* und klicken Sie auf die Schaltfläche *Durchsuchen*. Anschließend navigieren Sie zu der Datenbank und bestätigen mit *OK*. Vergessen Sie vorher nicht, unter *Dateityp* den Typ *Datenbanken* auszuwählen.

Wiederherstellen
der Verknüpfung

In einigen Fällen kann Access die verloren gegangene Verknüpfung automatisch wiederherstellen. Voraussetzung dafür ist, dass die Datenbank lediglich in das Verzeichnis *Windows* (bzw. das Windows-Installationsverzeichnis) oder *\Windows\System* verschoben und nicht etwa umbenannt oder gelöscht wurde. Wenn Access die Datenbank hier findet, wird die Verknüpfung ohne Anzeige einer Fehlermeldung wiederhergestellt. Diese Tatsache ist besonders wichtig, wenn Sie eine Datenbank zusammen mit einer referenzierten Datenbank an einen Kunden weitergeben. Hierbei bietet es sich an, alle referenzierten Datenbanken beim Kunden in das Windows- oder das Systemverzeichnis zu installieren. So ist sichergestellt, dass Access die Datenbanken stets finden kann.

Zugriff auf eine referenzierte Datenbank

Nachdem Sie eine Datenbank über den *Verweise*-Dialog unter der aktuellen Datenbank angemeldet haben, können Sie auf alle öffentlich deklarierten Elemente der referenzierten Datenbank zugreifen, die sich innerhalb von Standard- oder Klassenmodulen befinden. Dazu gehören öffentliche *Sub*- und *Function*-Prozeduren, Konstanten, Variablen und benutzerdefinierte Datentypen.

Der Zugriff auf diese Elemente erfolgt hierbei so, als wenn die entsprechenden Module in die aktuelle Datenbank importiert worden wären. Wenn sich beispielsweise in einem Modul mit dem Namen *Modul A* eine öffentliche Prozedur mit dem Namen *Testprozedur* befindet, können Sie sie von der Hauptdatenbank wie



gewohnt durch einfache Angabe ihres Namens aufrufen. Wenn es sich bei der Testprozedur um eine parameterlose *Sub*-Prozedur handelt, könnte dies folgendermaßen aussehen:

Testprozedur

Der Zugriff auf Formulare, Berichte oder andere Objekte der referenzierten Datenbanken ist von der Hauptdatenbank aus nicht möglich. Sie können jedoch innerhalb der referenzierten Datenbanken Prozeduren anlegen, die dann Zugriff auf alle Objekte dieser Datenbank haben.



Hinweis

Wie bei mehreren gleichnamigen öffentlichen Prozeduren in einer einzigen Datenbank können auch zwischen der Hauptdatenbank und einer referenzierten Datenbank Namenskonflikte auftreten, wenn zwei oder mehrere Prozeduren den gleichen Namen haben. Um diesen Konflikt aufzulösen, muss zusammen mit dem Prozedurnamen der Name des Moduls angegeben werden, in dem sich die Prozedur befindet. Das folgende Beispiel startet die Prozedur in dem *Modul A*:



Namenskonflikte

```
[Modul A].Testprozedur
```

Im ungünstigsten Fall kann es nun passieren, dass auch der Modulname nicht eindeutig ist. In diesem Fall muss der Name der Datenbank ohne die Endung *.mdb* angegeben werden. Der Pfad der Datenbank kann hierbei ausgelassen werden, da die Position der referenzierten Datenbank bereits über den *Verweise*-Dialog festgelegt wurde. Das folgende Beispiel startet die Testprozedur, die sich im *Modul A* der Datenbank *Bibliothek.mdb* befindet:

```
Bibliothek.[Modul A].Testprozedur
```

Die Auflösung von Namenskonflikten betrifft natürlich nicht nur Prozeduren, sondern auch Variable, Konstanten und benutzerdefinierte Typen. Auch hier muss ggf. der Modulname oder sogar der Datenbankname mit angegeben werden.

In einigen Spezialfällen ist es wünschenswert, dass die öffentlichen Elemente eines Moduls nur von der aktuellen Datenbank aus genutzt werden sollen. Wenn diese Datenbank dagegen als Verweis in eine andere Datenbank eingebunden wird, soll kein Zugriff möglich sein. Um dies zu bewerkstelligen, fügen Sie dem betreffenden Modul einfach die folgende Anweisung hinzu:



Hinweis

```
Option Private Module
```

Alle öffentlichen Elemente sind dann nur noch im aktuellen Projekt und nicht projektübergreifend nutzbar.

Namenskonflikte zwischen referenzierten Datenbanken

Wenn Sie in die Hauptdatenbank mehrere Datenbanken einbinden, kann es zwischen den öffentlichen Elementen dieser Datenbanken ebenfalls zu Namenskonflikten kommen. Diese werden von Access jedoch auf andere Weise gelöst.





Prioritäten

Bei der Lösung von Namenskonflikten in referenzierten Datenbanken kommt es darauf an, in welcher Reihenfolge die einzelnen Einträge im Dialog *Verweise* angeordnet sind. Wenn sich beispielsweise in zwei referenzierten Datenbanken eine öffentliche Prozedur mit dem Namen *Testprozedur* befindet und diese von der Hauptdatenbank ohne Angabe des Modul- oder Datenbanknamens aufgerufen wird, durchsucht Access die Liste von oben nach unten nach einer passenden Prozedur. Anschließend wird die Prozedur derjenigen Datenbank ausgeführt, in der Access zuerst fündig würde.

Um einem Eintrag im Dialog *Verweise* eine andere Priorität zuzuordnen, markieren Sie ihn einfach und verschieben ihn mithilfe der Pfeiltasten *Priorität*. Je weiter oben Sie eine Referenz anordnen, desto höher ist die Priorität.

Der Grund dafür, warum Access Namenskonflikte unter referenzierten Datenbanken anders auflöst, wird anhand des folgenden Beispiels deutlich: Sie entwickeln eine Datenbank, in die Sie über den *Verweise*-Dialog eine Bibliotheksdatenbank mit Standardprozeduren eingebunden haben. Um Namenskonflikte von vornherein zu vermeiden, haben Sie allen öffentlichen Elementen eindeutige Namen zugewiesen. Im Laufe der Entwicklung erwerben Sie eine Datenbank, deren Funktionen Sie in Ihre Datenbank integrieren möchten. Sie legen also einen Verweis zu dieser Datenbank an. Zufälligerweise enthält die neue Datenbank eine Prozedur, die unter dem gleichen Namen bereits in Ihrer ersten referenzierten Datenbank enthalten ist. Bei der herkömmlichen Konfliktlösung müssten Sie nun alle bereits vorhandenen Aufrufe der Prozedur eindeutig machen, indem Sie diese um den Modul- und eventuelle auch um den Datenbanknamen ergänzen. Es wäre also eine komplette Überarbeitung Ihres Programmcodes nötig.

Mit der speziell auf referenzierte Datenbanken zugeschnittenen Konfliktlösung ist dies nicht erforderlich, da Access automatisch die erste passende Prozedur aufruft. Wenn also der erste Verweis eine höhere Priorität als der zweite Verweis hat, wird weiterhin Ihre selbsterstellte Prozedur ausgeführt. Wenn Sie dagegen die neue Prozedur ausführen möchten, müssen Sie zusätzlich den Modulnamen und – falls dieser nicht eindeutig ist – den Datenbanknamen angeben.

Bearbeiten einer referenzierten Datenbank

Eine referenzierte Datenbank wird von Access im nicht exklusiven Modus geöffnet. Sie können also eine referenzierte Datenbank ohne Probleme öffnen, obwohl sie gerade von einer anderen geöffneten Datenbank aus eingebunden ist. Wenn Sie an der referenzierten Datenbank allerdings Änderungen vornehmen, wirken sich diese in der Hauptdatenbank erst dann aus, wenn Sie die Hauptdatenbank schließen und wieder öffnen.

Wenn Sie lediglich den Programmcode einer referenzierten Datenbank bearbeiten möchten, stellt der Objektkatalog einen besseren Weg zur Verfügung: Öffnen Sie den Objektkatalog über den Menüpunkt *Ansicht > Objektkatalog*. In der Liste *Projekt > Bibliothek* werden alle aktiven Verweise der Datenbank aufgeführt. Wählen Sie hier die gewünschte Datenbank aus und markieren Sie in der Liste *Klassen* das gewünschte Modul. In der rechten Liste *Elemente von* werden nun alle Prozeduren



dieses Moduls aufgelistet. Wählen Sie hier einen Eintrag aus und klicken Sie auf *Anzeigen*. Access öffnet daraufhin das passende Modulfenster zur Bearbeitung und zeigt die Prozedur an.



Abb. 3:35

Öffnen einer Prozedur in einer referenzierten Datenbank mithilfe des Objektkatalogs

Über den Objektkatalog können nicht nur Module referenzierter Datenbanken, sondern auch Objektmodule und OLE-Objektbibliotheken analysiert werden. Mehr dazu erfahren Sie im Abschnitt *Der Objektkatalog* des folgenden *Kapitel 4 Programmieren mit Objekten*.



Hinweis

Ereignisorientierte Programmierung

Das Prinzip der ereignisorientierten Programmierung haben Sie bereits im *Kapitel 1 Einführung in die Makroprogrammierung* dieses Buches kennen gelernt. Demnach wird jede einzelne Prozedur direkt oder indirekt über ein Ereignis gestartet. Es gibt also kein durchgängiges Programm, das Access von oben nach unten ausführt.

Der Großteil der Ereignisse wird von Formularen und deren Steuerelementen ausgelöst. Bei der Entwicklung einer Datenbankanwendung wird daher in der Regel mit dem Anlegen der Formulare begonnen, die anschließend mit dem Ereigniscode verknüpft werden.

Es gibt verschiedene Möglichkeiten, ein Ereignis mit einer Prozedur zu verknüpfen. Die einfachste und gleichzeitig beste Lösung haben Sie im *Kapitel 2 Schnellstart VBA* kennen gelernt. Hierbei positionieren Sie die Einfügemarke in die gewünschte Ereigniseigenschaft des Eigenschaftenfensters und klicken auf die Editor-Schaltfläche rechts neben dem Eingabefeld. Das folgende Dialogfenster bestätigen Sie mit einem Doppelklick auf *Code-Editor*, wodurch Access das betreffende Formular- bzw. Berichtsmodul öffnet und eine leere Prozedur anzeigt.



Verknüpfen von Ereignissen und Prozeduren

Wenn bereits eine Ereignisprozedur zu dem betreffenden Ereignis existiert, öffnet Access beim Klick auf die Editor-Schaltfläche automatisch das Formular- bzw. Berichtsmodul und zeigt die Prozedur an. Sie ersparen sich damit das Durchsuchen des Moduls nach der passenden Ereignisprozedur.



Tipp





Die Verknüpfung zwischen der Eigenschaft und dem Ereignis wird durch zwei Faktoren aufrechterhalten: den Namen der Prozedur und den Text *[Ereignisprozedur]*, den Access in die betreffende Eigenschaft einträgt. Wenn in einem Formular ein Ereignis auftritt, untersucht Access zunächst die dazugehörige Eigenschaft. Befindet sich hier kein Eintrag, erfolgt dementsprechend auch keine Reaktion auf das Ereignis. Sollte dagegen der Text *[Ereignisprozedur]* vorhanden sein, wird das Formularmodul nach einer Ereignisprozedur durchsucht. Diese Suche erfolgt anhand des Prozedurnamens, der sich bei Steuerelementen folgendermaßen zusammensetzen muss:

```
<Steuerelementname>_<Ereignisname>
```

Bei Ereignissen, die nicht von Steuerelementen, sondern von dem Formular selbst ausgelöst werden, muss der Prozedurname den folgenden Aufbau haben:

```
Form_<Ereignisname>
```

Mit diesem Wissen können Sie nun auch eine Ereignisprozedur "von Hand" anlegen. Wenn Sie beispielsweise auf den Klick auf eine Schaltfläche mit dem Namen *Befehlo* reagieren möchten, tragen Sie in die *Beim Klicken*-Eigenschaft der Schaltfläche den Text *[Ereignisprozedur]* ein.



Der nächste Schritt besteht nun darin, dass Sie das Modulfenster – falls noch nicht erledigt – öffnen. Dies bewerkstelligen Sie z.B. durch einen Klick auf das Symbol *Code*. Hier legen Sie nun eine *Sub*-Prozedur mit dem Namen *Befehlo_Click* an, die Sie anschließend mit Programmcode füllen können. Genau diese beiden Schritte werden auch von Access erledigt, wenn Sie die Editor-Schaltfläche verwenden und anschließend den Assistenten *Code-Editor* auswählen.

Durch diese Art der Verknüpfung zwischen Eigenschaft und Prozedur ergeben sich einige unangenehme Folgen: Wenn Sie beispielsweise ein Steuerelement umbenennen, das bereits mit Ereignisprozeduren verknüpft ist, passt Access den Namen dieser Prozeduren nicht automatisch an. Die Verknüpfung geht also verloren und die Prozedur liegt im Formularmodul brach. In diesem Fall haben Sie die Möglichkeit, den Namen der Prozedur nachträglich anzupassen, um die Verknüpfung wiederherzustellen. Da dies in der Regel eine aufwändige Angelegenheit ist, sollten Sie allen Steuerelementen noch vor der Anlage der ersten Ereignisprozeduren vernünftige Namen geben.

Löschen von Steuerelementen



Wenn Sie ein Steuerelement löschen, bleiben die Ereignisprozeduren im Formularmodul erhalten. Sie brauchen in diesem Fall also nicht den Verlust Ihres Programmcodes zu befürchten. Sollten Sie die Prozeduren nicht mehr benötigen, können Sie die entsprechenden Programmzeilen einfach löschen.

Um ein Steuerelement von einem Ereignis zu lösen, löschen Sie einfach die betreffende Ereignisprozedur. Access bemerkt diesen Vorgang und löscht gleichzeitig auch den Text *[Ereignisprozedur]* aus der betreffenden Eigenschaft.



Standardereignisse

Alle Steuerelemente und Formulare besitzen jeweils ein so genanntes *Standardereignis*. Hierbei handelt es sich um das Ereignis, das in der Praxis am häufigsten benötigt wird. Bei Befehlsschaltflächen ist dieses beispielsweise das *Click*-Ereignis. Für diese Ereignisse stellt Access einen besonders schnellen Weg zur Verfügung, um eine Ereignisprozedur anzulegen: Im Kontextmenü des jeweiligen Steuerelements – das Sie mit einem rechten Mausklick öffnen – befindet sich der Menüpunkt *Ereignis*. Durch Auswahl dieses Menüpunkts erscheint direkt das Dialogfenster *Generator auswählen* und Sie können durch einen Doppelklick auf *Code-Generator* eine Ereignisprozedur zu dem Standardereignis anlegen lassen. Sollte diese Ereignisprozedur bereits bestehen, wird sie durch Auswahl dieses Menüpunkts im Modulfenster zur Bearbeitung aktiviert.

Die folgende Tabelle enthält die Standardereigniseigenschaften der verschiedenen Access-Objekte:

Standardereigniseigenschaften von Access-Objekten

Objekt	Standardereigniseigenschaft
Befehlsschaltfläche	<i>Beim Klicken</i>
Bericht	<i>Beim Öffnen</i>
Berichtsbereiche	<i>Beim Formatieren</i>
Bezeichnungsfeld	<i>Beim Klicken</i>
Bild	<i>Beim Klicken</i>
Diagramm	<i>Bei OLE-Aktualisierung</i>
Formular	<i>Beim Laden</i>
Formularbereiche	<i>Beim Klicken</i>
Kombinationsfeld	<i>Vor Aktualisierung</i>
Kontrollkästchen	<i>Bei Fokuserhalt/Beim Klicken</i>
Linie	Keine Ereignisse!
Listenfeld	<i>Vor Aktualisierung</i>
Objektfeld	<i>Bei OLE-Aktualisierung</i>
Optionsfeld	<i>Bei Fokuserhalt/Beim Klicken</i>
Optionsgruppe	<i>Vor Aktualisierung</i>
Rechteck	<i>Beim Klicken</i>
Seitenumbruch	Keine Ereignisse!



Standardereigniseigenschaften von Access-Objekten (Forts.)

Objekt	Standardereigniseigenschaft
Textfeld	<i>Vor Aktualisierung</i>
Umschaltfläche	<i>Bei Fokuserhalt/Beim Klicken</i>
Unterformular	<i>Beim Hingehen</i>

Tabelle 3.38

Die Standardereigniseigenschaft von Umschaltflächen, Optionsfeldern und Kontrollkästchen ist davon abhängig, ob sich die betreffenden Steuerelemente in einer Optionsgruppe befinden (erste angegebene Ereigniseigenschaft) oder nicht (zweite angegebene Ereigniseigenschaft).

Bei ActiveX-Steuerelementen wird die Standardereigniseigenschaft vom Entwickler des Steuerelements bestimmt. Da diese Steuerelemente auch über Eigenschaften verfügen, die Access nicht kennt, müssen Sie die Standardereigniseigenschaft entweder der Dokumentation des jeweiligen Steuerelements entnehmen oder durch Ausprobieren ermitteln.

Bei Steuerelementen, die auf Berichten platziert sind, steht der Menüpunkt *Ereignis* des Kontextmenüs nicht zur Verfügung, da die Steuerelemente eines Berichts keine Ereignisse auslösen können.

Hinweis



Um das Kontextmenü von Formularen oder Berichten zu aktivieren, klicken Sie mit der rechten Maustaste auf den Titel des Formulars, den Kreuzungspunkt der beiden Lineale oder auf einen Bereich außerhalb des Berichts!

Ereignisprozeduren mit Parametern

Bei einer Reihe von Ereignissen startet Access nicht nur die betreffende Ereignisprozedur, sondern übergibt ihr zusätzlich einige Parameter. Ein Beispiel für ein solches Ereignis ist das *Taste*-Ereignis (engl. *KeyPress*). Dieses Ereignis löst Access aus, wenn eine Taste Ihrer Tastatur betätigt wurde. Die zusätzliche Information besteht in diesem Fall darin, dass Access den ANSI-Zeichencode der gedrückten Taste angibt. Zur Nutzung dieser Information muss der Kopf der Ereignisprozedur einen Parameter besitzen, der die Information innerhalb der Prozedur nutzbar macht. Die entsprechende Ereignisprozedur einer Befehlsschaltfläche könnte beispielsweise folgendermaßen aussehen:

```
Sub Befehl0_KeyPress (KeyAscii As Integer)

End Sub
```



Der Parameter *KeyAscii* enthält in diesem Fall den ASCII-Zeichencode der gedrückten Taste, der in dem für die Tastatur relevanten Bereich mit dem ANSI-Code übereinstimmt. Eine entsprechende Tabelle finden Sie in der auf der CD zum Buch enthaltenen Referenz.

Eine weitere Besonderheit dieses Parameters ist die Tatsache, dass er per *Call by reference* übergeben wird. Sie können der Parametervariablen *KeyAscii* also während der Ausführung der Prozedur einen neuen Wert zuweisen, um so eine gedrückte Taste umzuleiten. Wenn Sie der Variablen den Wert 0 zuweisen, ignoriert Access den Tastendruck vollständig.

Ein anderes Beispiel für solch einen Parameter ist die Ereignisprozedur der *Beim Öffnen*-Eigenschaft von Formularen:

```
Sub Form_Open (Cancel As Integer)
```

```
End Sub
```

Auch diese Prozedur besitzt einen Parameter. Die Variable *Cancel* besitzt beim Start der Prozedur stets den Inhalt *False*. Wenn Sie diese nun während der Abarbeitung der Prozedur auf *True* setzen, bricht Access nach Beendigung der Prozedur das Öffnen des Formulars ab. Das folgende Beispiel zeigt dies:

```
Sub Form_Open (Cancel As Integer)
```

```
Dim Auswahl As Integer
```

```
    Auswahl = MsgBox("Soll das Formular  
geöffnet werden?", _  
                    vbYesNo)
```

```
    If Auswahl = vbNo Then Cancel = True
```

```
End Sub
```

In diesem Beispiel wird eine Funktion mit dem Namen *MsgBox* verwendet, die eng mit der *MsgBox*-Anweisung verwandt ist. Anstelle einer einzelnen *OK*-Schaltfläche können jedoch mehrere Schaltflächen mit unterschiedlichen Beschriftungen angezeigt werden. Welche Schaltflächen angezeigt werden, bestimmt ein Wert, der als zweiter Parameter angegeben wird. Die folgende Tabelle gibt darüber Auskunft:

Schaltflächen der *MsgBox*-Funktion

Wert	Konstante	Bedeutung
0	<i>vbOkOnly</i>	Nur Schaltfläche <i>OK</i>
1	<i>vbOkCancel</i>	<i>OK</i> und <i>Abbrechen</i>



Schaltflächen der *MsgBox*-Funktion (Forts.)

Wert	Konstante	Bedeutung
2	<i>vbAbortRetryIgnore</i>	<i>Abbrechen, Wiederholen und Ignorieren</i>
3	<i>vbYesNoCancel</i>	<i>Ja, Nein und Abbrechen</i>
4	<i>vbYesNo</i>	<i>Ja und Nein</i>
5	<i>vbRetryCancel</i>	<i>Wiederholen und Abbrechen</i>

Tabelle 3.39

In der Beispielprozedur wird also eine *Ja*- und eine *Nein*-Schaltfläche angezeigt. Das Ergebnis der Funktion ist ein Wert, der die gedrückte Taste kennzeichnet. Folgende Werte können zurückgegeben werden:

Rückgabewerte der *MsgBox*-Funktion

Wert	Konstante	Betätigte Schaltfläche
1	<i>vbOK</i>	<i>OK</i>
2	<i>vbCancel</i>	<i>Abbrechen</i>
3	<i>vbAbort</i>	<i>Abbruch</i>
4	<i>vbRetry</i>	<i>Wiederholen</i>
5	<i>vbIgnore</i>	<i>Ignorieren</i>
6	<i>vbYes</i>	<i>Ja</i>
7	<i>vbNo</i>	<i>Nein</i>

Tabelle 3.40

Klickt der Benutzer also auf *Nein*, gibt die Funktion den Wert 7 (*vbNo*) zurück. In der Beispielprozedur wird dieses Ergebnis mithilfe einer *If*-Abfrage überprüft und dementsprechend die Variable *Cancel* auf den Wert *True* gesetzt. Das Öffnen des Formulars wird dadurch abgebrochen, und das Formular erscheint erst gar nicht auf dem Bildschirm.

Hinweis



Oft ist es notwendig, eine Prozedur direkt beim Öffnen einer Datenbank auszuführen, um beispielsweise Initialisierungsaufgaben zu erledigen. Zu diesem Zweck legen Sie ein Makro mit dem Namen *AUTOEXEC* an. Beim Start einer Datenbank durchsucht Access die Datenbank automatisch nach einem Makro mit diesem



Namen. Ist es vorhanden, wird es automatisch gestartet. Von diesem Makro aus können Sie nun mithilfe der Aktion *AusführenCode* auch eine oder mehrere VBA-Funktionen ausführen.

Verknüpfen von Funktionen mit Ereignissen

In früheren Access-Versionen (1.0/1.1) wurde die Verknüpfung zwischen einem Ereignis und einer Prozedur hergestellt, indem der Name einer *Function*-Prozedur inklusive Parametern und voranstehendem Gleichheitszeichen direkt in das Eingabefeld der betreffenden Ereignisseigenschaft eingetragen wurde. Diese Möglichkeit wird aus Kompatibilitätsgründen zu den älteren Versionen in Access 97 immer noch unterstützt, in der Praxis aber nur noch selten eingesetzt.

Um beispielsweise beim Mausklick auf eine Schaltfläche eine *Function*-Prozedur mit dem Namen *MeldungAusgeben* auszuführen, tragen Sie in die *Beim Klicken*-Eigenschaft der Schaltfläche den folgenden Ausdruck ein:

```
= MeldungAusgeben()
```

Die Prozedur *MeldungAusgeben* muss sich dabei in dem aktuellen Formularmodul (privat oder öffentlich) oder einem Standardmodul (öffentlich) befinden. Der Rückgabewert der Funktion wird stets ignoriert!

Wenn Sie eine Prozedur angeben, die Parameter erwartet, können Sie diese ebenfalls wie gewohnt eingeben. Anstelle der Kommas müssen Sie hierbei allerdings Semikolons als Trennzeichen zwischen den Parametern verwenden.

Auch mit Access XP kann der Einsatz von *Function*-Prozeduren in Verbindung mit Ereignisseigenschaften sinnvoll sein. Wenn Sie beispielsweise mehrere Objekte mit einem identischen Ereigniscode versehen möchten, können Sie eine einzelne *Function*-Prozedur von mehreren Ereignisseigenschaften aus starten. Wenn Sie die Prozedur öffentlich in einem Standardmodul anlegen, ist diese sogar formularübergreifend nutzbar.

Verwenden von
Funktionen als
Ereignisprozeduren





Programmieren mit Objekten



Was ist ein Objekt?	194
Die Access-Objekthierarchie	195
Zugriff auf Eigenschaften und Methoden	208
Objektvariablen	213
Auflistungen	217
Der Objektkatalog	220





Im Gegensatz zu klassischen Programmiersprachen stellt Access ein komplettes Entwicklungssystem dar, das neben der eigentlichen Programmiersprache aus Tabellen, Abfragen, Formularen und Berichten besteht. Damit das Zusammenspiel dieser Komponenten funktioniert, bietet Access die Möglichkeit, von der Programmiersprache aus auf die Objekte der Datenbank zuzugreifen.

In diesem Kapitel geht es um die grundlegenden Techniken, die den Zugriff auf Datenbankobjekte ermöglichen. Das Thema ist vor allem der Umgang mit Objekten sowie der Zugriff auf Formulare und Steuerelemente.

Die in diesem Kapitel vorgestellten Beispiele finden Sie zusammen mit dem Formular *Kunden* in der Datenbank *Kapitel4.mdb*.

Was ist ein Objekt?

Die Elemente von Access wie beispielsweise Tabellen, Formulare oder Steuerelemente werden von VBA als Objekte betrachtet. Jedes Objekt hat einen Satz an Eigenschaften und Methoden, über die auf das Objekt Einfluss genommen werden kann. Sie bilden damit die Schnittstelle des Objekts zur Außenwelt. Über eine Objekteigenschaft kann der Zustand eines Objekts ermittelt oder verändert werden. Formulare haben beispielsweise eine Eigenschaft mit dem Namen *Beschriftung*, über den Sie den Titel des Formularfensters auslesen oder verändern können. Methoden haben dagegen den Charakter eines Befehls, der an das Objekt ausgegeben wird. Eine typische Methode ist die *Requery*-Methode von Formularen. Diese Methode bewirkt, dass Access alle Datensätze aktualisiert, die in dem Formular angezeigt werden.

Eigenschaften und Methoden

Jedes Objekt stellt individuelle Eigenschaften und Methoden zur Verfügung. So unterstützen z.B. Formulare andere Eigenschaften als Steuerelemente. Auf der anderen Seite haben alle Formulare die gleichen Eigenschaften und sind sich damit ähnlich. Aus diesem Grund wird zwischen den folgenden beiden Bezeichnungen unterschieden:

Objektklasse und Objektinstanz

Die *Objektklasse* definiert ein allgemeines Objekt, das einen bestimmten Satz an Eigenschaften und Methoden hat. Eine Objektklasse kann daher auch mit einer Schablone oder einer Vorlage verglichen werden. Formulare und Steuerelemente der Toolbox sind solche Objektklassen. Wenn Sie ein neues Formular öffnen oder ein neues Steuerelement anlegen, legt Access eine *Objektinstanz* nach dem Vorbild der Objektklasse an. Solche Objektinstanzen werden kurzerhand als *Objekt* bezeichnet.

Jedes einzelne Objekt hat bestimmte Merkmale, die es von anderen Objekten unterscheidet. So können Sie beispielsweise innerhalb eines Formulars mehrere Textfelder anlegen, die unterschiedliche Ausmaße oder Farben haben. Diese Merkmale können Sie mithilfe von Eigenschaften verändern. Eigenschaften wirken sich also immer auf ein Objekt und nicht auf eine Objektklasse aus. Gleiches gilt für Methoden.



In den folgenden Abschnitten lernen Sie die Access-Objekte kennen, die beispielsweise für den Zugriff auf Formulare, Berichte und Steuerelemente erforderlich sind.

Die Access-Objekthierarchie

Die einzelnen vordefinierten Objektklassen von Access lassen sich in einer Hierarchie ordnen. Die folgende Abbildung gibt Ihnen einen groben Überblick über diese Hierarchie.

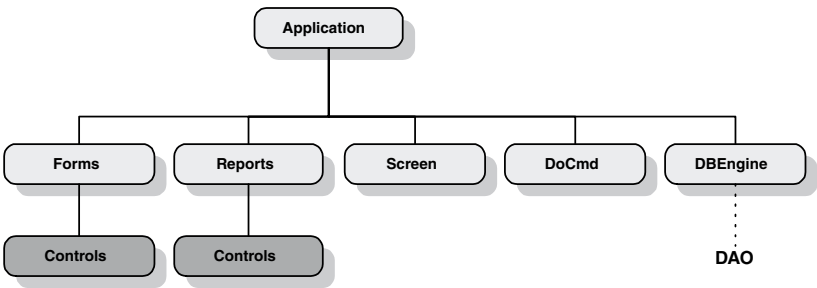


Abb. 4.1

Die Access-Objekthierarchie

An der Spitze der Hierarchie befindet sich das *Application*-Objekt, das die komplette Access-Anwendung repräsentiert. Aus der Sicht des Access-Entwicklers handelt es sich hierbei nicht um eine Objektklasse, sondern um ein Objekt, das von Access automatisch beim Öffnen einer Datenbank angelegt wird. Wie Sie aber später noch im Kapitel 13 OLE und DDE erfahren werden, können andere Windows-Anwendungen weitere Instanzen des *Application*-Objekts anlegen, um so die Access-Funktionen in ihren Funktionsumfang zu integrieren.

Application-Objekt

Das *Application*-Objekt stellt allgemeine Methoden und Eigenschaften zur Verfügung, die Access betreffen. Dazu gehört beispielsweise der Zugriff auf den *Optionen*-Dialog. In der Programmierpraxis werden die Methoden und Eigenschaften des *Application*-Objekts selten benötigt. Wichtiger ist da schon die Möglichkeit, über das *Application*-Objekt Zugriff auf die Objekte der unteren Ebenen zu erhalten. Die folgende Tabelle enthält eine Übersicht über alle Objekte:

Access-Objekte

Objekt	Beschreibung
<i>Application</i>	Repräsentiert Access.
<i>AnswerWizard</i>	Der Antwort-Assistent
<i>Assistant</i>	Dieses Objekt repräsentiert den animierten Office-Assistenten. Über dessen <i>Balloon</i> -Objekt kann dann auch die Sprechblase programmiert werden.



Access-Objekte (Forts.)

Objekt	Beschreibung
<i>CodeData</i>	Über dieses Objekt kann auf die Code-Objekte des Servers zugegriffen werden (siehe auch <i>CurrentData</i>).
<i>CodeProject</i>	Hiermit kann auf die Code-Objekte des Projekts zugegriffen werden (siehe auch <i>CurrentProject</i>).
<i>COMAddIns</i>	Über dieses Objekt kann auf die verfügbaren Add-Ins zugegriffen werden.
<i>CommandBars</i>	Eine Auflistung, die die Menü- und Werkzeugleisten der Anwendung enthält.
<i>Controls</i>	Eine Auflistung, in der sich pro Steuerelement des Formulars oder Berichts ein <i>Control</i> -Objekt befindet.
<i>CurrentData</i>	Über dieses Objekt kann auf alle Objekte (Datenbanken, Abfragen usw.) des Servers zugegriffen werden. Es enthält die Auflistungen <i>AllTables</i> , <i>AllQueries</i> , <i>AllViews</i> , <i>AllStoredProcedures</i> und <i>AllDatabaseDiagrams</i> .
<i>CurrentProject</i>	Dieses Objekt enthält die Objekte (auch ungeöffnete) des aktuellen Projekts in den Auflistungen <i>AllForms</i> , <i>AllReports</i> , <i>AllMacros</i> , <i>AllModules</i> und <i>AllDataAccessPages</i> .
<i>DataAccessPages</i>	Eine Auflistung, die die geöffneten Datenzugriffsseiten enthält.
<i>DBEngine</i>	Ermöglicht den Zugriff auf die DAO-Datenbank-Engine. Unterhalb des <i>DBEngine</i> -Objekts folgen die Data Access Objects zum Zugriff auf Daten. Mehr dazu erfahren Sie im <i>Kapitel 5 Datenzugriff mit VBA</i> .
<i>DefaultWebOptions</i>	Dieses Objekt enthält die Attribute einer Datenzugriffsseite, die als Webseite gespeichert wird.
<i>DoCmd</i>	Stellt Makroaktionen unter VBA zur Verfügung.
<i>FileSearch</i>	Dieses Objekt repräsentiert den Dialog zum Öffnen einer Datei.
<i>Forms</i>	Eine Auflistung, die alle geöffneten Formulare enthält. Ein Formular wird hierbei jeweils durch ein <i>Form</i> -Objekt repräsentiert.
<i>LanguageSettings</i>	Dieses Objekt enthält die Einstellungen der unterstützten Sprachen der Anwendung.
<i>Modules</i>	Eine Auflistung, die alle geöffneten Module enthält.



Access-Objekte (Forts.)

Objekt	Beschreibung
<i>References</i>	Eine Auflistung, die alle aktuellen Verweise enthält.
<i>Reports</i>	Eine Auflistung, die alle geöffneten Berichte enthält. Ein Bericht wird hierbei jeweils durch ein <i>Report</i> -Objekt repräsentiert.
<i>Screen</i>	Ermöglicht den Zugriff auf momentan aktive Objekte.
<i>VBE</i>	Dieses Objekt enthält alle Objekte und Auflistungen für V isual B asic for A pplications (VBA).

Tabelle 4.1

Bei den oben aufgeführten Objekten handelt es sich genau wie beim *Application*-Objekt um Objektinstanzen und keine Objektklassen. Das *Forms*-Objekt existiert beispielsweise nur ein einziges Mal. Es ist nicht möglich, zusätzliche *Forms*-Objekte anzulegen.

Aus zeichentechnischen Gründen ist die Grafik an einem Punkt vereinfacht: Bei den Objekten *Forms* und *Reports* handelt es sich um so genannte *Auflistungen*. Solche Auflistungen erlauben den Zugriff auf untergeordnete Objekte. Die *Forms*-Auflistung enthält beispielsweise alle Formulare, die unter Access momentan geöffnet sind. Jedes einzelne Formular wird innerhalb der Auflistung durch ein *Form*-Objekt repräsentiert. Das *Forms*-Objekt hätte also folgendermaßen gezeichnet werden müssen:

Auflistungen

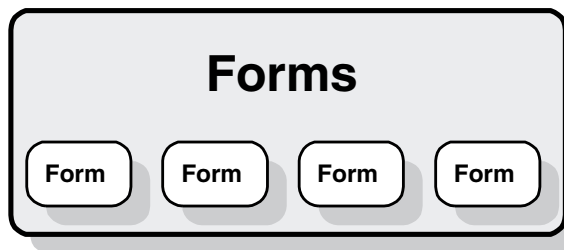


Abb. 4.2

Form-Objekte der Auflistung *Forms*

Jedes *Form*-Objekt – also jedes Formular – enthält wiederum eine Auflistung mit dem Namen *Controls*, über die der Zugriff auf die einzelnen Steuerelemente des Formulars möglich ist.

In den folgenden Abschnitten lernen Sie die wichtigsten Objekte der Objekthierarchie noch genauer kennen. Sie erfahren beispielsweise, wie Sie die Eigenschaften eines Steuerelements von VBA aus verändern können.

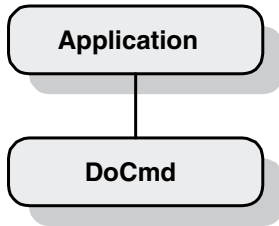




Ausführen von Makroaktionen (*DoCmd*-Objekt)

Abb. 4.3

Das *DoCmd*-Objekt



OpenForm-Methode

Das *DoCmd*-Objekt haben Sie bereits im *Kapitel 2 Schnellstart VBA* kennen gelernt. Das Objekt ermöglicht die Ausführung aller Makroaktionen von VBA aus. Zu diesem Zweck stellt das Objekt passend zu jeder Aktion eine entsprechende Methode zur Verfügung. Die Namen der Methoden entsprechen der englischen Übersetzung der deutschen Aktionsnamen. Aus der *ÖffnenFormular*-Aktion wird so beispielsweise die *OpenForm*-Methode. Die folgende Anweisung öffnet beispielsweise das Formular *Projekte*:

```
Application.DoCmd.OpenForm "Projekte"
```

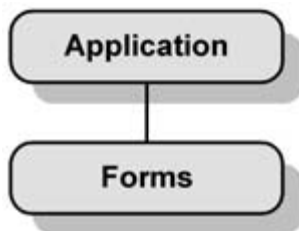
Bei allen Objekten, die direkt unterhalb des *Application*-Objekts angeordnet sind, muss der Name *Application* nicht unbedingt angegeben werden. In der Praxis wird daher folgende Kurzform verwendet:

```
DoCmd.OpenForm "Projekte"
```

Bezugnahme auf Formulare (*Form*-Objekte)

Abb. 4.4

Die *Forms*-Auflistung



Auflistungen

Der Zugriff auf Formulare erfolgt innerhalb von VBA mithilfe der *Forms*-Auflistung. Eine Auflistung ist nichts anderes als eine Gruppe von Objekten, die in dieser Auflistung tabellarisch erfasst sind. So enthält die Auflistung *Forms* alle Formulare, die momentan geöffnet sind. Die Betonung liegt in diesem Fall auf "geöffnet", da die geschlossenen Formulare hier nicht enthalten sind! Daraus ergibt sich auch, dass der Zugriff auf ein Formular nur dann möglich ist, wenn es geöffnet ist. Dies bedeu-



tet allerdings keine Einschränkung, da Formulare leicht mithilfe der *ÖffnenFormular*-Aktion (bzw. der *OpenForm*-Methode) geöffnet werden können. Das geöffnete Formular wird dann automatisch der *Forms*-Auflistung hinzugefügt.

Sie können aber auch auf alle Formulare eines Projekts zugreifen, selbst wenn diese nicht geöffnet sind. Hierzu dient die Auflistung *AllForms* in den Objekten *CurrentProject* bzw. *CodeProject*. In den folgenden Beispielen wird jedoch die *Forms*-Auflistung des *Application*-Objekts eingesetzt.

○ —————
AllForms

Um ein Formular aus der Auflistung *Forms* auszuwählen, gibt es drei verschiedene Möglichkeiten. Die erste Möglichkeit besteht darin, hinter dem Namen der Auflistung den Namen des Formulars anzugeben. Der Formularname muss hierbei mit einem Ausrufungszeichen vom Namen der Auflistung getrennt werden:

○ —————
Formularname

```
Application.Forms!<Formularname>
```

Der folgende Ausdruck nimmt beispielsweise auf ein Formular mit dem Namen *Formular1* Bezug. Beachten Sie, dass das Formular geöffnet sein muss. Sollte dies nicht der Fall sein, führt der Ausdruck zu einem Fehler:

```
Application.Forms!Formular1
```

Wenn der Name des Formulars Leer- oder andere Sonderzeichen enthält, muss er in eckigen Klammern angegeben werden:

```
Application.Forms![Adresse auswählen, Dialog]
```

Genau wie bei dem *DoCmd*-Objekt kann der Name des Objekts *Application* auch ausgelassen werden. In der Praxis wird daher zur Bezugnahme auf ein Formular die folgende Kurzform verwendet:

```
Forms!<Formularname>
```

Die beiden obigen Beispiele sehen unter Verwendung dieser Kurzform folgendermaßen aus:

```
Forms!Formular1
```

```
Forms![Adresse auswählen, Dialog]
```

In den beiden Beispielen wird der Name der Formulare fest im Programmcode angegeben. Der jeweilige Ausdruck nimmt also immer auf das gleiche Formular Bezug. Wenn Sie dies ändern möchten, ist eine Änderung des Programmcodes erforderlich.

Um hier mehr Flexibilität zu ermöglichen, kann der Name des gewünschten Formulars auch in einer Variablen des Datentyps *String* oder *Variant* enthalten sein. In diesem Fall kann die Bezugnahme folgendermaßen erfolgen:

○ —————
Zeichenkette

```
Forms <Formularname>
```





Formularname ist in diesem Beispiel eine Variable, die den Namen des Formulars enthält. Natürlich kann anstelle dieser Variablen auch direkt eine Zeichenkette angegeben werden:

```
Forms <"Formular1">
```

Position



Die dritte Möglichkeit, auf ein Formular Bezug zu nehmen, bezieht sich auf die Reihenfolge, mit der die einzelnen Formulare in der Auflistung *Forms* angeordnet sind. Wenn beispielsweise drei Formulare geöffnet sind, haben diese in der *Forms*-Auflistung die Positionsnummern 0, 1 und 2. Die Positionsnummer wird in diesem Zusammenhang auch als *Index* bezeichnet. Um z. B. auf das zweite Formular zu verweisen, kann der folgende Ausdruck verwendet werden:

```
Forms (1)
```

Anstelle des Zahlenwertes 1 ist auch die Angabe einer numerischen Variablen zulässig:

```
Forms <Formularnummer>
```

Die dritte Variante wird in der Praxis nur selten eingesetzt, wenn auf ein bestimmtes Formular verwiesen werden soll. Der Grund dafür ist, dass Access die einzelnen Formulare innerhalb der *Forms*-Auflistung beliebig anordnet. Solange Sie den Index eines bestimmten Formulars nicht genau kennen, können Sie diese Variante nicht sinnvoll einsetzen.

Hinweis



Bei der Anordnung der Objekte innerhalb der *Forms*-Auflistung achtet Access immer darauf, dass die Indizes stets ohne Lücke vergeben werden. Wenn beispielsweise ein Formular mit dem Index 2 geschlossen wird, rücken alle folgenden Formulare automatisch auf, um die Lücke zu schließen.

Die Me-Eigenschaft

Eine weitere Möglichkeit zur Bezugnahme auf ein Formular stellt die *Me*-Eigenschaft zur Verfügung. Sie ist auf den Einsatz in Formularmodulen beschränkt und verweist automatisch immer auf das zu dem Formularmodul gehörende Formular. Wenn Sie die *Me*-Eigenschaft beispielsweise in dem Formularmodul des Formulars *Formular1* verwenden, sind die beiden folgenden Ausdrücke vom Ergebnis identisch:

```
Me  
Forms!Formular1
```

Hinweis



Wenn Sie innerhalb eines Formularmoduls auf das zugehörige Formular verweisen, sollten Sie in jedem Fall die *Me*-Eigenschaft verwenden. Der Vorteil besteht unter anderem darin, dass der Programmcode dadurch wesentlich kürzer und übersichtlicher wird. Der wichtigere Grund ist aber, dass Sie den Namen des Formulars nicht im Programmcode angeben müssen. Wenn Sie also nachträglich den Namen des Formulars abändern, sind keine Änderungen am Programmcode erforderlich!



Die folgende Tabelle fasst noch einmal alle Möglichkeiten für die Bezugnahme auf Formulare zusammen:

Bezugnahme auf Formulare

Ausdruck	Beschreibung
<i>Forms!</i> <Formularname>	Verweist auf das geöffnete Formular mit dem Namen <Formularname>.
<i>Forms</i> (<Zeichenkette>)	Verweist auf das geöffnete Formular, dessen Name innerhalb der <Zeichenkette> angegeben ist.
<i>Forms</i> (<Index>)	Verweist auf das geöffnete Formular, das sich in der Liste aller geöffneten Formulare an der angegebenen <Position> befindet.
<i>Me</i>	Verweist auf das aktuelle Formular.

Tabelle 4.2

Formularbereiche

Jedes Formular kann maximal fünf Bereiche enthalten, die ebenfalls mit Namen versehen sind:

- Detailbereich
- Formularkopf
- Formularfuß
- Seitenkopf
- Seitenfuß

Die allgemeine Syntax zum Verweis auf einen Bereich eines Formulars lautet folgendermaßen:

<Formularobjekt>.<Bereichsname>

Unter <Formularobjekt> geben Sie einen beliebigen Verweis auf ein bestimmtes Formular an. Anschließend folgt ein Punkt und der Name eines Bereichs. Der folgende Ausdruck verweist beispielsweise auf den Detailbereich des Formulars *Kunden*:

`Forms!Kunden.Detail0`



Section-Eigenschaft

Als zweite Möglichkeit zum Verweis auf einen Bereich stellt Access die *Section*-Eigenschaft von Formularen zur Verfügung. Bei der *Section*-Eigenschaft handelt es sich um keine typische Eigenschaft, da sie im Gegensatz zu anderen Eigenschaften einen Parameter erwartet:

```
<Formularobjekt>.Section (<Bereichsnummer>)
```

Über das numerische Argument *<Bereichsnummer>* legen Sie den Bereich fest, auf den Sie verweisen möchten. Die folgende Tabelle gibt Ihnen Auskunft darüber, welche Werte Sie angeben können:

Werte der *Section*-Eigenschaft

<i><Bereichsnummer></i>	Beschreibung
0	Detailbereich
1	Formularkopf
2	Formularfuß
3	Formularseitenkopf
4	Formularseitenfuß

Tabelle 4.3

Das folgende Beispiel verweist auf den Detailbereich des Formulars *Kunden*:

```
Forms!Kunden.Section(0)
```



Bezugsnahme auf Berichte (Report-Objekte)

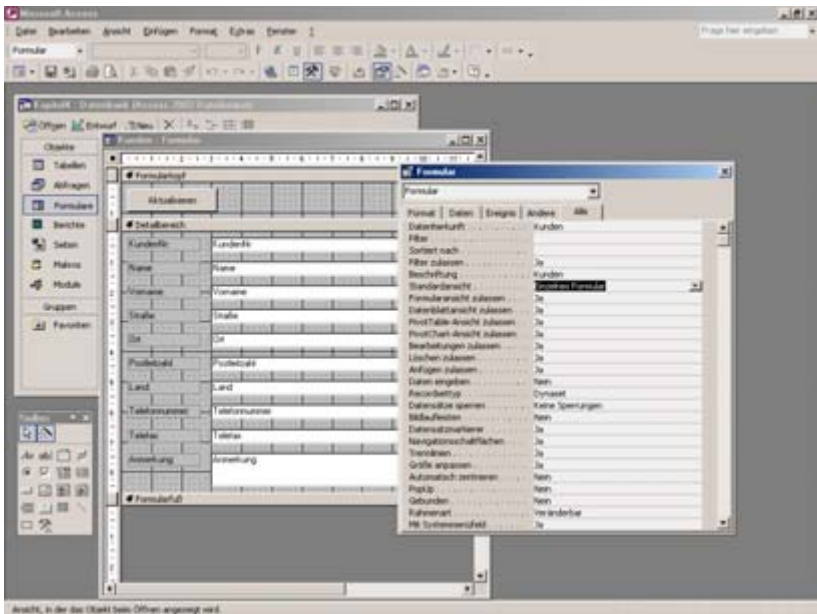


Abb. 4-5

Die Reports-Auflistung

Der Zugriff auf Berichte erfolgt auf die gleiche Weise wie der Zugriff auf Formulare. Anstelle der Forms-Auflistung muss lediglich die Reports-Auflistung verwendet werden. Das folgende Beispiel verweist beispielsweise auf den Bericht *Kundenliste*:

Reports-Auflistung

```
Reports! [Kundenliste]
```

Voraussetzung ist natürlich, dass der Bericht in der Seitenansicht geöffnet ist. Die *Me*-Eigenschaft steht genau wie bei Formularen auch bei Berichten zur Verfügung.

Wenn Sie beim Verweis auf die Bereiche eines Berichts die *Section*-Eigenschaft verwenden, geben Sie einen der folgenden Werte an:

Werte der Section-Eigenschaft bei Berichten

<Bereichsnummer>	Beschreibung
0	Detailbereich
1	Berichtskopf
2	Berichtsfuß
3	Berichtsseitenkopf

Werte der *Section*-Eigenschaft bei Berichten (Forts.)

<Bereichsnummer>	Beschreibung
4	Berichtsseitenfuß
5	Gruppenebene 1, Kopfbereich
6	Gruppenebene 1, Fußbereich
7	Gruppenebene 2, Kopfbereich
8	Gruppenebene 2, Fußbereich
>= 9	Alle weiteren Kopf- und Fußbereiche

Tabelle 4.4

Hinweis



Da Sie bei Berichten beliebig viele Kopf- und Fußbereiche definieren können, erfolgt der Verweis auf die Gruppierungsebenen oberhalb der zweiten Ebene mit Werten ab 9.

Bezugnahme auf Steuerelemente (*Control*-Objekte)

Jedes Formular und jeder Bericht besitzt eine *Controls*-Auflistung, in der alle Steuerelemente des Formulars aufgelistet sind. Jedes einzelne Steuerelement wird hierbei durch ein Objekt des Typs *Control* repräsentiert.

Controls-Auflistung

Ein bestimmtes Steuerelement kann aus der *Controls*-Auflistung auf die gleiche Weise ausgewählt werden, wie Sie beispielsweise ein Formular aus der *Forms*-Auflistung auswählen. Der folgende Ausdruck verweist beispielsweise auf das Steuerelement mit dem Namen *Ort*, das sich im Formular *Kunden* befindet:

```
Forms!Kunden.Controls!Ort
```

Auch die anderen beiden Varianten zum Verweis auf Objekte in Auflistungen sind zulässig. Wenn der Name des gewünschten Steuerelements in der Variablen *MeinSteuerelementname* enthalten ist, können Sie beispielsweise den folgenden Ausdruck verwenden:

```
Forms!Kunden.Controls(MeinSteuerelementname)
```

Anstelle des Steuerelementnamens kann auch ein Index angegeben werden, der ein bestimmtes Steuerelement über die Position innerhalb der *Controls*-Auflistung auswählt:

```
Forms!Kunden.Controls(Steuerelementnummer)
```



Alle drei Varianten können innerhalb des Pfades beliebig gemischt werden. Es ist beispielsweise zulässig, auf das Formular per Zeichenkette und auf das Steuerelement direkt über den Namen zuzugreifen:



Hinweis

```
Forms("Kunden").Controls!Ort
```

Wenn z.B. ausschließlich Zeichenketten verwendet werden, könnte der Ausdruck folgendermaßen aussehen:

```
Forms("Kunden").Controls("Ort")
```

In der Praxis wird zum Verweis auf ein bestimmtes Steuerelement oft eine Kurzform verwendet. Hierbei wird der Name des Objekts *Controls* einfach ausgelassen:



Kurzform

```
Forms!Kunden!Ort  
Forms!Kunden(MeinSteuerelementname)  
Forms!Kunden(Steuerelementnummer)
```

Dies ist zulässig, da Formulare nur eine einzige Auflistung – nämlich die *Controls*-Auflistung – besitzen. Wenn Sie den Namen *Controls* also nicht explizit angeben, kann Access trotzdem eindeutig ermitteln, dass Sie ein Objekt aus dieser Auflistung auswählen möchten. Wie Sie im Zusammenhang mit den Datenzugriffsobjekten noch kennen lernen, gibt es Objekte, die mehrere Auflistungen enthalten. Hier wird in der Regel eine Auflistung als Standardauflistung definiert, die Access automatisch dann verwendet, wenn Sie den Namen der Auflistung auslassen. Wenn Sie dagegen ein Objekt aus einer anderen Auflistung auswählen möchten, müssen Sie den Namen explizit angeben.

Unterformulare

Unterformulare haben einen besonderen Status, da sie nicht als eigenständige Formulare auf der Arbeitsoberfläche von Access auftauchen, sondern in andere Formulare eingebettet sind.

Aus der Sicht von Access ist ein Unterformular nichts anderes als ein Steuerelement des Typs *Unterformular/-bericht*, das in einem Formular platziert ist. Innerhalb dieses Steuerelements wird das eigentliche Unterformular angezeigt. Das Steuerelement des Typs *Unterformular/-bericht* besitzt wie jedes Steuerelement einen Namen, der mithilfe der *Name*-Eigenschaft festgelegt wird. Welches Formular nun innerhalb dieses Steuerelements erscheint, legt schließlich die *Herkunftsobjekt*-Eigenschaft des Steuerelements fest. Wenn Sie ein Unterformular per Drag & Drop in ein Formular einfügen, vergibt Access für das Steuerelement den gleichen Namen, den auch das Unterformular besitzt. Aus diesem Grund bleibt die Zweiteilung in Steuerelement und Herkunftsobjekt für den Entwickler oftmals unbemerkt.



**Unterformular-
Steuerelemente**

Das folgende Beispiel bezieht sich auf ein Formular mit dem Namen *Kunden*, in das ein Unterformular mit dem Namen *Rechnungen* eingebettet ist. Das Unterformular *Rechnungen* befindet sich hierbei in einem Steuerelement des Typs *Unterformular/-bericht*, das den Namen *RechnungenUnterformular* hat. Beachten Sie hierbei, dass sich der Name des Steuerelements vom Namen des Herkunftsobjekts unterscheidet.



Um nun den Zugriff auf das Unterformular zu ermöglichen, muss erst einmal der Verweis auf das *Unterformular*-Steuerelement hergestellt werden. Dies erledigen Sie z. B. auf die folgende Art und Weise:

```
Forms!Kunden!RechnungenUnterformular
```

Die *Form*-Eigenschaft

Der Übergang von dem Steuerelement in das eigentliche Unterformular wird nun mithilfe der *Form*-Eigenschaft bewerkstelligt, die von allen Steuerelementen des Typs *Unterformular/-bericht* zur Verfügung gestellt wird:

```
Forms!Kunden!RechnungenUnterformular.Form
```

Dieser komplette Ausdruck hat nun wieder einen Verweis auf ein "normales" Formular zum Ergebnis, über den Sie z. B. auf Steuerelemente innerhalb dieses Formulars Bezug nehmen können. Der folgende Ausdruck verweist z. B. auf das Steuerelement *Rechnungsdatum* des Unterformulars:

```
Forms!Kunden!RechnungenUnterformular.Form!Rechnungsdatum
```

Das *Screen*-Objekt

Das *Screen*-Objekt stellt einige Eigenschaften zur Verfügung, mit denen sich etwa auf ein aktives Formular, einen aktiven Bericht oder das momentan aktive Steuerelement verweisen lässt. Die folgende Tabelle gibt Ihnen eine Übersicht über diese Eigenschaften:

Eigenschaften des *Screen*-Objekts

Eigenschaft	Beschreibung
<i>ActiveForm</i>	Hat einen Verweis auf das momentan aktive Formular zum Ergebnis.
<i>ActiveReport</i>	Hat einen Verweis auf den momentan aktiven Bericht zum Ergebnis.
<i>ActiveDatasheet</i>	Verweist auf das aktive Datenblatt.
<i>ActiveDataAccessPage</i>	Verweist auf die aktive Datenzugriffsseite.
<i>Application</i>	Verweist auf die aktive Anwendung (<i>Application</i> -Objekt).
<i>MousePointer</i>	Bestimmt die Form des Mauszeigers.
<i>ActiveControl</i>	Verweist auf das aktive Steuerelement.
<i>PreviousControl</i>	Verweist auf das vorher aktive Steuerelement.

Tabelle 4.5

Wenn beispielsweise das Formular *Kunden* momentan aktiv ist, sind die folgenden beiden Ausdrücke gleichwertig:

```
Forms! [Kunden]  
Screen.ActiveForm
```

Wenn in dem Formular das Steuerelement *Nachname* aktiv ist, sind die folgenden beiden Ausdrücke ebenfalls gleichwertig:

```
Forms! [Kunden] ! [Nachname]  
Screen.ActiveControl
```

Die letzte Eigenschaft der Tabelle mit dem Namen *PreviousControl* hat eine besondere Funktion: Damit lässt sich auf das Steuerelement verweisen, das direkt vor dem momentan aktiven Steuerelement den Fokus hatte. Dies wird beispielsweise benötigt, wenn Sie eine universell einsetzbare Prozedur entwickeln, die den Inhalt eines beliebigen Steuerelements verändert. Wenn die Prozedur nun über eine Schaltfläche gestartet wird, erhält diese Schaltfläche beim Anklicken den Fokus. Mit der *PreviousControl*-Eigenschaft können Sie daraufhin ermitteln, welches Steuerelement vorher den Fokus hatte.

 *PreviousControl*

Alle Eigenschaften des *Screen*-Objekts führen zu einem Laufzeitfehler, wenn kein betreffendes Objekt aktiv ist. Vor dem Aufruf sollte daher immer per *On Error*-Anweisung eine Fehlerbehandlungsroutine aktiviert werden.



Hinweis

Das *CurrentProject*-Objekt

Das *CurrentProject*-Objekt ermöglicht den Zugriff auf alle Elemente des aktuellen Projekts. Im Gegensatz zum direkten Zugriff, etwa über die *Forms*-Auflistung, können hier auch die ungeöffneten Formulare, Berichte, Module usw. erreicht werden.

Das Objekt enthält die Auflistungen *AllForms* (alle Formulare), *AllReports* (alle Berichte), *AllDataAccessPages* (alle Datenzugriffsseiten) *AllMacros* (alle Makros) und *AllModules* (alle Module). Mit diesen Auflistungen können Sie genauso arbeiten wie mit den entsprechenden Varianten des *Application*-Objekts (ohne *All*). Außerdem erlaubt die Auflistung *AccessObjectProperties* den Zugriff auf Standard- oder selbstdefinierte Eigenschaften des Objekts.

Die folgende Prozedur listet alle Formulare in dem aktuellen Projekt im Direktfenster auf. Außerdem wird mit angegeben, ob das jeweilige Formular geladen ist (*True*) oder nicht (*False*).

```
Sub AlleFormulareAuflisten()  
  
    Dim i As Integer  
  
    For i = 0 To CurrentProject.AllForms.Count - 1  
  
        With CurrentProject.AllForms(i)
```





```

        Debug.Print .Name, .IsLoaded
    End With
Next i
End Sub

```

Das *CurrentData*-Objekt

Das *CurrentData*-Objekt ermöglicht den Zugriff auf alle Serverelemente der aktuellen Datenbank, auch ungeöffnete. Es enthält die Auflistungen *AllTables* (alle Tabellen), *AllQueries* (alle Abfragen), *AllViews* (alle Ansichten), *AllStoredProcedures* (alle gespeicherten Prozeduren) und *AllDatabaseDiagrams* (alle Diagramme).

Die folgende Prozedur listet alle Tabellen in der aktuellen Datenbank im Direktfenster auf. Hierbei werden die internen Systemtabellen, deren Namen mit *MSys* beginnen, ausgelassen.

```

Sub AlleTabellenAuflisten()

    Dim i As Integer

    For i = 0 To CurrentData.AllTables.Count - 1

        With CurrentData.AllTables(i)
            If Left(.Name, 4) <> "MSys" Then
                Debug.Print .Name
            End If
        End With

    Next i
End Sub

```

Zugriff auf Eigenschaften und Methoden

Eigenschaften

Wie bereits erwähnt, hat jedes Objekt eine Reihe von Eigenschaften und Methoden, über die sich das Objekt beeinflussen und steuern lässt. Viele der Eigenschaften sind Ihnen bereits vom Eigenschaftenfenster bekannt, das Sie im Formularentwurf aktivieren können. Alle hier aufgelisteten Eigenschaften lassen sich auch von VBA aus ansprechen. Anstelle des deutschen Namens müssen Sie allerdings die englische Bezeichnung verwenden.

Um beispielsweise den Titel eines Formulars mit dem Namen *Kunden* zu ändern, verwenden Sie die *Beschriftung*-Eigenschaft. Die englische Übersetzung lautet *Caption*:

```
Forms!Kunden.Caption = "Neuer Fenstertitel"
```



Wie Sie dem obigen Ausdruck entnehmen können, muss der Name der Eigenschaft durch einen Punkt von dem Verweis auf das Objekt getrennt werden. Die allgemeine Syntax bei der Verwendung von Eigenschaften lautet dementsprechend folgendermaßen:

```
<Objektverweis>.<Eigenschaftsname>
```

In dem obigen Beispiel wird der Fenstertitel des geöffneten Formulars *Kunden* abgeändert. Es wird also eine Zuweisung durchgeführt, bei der der Inhalt der *Caption*-Eigenschaft des Formulars mit einem neuen Inhalt versehen wird.

Neben dem Verändern einer Eigenschaft ist es ebenfalls möglich, den Inhalt einer Eigenschaft auszulesen, um diesen beispielsweise in einem Meldungsfenster anzuzeigen oder in einer Variablen zu speichern:

```
MsgBox Forms!Kunden.Caption  
Dim Titel As Variant  
Titel = Forms!Kunden.Caption
```

Der Zugriff auf die Eigenschaften von Steuerelementen erfolgt auf die gleiche Weise. Das folgende Beispiel ermittelt den Inhalt der *DfaultValue*-Eigenschaft (*Standardwert*-Eigenschaft) des Steuerelements *Name*, das sich im Formular *Kunden* befindet:

```
Dim MeinStandardwert As Variant  
MeinStandardwert = Forms!Kunden!Name.DfaultValue
```

Besonders häufig wird ein Verweis auf Steuerelemente verwendet, um den aktuellen Inhalt zu ermitteln. Wenn Sie beispielsweise herausfinden möchten, welcher Text sich gerade in dem Steuerelement *Name* des Formulars *Kunden* befindet, verwenden Sie die *Value*-Eigenschaft:

```
MsgBox Forms!Kunden!Name.Value
```

Access erlaubt beim Zugriff auf den Inhalt eines Steuerelements auch eine Kurzschreibweise, indem Sie die Eigenschaft *Value* einfach komplett auslassen. Der folgende Ausdruck hat also dasselbe Ergebnis:

```
MsgBox Forms!Kunden!Name
```

Diese Kurzschreibweise weist allerdings im Gegensatz zur ausführlichen Schreibweise einen wichtigen Unterschied auf: Wenn Access kein Steuerelement ausfindig machen kann, das den Namen *Name* besitzt, durchsucht es die Datenquelle des Formulars nach einem passenden Feldnamen. Wird Access hierbei fündig, gibt es den Inhalt dieses Feldes zurück.

In der Praxis wird dies oft dazu verwendet, den Inhalt eines Datenfeldes auszulesen, für das Sie kein Steuerelement angelegt haben.

Die *Value*-
Eigenschaft



Hinweis



Die Eigenschaften von Formularen, Berichten und Steuerelementen haben stets den Datentyp *Variant*. Wenn Sie also eine Zuweisung vornehmen oder den Inhalt einer Eigenschaft auslesen möchten, bietet sich hier die Verwendung des Datentyps *Variant* an. Bei anderen Datentypen nimmt Access – wenn möglich – automatisch eine Konvertierung vor. Beachten Sie, dass Eigenschaften ohne Inhalt den Wert *NULL* besitzen, der nur von Variablen des Datentyps *Variant* unterstützt wird!

Angaben für Eigenschaften

Jede Eigenschaft hat einen bestimmten Datentyp, der einem der Grunddatentypen von Access entspricht. Die Eigenschaft *Caption* zur Angabe der Formularbeschriftung hat beispielsweise den Datentyp *String*. Wenn Sie diese Eigenschaft von VBA verändern möchten, geben Sie also eine Zeichenkette an. Ob diese in einer Variablen des Datentyps *String* oder *Variant* abgelegt ist, ist hierbei unerheblich.

True und False



Andere Eigenschaften wie beispielsweise die Eigenschaft *Bearbeitungen zulassen* (englisch: *AllowEdits*) erlauben beim Formularentwurf die Auswahl zwischen *Ja* und *Nein*. Beim Zugriff von VBA aus werden diese beiden Zustände durch *True* und *False* angegeben. Der Datentyp ist in diesem Fall *Boolean*.

Neben Eigenschaften mit dem Datentyp *String* und *Boolean* gibt es eine Reihe von Eigenschaften, bei denen Access im Eigenschaftenfenster eine Liste mit zulässigen Einstellungen anzeigt. Diese Eigenschaften haben in der Regel den Datentyp *Integer*, wobei jede Einstellung einem bestimmten Zahlenwert entspricht.

Standardansicht-Eigenschaft



Ein Beispiel für solch eine Eigenschaft ist die *Standardansicht*-Eigenschaft (englisch: *DefaultView*) von Formularen. Hier sind beispielsweise die drei Einstellungen *Einzelnes Formular*, *Endlosformular* und *Datenblattansicht* möglich. Diese drei Einstellungen werden innerhalb von VBA über die drei Werte 0, 1 und 2 vorgenommen. Um also die Standardansicht des Formulars *Kunden* in *Datenblattansicht* zu ändern, geben Sie folgenden Ausdruck an:

```
Forms!Kunden.DefaultView = 2
```

Hinweis



Fast alle Eigenschaften können auch dann verändert werden, wenn sich das Formular nicht in der Entwurfs-, sondern der Formularansicht befindet. Sie können so beispielsweise den Titel eines Formulars jederzeit mithilfe der *Caption*-Eigenschaft ändern. Diese Änderungen werden allerdings nicht dauerhaft gespeichert. Wenn Sie das Formular also schließen und wieder öffnen, besitzt es wieder den Titel, den Sie beim Formularentwurf festgelegt haben. Um eine Änderung dauerhaft durchzuführen, öffnen Sie das Formular mithilfe der *OpenForm*-Methode des *DoCmd*-Objekts in der Entwurfsansicht, führen die Änderungen durch und schließen es anschließend wieder mittels der *Close*-Methode.

Einige Eigenschaften lassen sich ausschließlich in der Entwurfsansicht verändern. Dazu gehört beispielsweise die *Name*-Eigenschaft, mit der sich die Namen von Formularen, Berichten oder Steuerelementen bestimmen lassen. In der Formularansicht sind diese Eigenschaften schreibgeschützt und können daher nur ausgelesen werden.



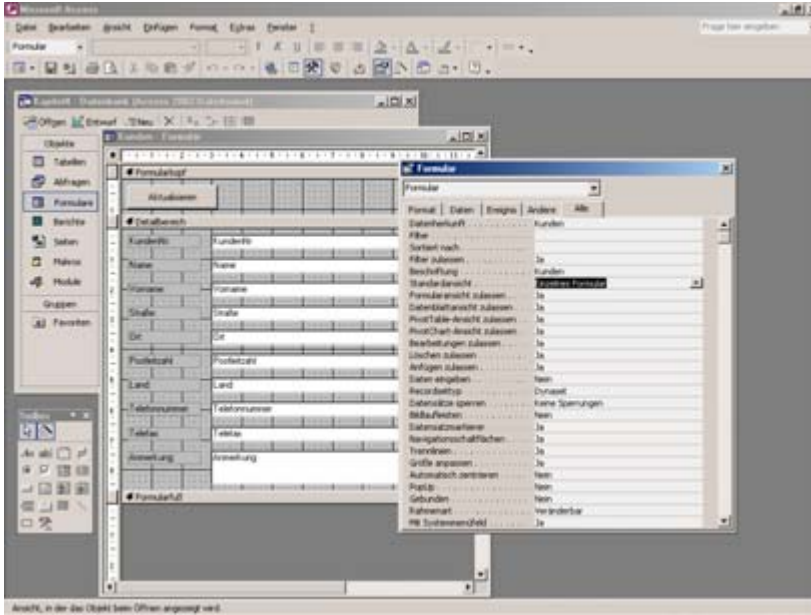


Abb. 4.6

Die Standardansicht-Eigenschaft eines Formulars

Methoden

Im Gegensatz zu Eigenschaften erteilen Methoden einen Befehl an ein Objekt, das daraufhin eine bestimmte Aufgabe erledigt. Eine Methode ist im Prinzip nichts anderes als eine Anweisung, die sich auf ein ganz bestimmtes Objekt bezieht. Ein Beispiel für eine solche Methode ist die *Requery*-Methode von Formularen, die das Formular dazu veranlasst, alle zugrunde liegenden Datensätze neu abzufragen und den ersten Datensatz anzuzeigen. Genau wie bei Eigenschaften werden auch die Namen von Methoden durch einen Punkt von dem Objektverweis getrennt:

<Objekt>.<Methode>

Um beispielsweise das Formular *Kunden* zu aktualisieren, kann der folgende Ausdruck verwendet werden:

Requery-Methode

Forms!Kunden.Requery

Diesen Ausdruck könnten Sie beispielsweise im Zusammenhang mit einer *Aktualisieren*-Schaltfläche einsetzen, die Sie auf einem Formular anlegen. Legen Sie einfach für die *Beim Klicken*-Ereigniseigenschaft eine Ereignisprozedur an und füllen Sie diese mit dem Ausdruck. Da sich die Ereignisprozedur in dem Formularmodul des Formulars *Kunden* befindet, können Sie den Ausdruck mithilfe der *Me*-Eigenschaft auch etwas abkürzen. Es folgt die komplette Ereignisprozedur:



Hinweis





```
Sub btnAktualisieren_Click ()
```

```
    Me.Requery
```

```
End Sub
```

Methoden mit
Parametern

Einige Methoden erwarten beim Aufruf einen oder mehrere Parameter und geben gleichzeitig ein Ergebnis zurück. Solche Methoden werden Sie im folgenden *Kapitel 5 Datenzugriff mit VBA* noch kennen lernen.

Die *With*-Anweisung

Wenn Sie nacheinander auf mehrere Eigenschaften oder Methoden eines Objekts zugreifen, ist es etwas lästig, jedes Mal den kompletten Objektverweis anzugeben. Das folgende Beispiel liest die Inhalte von vier Eigenschaften eines Steuerelements aus und zeigt diese im Testfenster an:

```
Debug.Print Forms!Kunden!Name.Visible  
Debug.Print Forms!Kunden!Name.DefaultValue  
Debug.Print Forms!Kunden!Name.Locked  
Debug.Print Forms!Kunden!Name.Enabled
```

Übersichtlicher wird dies mithilfe der *With*-Anweisung, die folgende Syntax hat:

```
With <Objekt>
```

```
    <Anweisungen>
```

```
End With
```

Bei Verwendung der *With*-Anweisung muss das Objekt bzw. der Objektverweis nur ein einziges Mal angegeben werden. Innerhalb des *With*-Blocks genügt es dann, lediglich den Punkt gefolgt von dem Namen der Eigenschaft oder der Methode anzugeben:

```
With Forms!Kunden!Name
```

```
    Debug.Print .Visible  
    Debug.Print .DefaultValue  
    Debug.Print .Locked  
    Debug.Print .Enabled
```

```
End With
```

Wenn Sie nun nachträglich einmal eine Änderung an dem Objektnamen *Forms!Kunden!Name* vornehmen, müssen Sie die Änderung nur an einer einzigen Stelle durchführen.



With-Anweisungen lassen sich beliebig ineinander verschachteln. Es ist also zulässig, innerhalb eines *With*-Blocks einen weiteren *With*-Block anzuordnen, der sich auf ein anderes Objekt bezieht. Eigenschaften und Methoden beziehen sich hierbei immer auf den zuletzt eingeleiteten *With*-Block.



Hinweis

Wenn Sie Programmcode innerhalb eines Objektmoduls anordnen, legt Access automatisch einen speziellen *With*-Block an, der Ihnen den Zugriff auf das aktuelle Formular bzw. den aktuellen Bericht ermöglicht. Dieser *With*-Block erscheint allerdings nicht im Programmcode.



Eigenschaften und Methoden in Objektmodulen

Wenn Sie beispielsweise vom Objektmodul des Formulars *Kunden* auf die *Sichtbar*-Eigenschaft (englisch *Visible*) zugreifen möchten, können Sie einen der drei folgenden Ausdrücke angeben:

```
Forms!Kunden.Visible  
Visible  
Me.Visible
```

In der Praxis wird häufig die letzte Variante verwendet, da die zweite (ohne Angabe eines Objekts) leicht mit einer Variablen verwechselt werden kann. Der Programmcode wird dadurch unübersichtlich. Die erste Variante sollten Sie auf jeden Fall vermeiden, dass Sie hierbei im Programmcode den Namen des Formulars angeben müssen. Wenn Sie diesen nachträglich ändern, müssen Sie den kompletten Programmcode überarbeiten.

Objektvariablen

In den vorhergehenden Abschnitten haben Sie mehrere Möglichkeiten kennen gelernt, um auf Formulare, Berichte und Steuerelemente Bezug zu nehmen. Alle Verweise auf Formulare, Berichte und Steuerelemente hatten hierbei Objekte des Typs *Form*, *Report* und *Control* zum Ergebnis. Die Bezeichnungen dieser Objekte können Sie auch direkt innerhalb von VBA als so genannte *Objektdatentypen* oder *Objektklassen* verwenden. Mithilfe des Objektdatentyps *Form* können Sie beispielsweise eine Variable deklarieren, die einen Verweis auf ein bestimmtes Formular speichert. Solche Variablen werden auch als *Objektvariablen* bezeichnet.



Die Set-Anweisung

Genau wie bei der Deklaration "normaler" Variablen können Objektvariablen mithilfe der Anweisungen *Dim*, *Private* und *Public* deklariert werden. Darüber hinaus gelten für Objektvariablen exakt die gleichen Regeln wie für "normale" Variablen. Dies betrifft beispielsweise die Variablennamen, das Lokalitätsprinzip sowie den Geltungsbereich.

Das folgende Beispiel deklariert zunächst eine Objektvariable mit dem Namen *MeinFormular*. Der Datentyp ist in diesem Fall *Form*, wodurch die Variable also eine Referenz auf ein Formular speichern kann:

```
Dim MeinFormular As Form  
Set MeinFormular = Forms!Kunden
```





Zuweisen einer Formularreferenz

In der zweiten Programmzeile wird dieser Objektvariablen nun eine Referenz auf das geöffnete Formular *Kunden* zugewiesen. Dies geschieht allerdings nicht – wie bei "normalen" Variablen üblich – nur mithilfe des Zuweisungsoperators =, sondern unter Zuhilfenahme der *Set*-Anweisung. Die *Set*-Anweisung muss stets verwendet werden, wenn einer beliebigen Objektvariablen eine Referenz auf ein bestimmtes Objekt zugewiesen werden soll.

Nach der Zuweisung kann die Variable synonym für den kompletten Verweis auf das Objekt verwendet werden. Die folgenden paarweisen Ausdrücke sind also immer gleichwertig:

```
Forms!Kunden.Caption = "Ein neuer Formulartitel"
MeinFormular.Caption = "Ein neuer Formulartitel"
Forms!Kunden.Requery
MeinFormular.Requery
MsgBox Forms!Kunden!Ort
MsgBox MeinFormular!Ort
```

Auf die gleiche Weise lässt sich auch eine Referenz auf ein Steuerelement in einer Objektvariablen des Typs *Control* speichern:

```
Dim MeinSteuerelement As Control

Set MeinSteuerelement = Forms!Kunden!Ort
```

Auch hier kann die Variable wieder synonym für den kompletten Verweis auf das Steuerelement verwendet werden.

Hinweis



Eine Objektvariable, der (noch) keine Referenz auf ein existierendes Objekt zugewiesen wurde, ist ungültig. Die einzige zulässige Operation mit einer solchen "leeren" Objektvariablen ist die Zuweisung mithilfe der *Set*-Anweisung.

Parameter

Ein großer Vorteil von Objektvariablen besteht darin, dass sie als Parameter an *Sub*- oder *Function*-Prozeduren übergeben werden können. Die folgende *Sub*-Prozedur schließt beispielsweise das Formular, das ihr mittels des Parameters *EinFormular* übergeben wurde:

```
Sub FormularSchließen (EinFormular As Form)

    DoCmd.Close acForm, EinFormular.Name

End Sub
```

Zum Schließen des Formulars verwendet die Prozedur die *Close*-Aktion. Als ersten Parameter erwartet die Aktion den Typ des Objekts, das geschlossen werden soll. Da es sich in diesem Fall um ein Formular handelt, ist hier die Konstante *acForm* angegeben. Als zweiten Parameter erwartet die Aktion den Namen des zu schließenden Formulars. Dieser wird mithilfe der *Name*-Eigenschaft aus dem übergebenen *Form*-Objekt ermittelt.



Um die Prozedur nun zu starten, rufen Sie diese einfach mit einem Verweis auf ein Formular auf:

```
FormularSchließen Forms!Kunden
```

Wenn Sie die Prozedur im Formularmodul des Formulars *Kunden* einsetzen, können Sie wieder die *Me*-Eigenschaft verwenden:

```
FormularSchließen Me
```

Spezielle Objektdatentypen

Eine *Form*-Objektvariable kann eine Referenz auf jedes beliebige Formular speichern. Access unterstützt allerdings auch Objektdatentypen, die speziell auf ein bestimmtes Formular zugeschnitten sind. Solche Objektdatentypen beginnen mit *Form_* und enden mit dem Namen des Formulars. Das folgende Beispiel deklariert eine Objektvariable, die nur einen Verweis auf das Formular *Kunden* speichern kann:

```
Dim MeinKundenformular As Form_Kunden
```

Anschließend kann der Variablen das Formular zugewiesen werden:

```
Set MeinKundenformular = Forms!Kunden
```

Auf den ersten Blick macht es wenig Sinn, eine Objektvariable auf ein bestimmtes Formular zu beschränken. Im *Kapitel 9 Objektorientierte Programmierung* werden Sie jedoch noch erfahren, wie Sie mehrere Instanzen eines Formulars öffnen. In der Variablen *MeinKundenformular* könnten Sie dann beispielsweise Verweise auf unterschiedliche Instanzen des Kundenformulars ablegen.

Bei Berichten beginnt der Name der Objektvariablen mit der Zeichenfolge *Report_*.

Ähnlich wie der Objektdatentyp *Form* Verweise auf ein beliebiges Formular aufnimmt, kann der Objektdatentyp *Control* einen Verweis auf ein beliebiges Steuerelement darstellen. Wenn Sie eine Objektvariable anlegen möchten, die nur auf einen bestimmten Steuerelementtyp verweisen kann, verwenden Sie einen der folgenden Objektdatentypen:



Hinweis



**Objektdatentypen
für Steuerelemente**

Objektdatentypen zum Verweis auf Steuerelemente

Objektdatentyp	Steuerelementtyp
<i>Label</i>	Bezeichnungsfeld
<i>TextBox</i>	Textfeld
<i>OptionGroup</i>	Optionsgruppe



Objektdatentypen zum Verweis auf Steuerelemente (Forts.)

Objektdatentyp	Steuerelementtyp
<i>ToggleButton</i>	Umschaltfläche
<i>OptionButton</i>	Optionsfeld
<i>CheckBox</i>	Kontrollkästchen
<i>ComboBox</i>	Kombinationsfeld
<i>ListBox</i>	Listenfeld
<i>CommandButton</i>	Befehlsschaltfläche
<i>Image</i>	Bild
<i>ObjectFrame</i>	Objektfeld
<i>BoundObjectFrame</i>	Gebundenes Objektfeld
<i>PagePreak</i>	Seitenumbruch
<i>SubForm</i>	Unterformular
<i>SubReport</i>	Unterbericht
<i>Line</i>	Linie
<i>Rectangle</i>	Rechteck
<i>TabControl</i>	Register
<i>Page</i>	Registerseite
<i>Section</i>	Bereich

Tabelle 4.6

Das folgende Beispiel deklariert eine Variable, die nur ein Listenfeld aufnehmen kann:

```
Dim EinListenfeld As ListBox
```

Solche Objektdatentypen werden in der Regel bei der Deklaration von Parametervariablen in Prozeduren verwendet. Da ein Listenfeld andere Eigenschaften und Methoden unterstützt als andere Steuerelemente, können Sie beim Aufruf der Prozedur sicherstellen, dass wirklich ein Listenfeld übergeben wurde. Wenn dies nicht der Fall ist, tritt ein Laufzeitfehler auf.





Ein weiterer Objektdatentyp mit dem Namen *Object* hat den größten Einsatzbereich. Eine Variable, die mit diesem Objektdatentyp deklariert wurde, kann beliebige Objekte – also Formulare, Berichte oder Steuerelemente – speichern. Darüber hinaus werden auch alle Datenzugriffsobjekte und sogar OLE-Automationsobjekte anderer Anwendungen unterstützt, die in Access eingebunden werden. Mehr dazu erfahren Sie im *Kapitel 13 OLE und DDE*.

Der *Object*-
Datentyp

Auflistungen

Da Auflistungen nichts anderes als normale Objekte mit speziellem Aufgabenbereich sind, stellen sie ebenfalls Methoden und Eigenschaften zur Verfügung. Die einzige und gleichzeitig auch wichtigste Eigenschaft von Auflistungen ist die *Count*-Eigenschaft, mit der sich die Anzahl der Objekte innerhalb einer Auflistung ermitteln lässt. Das folgende Beispiel ermittelt die Anzahl der *Form*-Objekte innerhalb der *Forms*-Auflistung und gibt diese im Testfenster aus. Das Ergebnis ist also die Anzahl der geöffneten Formulare:

Die *Count*-
Eigenschaft

```
Sub AnzahlFormulare()  
  
    Debug.Print Forms.Count  
  
End Sub
```

Das Gleiche funktioniert auch mit Berichten, wenn Sie die Anzahl der (in der Seitenansicht oder der Entwurfsansicht) geöffneten Berichte ermitteln möchten:

```
Debug.Print Reports.Count
```

Das folgende Beispiel nutzt die *Count*-Eigenschaft der *Forms*-Auflistung, um die Namen aller geöffneten Formulare im Testfenster anzuzeigen:

```
Sub FormulareAuflisten ()  
  
    Dim Anzahl As Integer  
    Dim i As Integer  
  
    Anzahl = Forms.Count  
  
    For i = 0 To Anzahl - 1  
  
        Debug.Print Forms(i).Name  
  
    Next i  
  
End Sub
```





Auf die einzelnen *Form*-Objekte wird hierbei über die Position der Objekte innerhalb der *Forms*-Auflistung verwiesen. Beachten Sie hierbei, dass das erste Formular die Positionsnummer 0 enthält. Die *For*-Schleife zählt aus diesem Grund nur bis zu dem Wert *Anzahl -1*.

Eine ähnliche Prozedur eignet sich zur Überprüfung, ob ein bestimmtes Formular geöffnet ist. Die Prozedur erwartet dazu eine Zeichenkette, die den Namen des zu überprüfenden Formulars enthält. Wenn das Formular geöffnet ist, gibt die Funktion den logischen Wert *True* zurück. Ansonsten ist das Ergebnis *False*:

```
Function IstGeöffnet2(Formularname As String) As Boolean

Dim i As Integer

    For i = 0 To Forms.Count - 1

        If Forms(i).Name = Formularname Then

            IstGeöffnet2 = True
            Exit Function

        End If

    Next i

    IstGeöffnet2 = False

End Function
```

Hinweis



Wenn sich das gesuchte Formular innerhalb der *Forms*-Auflistung an letzter Position befindet, müssen alle *Form*-Objekte durchlaufen werden. Da dies Zeit kostet, stellt Access eine spezielle Funktion mit dem Namen *SysCmd* zur Verfügung, die diese Aufgabe schneller erledigt. Die folgende Prozedur zeigt den Aufbau der verbesserten *IstGeöffnet*-Funktion:

```
Function IstGeöffnet(Formularname As String) As Boolean

IstGeöffnet = IIf(SysCmd(acSysCmdGetObjectState, _
                    acForm, Formularname) = acObjStateOpen, _
                    True, False)

End Function
```

Steuerelemente



Mithilfe der *Count*-Eigenschaft der *Controls*-Auflistung lässt sich auch die Anzahl der Steuerelemente bestimmen, die in einem bestimmten Formular enthalten sind:

```
Forms!Kunden.Controls.Count
```



Eine Auflistung der einzelnen Steuerelementnamen im Testfenster ist hiermit ebenfalls möglich:

```
Sub SteuerelementeAuflisten(MeinFormular As Form)

Dim i As Integer

    For i = 0 To MeinFormular.Controls.Count - 1

        Debug.Print MeinFormular.Controls(i).Name

    Next i

End Sub
```

Beim Start der Prozedur muss lediglich ein Verweis auf das geöffnete Formular angegeben werden. Das folgende Beispiel zeigt zwei Möglichkeiten zum Aufruf der Prozedur:

```
SteuerelementeAuflisten Forms!Kunden
Dim MeinFormular As Form

Set MeinFormular = Forms!Kunden
SteuerelementeAuflisten MeinFormular
```

Die *For Each...Next*-Schleife

In der Praxis ist es häufig erforderlich, dass alle Objekte einer Auflistung nacheinander durchlaufen werden sollen. Die Verwendung einer *For...Next*-Schleife ist hierbei nicht sehr komfortabel, da immer eine Zählervariable deklariert werden muss. In den Beispielen des vorhergehenden Abschnitts war dies die Variable *i*.

Zur Verbesserung haben die Access-Entwickler eine spezielle Schleife entwickelt, die eng mit der *For...Next*-Schleife verwandt ist:

```
For Each <Element> In <Auflistung>

    [<Anweisungsblock>]

Next [<Element>]
```

Unter *<Auflistung>* kann der Name einer beliebigen Auflistung angegeben werden. *<Element>* ist ein Variable, die vom Datentyp mit den einzelnen Elementen der Auflistung übereinstimmt. Beim Durchlauf der Schleife werden der Reihe nach die einzelnen Objekte der Auflistung der Variablen *<Element>* zugewiesen.



Auflisten aller Formulare

Das folgende Beispiel listet wieder die Namen aller geöffneten Formulare auf. Diesmal wird allerdings eine *For Each*-Schleife verwendet:

```
Sub FormulareAuflisten()  
  
Dim EinFormular As Form  
  
    For Each EinFormular In Forms  
  
        Debug.Print EinFormular.Name  
  
    Next EinFormular  
  
End Sub
```

In dem Beispiel wird zunächst eine Objektvariable namens *EinFormular* mit dem Objekttyp *Form* deklariert. Innerhalb der *For Each*-Schleife wird nun bei jedem Durchlauf der Variablen *EinFormular* ein anderes Formular zugewiesen, dessen Name im Testfenster ausgegeben wird.

Der Objektkatalog



Bei der Vielzahl der Objekte, Eigenschaften, Methoden und Ereignisse, die Access zur Verfügung stellt, kann schnell die Übersicht verloren gehen. Abhilfe schafft der Objektkatalog. Sie öffnen den Objektkatalog vom Modulfenster aus über den Menüpunkt *Ansicht > Objektkatalog*, das gleichnamige Symbol in der Symbolleiste oder die Taste **[F2]**.

Abb. 4.7

Der Objektkatalog



Der Einsatz des Objektkatalogs ist hierbei nicht auf die von Access unterstützten Objekte beschränkt. Bei Bedarf lassen sich die Objekte anderer Windows-Anwendungen wie beispielsweise Word oder Excel in den Objektkatalog einbinden. Dies ist von großem Vorteil, wenn Sie applikationsübergreifende Anwendungen entwickeln und hierbei Access als Steuerzentrale einsetzen. Mehr Informationen zu diesem Thema erhalten Sie im *Kapitel 13 OLE und DDE*.

Die Objekte einer bestimmten Windows-Anwendung sind in so genannten *Bibliotheken* zusammengefasst. So setzt sich Access beispielsweise aus den folgenden Bibliotheken zusammen:

Auswahl der gewünschten Bibliothek

-  Access
-  Datenzugriffsobjekte *DAO* (**D**ata **A**ccess **O**bjects) oder *ADO* (**A**ctive(X) **D**ata **O**bjects)
-  *VBA* (**V**isual **B**asic for **A**pplications)

Die Bibliothek *Access* enthält alle Objekte, die z.B. für den Zugriff auf Formulare oder Steuerelemente zuständig sind. Die *Datenzugriffsobjekte*-Bibliothek enthält dagegen alle Objekte, die für den Zugriff auf Daten erforderlich sind. Dazu gehören beispielsweise *Recordset*-Objekte. Die letzte Bibliothek *VBA* enthält dagegen die allgemeinen Elemente der Programmiersprache VBA, wie z.B. Zeichenketten- und Datumsfunktionen.

Nachdem Sie den Objektkatalog geöffnet haben, sind standardmäßig alle drei Bibliotheken aktiv. Sie erkennen dies an dem Eintrag *<Alle Bibliotheken>* im Kopf des Objektkatalogs. Bei Bedarf können Sie auch nur eine einzelne Bibliothek aktivieren, indem Sie diese aus dem Kombinationsfeld auswählen.

Zusätzlich zu den drei Einträgen finden Sie in der Liste des Kombinationsfeldes einen weiteren Eintrag, der mit dem Namen Ihrer Datenbank übereinstimmt. Mit diesem Eintrag erkunden Sie die einzelnen Formulare, Berichte und Module Ihrer Datenbank. Der Objektkatalog listet hierbei nicht nur die von Access vorgegebenen Elemente, sondern auch Ihre selbst erstellten Methoden und Eigenschaften auf.

Um dem Objektkatalog die Bibliotheken fremder Access-Anwendungen hinzuzufügen, aktivieren Sie diese im Verweisdialog (Menüpunkt *Extras > Verweise*). Mehr dazu erfahren Sie im *Kapitel 13 OLE und DDE*.



Hinweis

Objekte begutachten

Das Fenster des Objektkatalogs ist in zwei Listen mit den Beschriftungen *Klassen* und *Elemente von* unterteilt. Die linke Liste enthält alle Klassen der momentan ausgewählten Bibliothek in alphabetischer Reihenfolge. Sollten Sie den Eintrag *<Alle Bibliotheken>* ausgewählt haben, enthält die Liste alle Objektklassen, die Sie derzeit innerhalb von Access nutzen können. Dazu gehören beispielsweise Klassen wie *Form*, *Recordset* und *Control*.



Die einzelnen Klassen sind mit Symbolen versehen, die Ihnen die Unterscheidung der einzelnen Klassentypen erleichtern. Über die Bedeutung dieser Symbole informiert Sie die folgende Tabelle:

Klassentypen

Symbol	Elementtyp
	Modul
	Klasse
	Typ
	Aufzählung von Konstanten

Tabelle 4.7

Wie bereits erwähnt, listet Access alle Einträge in alphabetischer Reihenfolge auf. Das Kontextmenü der Liste stellt jedoch den Menüpunkt *Elemente gruppieren* zur Verfügung, mit dem Sie die Liste bei Bedarf nach dem Klassentyp gruppieren können.

Klasse analysieren

Die Hauptaufgabe des Objektkatalogs ist, die einzelnen Elemente eines Objekts aufzulisten. Diese Aufgabe hat die rechte Elementliste. Sobald Sie in der Klassenliste eine bestimmte Klasse markiert haben, listet Access in der Elementliste alle Klassenelemente wie Eigenschaften, Methoden, Ereignisse und Konstanten auf. Zu besseren Unterscheidung sind diese mit Symbolen versehen:

Elemente von Klassen

Symbol	Elementtyp
	Eigenschaft
	Methode
	Ereignis
	Konstante

Tabelle 4.8



Standardelemente wie z. B. Standardauflistungen sind zusätzlich mit einem kleinen blauen Punkt gekennzeichnet.

Der Eintrag *<Global>* der Klassenliste zeigt alle Elemente an, die innerhalb von Access unabhängig von einer bestimmten Klasse zur Verfügung stehen. Dies sind beispielsweise allgemeine Funktionen oder Konstanten.



Hinweis

Unterhalb beider Listen zeigt Access im Informationsfenster zusätzliche Informationen zur ausgewählten Klasse oder zum ausgewählten Element an. Bei Eigenschaften erscheint hier beispielsweise der Datentyp. Genauere Informationen zu einem Element erhalten Sie, indem Sie den Befehl *Hilfe* aus dem Kontextmenü auswählen. Access zeigt daraufhin die passende Seite der Onlinehilfe an.



Informationsfenster

Einige Texte im Informationsfenster erscheinen in grüner Farbe. Mit einem Klick auf einen solchen Eintrag aktivieren Sie im Objektkatalog – falls möglich – die angeklickte Klasse bzw. Bibliothek.

Wenn Sie den Namen einer Klasse oder eines Elements als Text in ein Modul übernehmen möchten, wählen Sie aus dem Kontextmenü den Befehl *Kopieren* aus. Alternativ klicken Sie auf das gleichnamige Symbol im Kopf des Objektkatalogs. Von der Zwischenablage aus fügen Sie den Text nun in Ihr Modul ein. Sollte es sich bei dem gewünschten Element um eine in VBA entwickelte Methode oder Eigenschaft handeln, können Sie mit dem Befehl *Definition anzeigen* des Kontextmenüs den betreffenden VBA-Programmcode anzeigen lassen. Im Kopf des Fensters steht Ihnen alternativ ein entsprechendes Symbol zur Verfügung.



Hinweis

Alle von Ihnen ausgewählten oder markierten Einträge der Klassen- und Elementliste speichert der Objektkatalog in einer internen Verlaufsliste. Innerhalb dieser Liste blättern Sie mithilfe der beiden Pfeilschaltflächen im Kopf des Objektkatalogs. Wenn Sie beispielsweise die Klasse *Application* und anschließend die Klasse *Screen* aktivieren, gelangen Sie mit dem linken Pfeilsymbol wieder zur Klasse *Application*. Diese Art der Navigation ist besonders bei umfangreichen Klassen- und Elementlisten hilfreich.



Blättern im Objektkatalog



Elemente suchen

Der Objektkatalog stellt eine Suchfunktion zur Verfügung, mit der Sie auf einfache Weise eine Klasse oder ein Element ausfindig machen können. Unterhalb des Kombinationsfeldes zur Auswahl der Bibliothek finden Sie zu diesem Zweck ein Textfeld. Hier geben Sie als Suchbegriff den Teil eines Klassen- oder Elementnamens ein.

Sie starten die Suche mit einem Klick auf das *Suchen*-Symbol bzw. über die Taste . In der Liste *Suchergebnis* erscheinen daraufhin alle gefundenen Klassen bzw. Elemente. Mit einem Doppelklick auf einen Eintrag dieser Liste wählt Access diesen in der Klassen- oder Elementliste zur genaueren Begutachtung aus.



Sie können das Suchergebnis wahlweise ein- oder ausblenden, um den Platz des Objektkatalogs optimal auszunutzen. Im Kopf des Katalogs steht Ihnen zu diesem Zweck ein entsprechendes Symbol zur Verfügung.



Hinweis





Datenzugriff mit VBA



Schnellzugriff auf Daten	226
Datenzugriff mit den DAO-Objekten	229
Datenzugriff mit ADO	263
Einführung in SQL	284





Im vorhergehenden Kapitel haben Sie bereits eine Möglichkeit kennen gelernt, von VBA aus auf Daten zuzugreifen. Hierbei verweisen Sie einfach auf ein bestimmtes Steuerelement eines geöffneten Formulars und lesen den Inhalt aus. Auch das Verändern von Steuerelement- und Feldinhalten ist auf diese Weise möglich.

Bei reinen Datenmanipulationen ist diese Vorgehensweise allerdings etwas umständlich. Schließlich möchten Sie nicht immer erst ein Formular anlegen, um auf die Daten einer Tabelle zuzugreifen. Aus diesem Grund stellt Access eine Reihe von Objektklassen zur Verfügung, mit denen der direkte Datenzugriff möglich ist. Genauso wie Formulare, Berichte und Steuerelemente werden auch Tabellen oder Abfragen als Objekte angesehen, auf die der Zugriff – beispielsweise per Objektvariablen – möglich ist.

Access XP unterstützt zwei verschiedene Datenzugriffsmodelle: **DAO (Data Access Objects)** und **ADO (Active(X) Data Objects)**. In den Beispielen dieses Kapitels lernen Sie sowohl DAO als auch ADO kennen.

Bevor es in den Abschnitten *Datenzugriff mit den DAO-Objekten* und *Datenzugriff mit ADO* richtig losgeht, erfahren Sie im direkt nach dieser Einleitung folgenden Abschnitt *Schnellzugriff auf Daten* erst einmal etwas über die so genannten *Domänenfunktionen*, mit deren Hilfe sich einzelne Daten sehr leicht aus Tabellen abfragen lassen. Der vierte und letzte Abschnitt *Einführung in SQL* beschäftigt sich schließlich mit der Abfragesprache SQL.

Wie Sie die Struktur von Tabellen und Abfragen ändern oder sogar neue Tabellen und Abfragen anlegen, erfahren Sie im *Kapitel 10 Design von Datenbankobjekten*.

Alle Beispiele zu diesem Kapitel finden Sie in den Datenbanken *Kapitel5.mdb* und *Kapitel5_ADO.mdb*.

Schnellzugriff auf Daten

Die *DLookup*-
Funktion

Zum Ermitteln einzelner Daten aus Tabellen oder Abfragen stellt Access die so genannten *Domänenfunktionen* zur Verfügung. Ein typischer Vertreter dieser Funktionen ist die *DLookup*-Funktion, mit der ein einzelner Feldinhalt einer Tabelle oder einer Abfrage ausgelesen werden kann. Die Funktion hat folgende Syntax:

```
DLookup (<Expr>, <Domain> [, <Criteria>])
```

Alle drei Parameter dieser Funktion sind Zeichenketten. Unter *<Expr>* geben Sie den Namen des Feldes an, dessen Inhalt Sie ermitteln möchten. Die Datenquelle – also den Namen einer Tabelle oder einer Abfrage – legen Sie über den zweiten Parameter mit dem Namen *<Domain>* fest. Als Letztes muss nun noch der gewünschte Datensatz ausgewählt werden. Dazu geben Sie unter *<Criteria>* den *WHERE*-Teil einer SQL-Abfrage an. Das Schlüsselwort *WHERE* müssen Sie allerdings auslassen. Sollten mehrere Datensätze der angegebenen Bedingung entsprechen, wählt Access automatisch den ersten passenden Datensatz aus. Wenn Sie die Bedingung komplett auslassen, berücksichtigt Access den ersten Datensatz der Datenquelle.



Die folgende Tabelle enthält einige Beispiele für Zeichenketten, die Sie unter *<Criteria>* angeben können:

Beispiele für Bedingungen

Bedingung

KundenNr = 6

Name = "Müller"

*Name Like "M*er"*

Umsatz > 20000

Umsatz > 10000 AND Umsatz < 20000

Tabelle 5.1

Das Ergebnis der *DLookup*-Funktion ist vom Datentyp *Variant*. Sollte der betreffende Datensatz nicht gefunden werden, gibt die Funktion den Wert *NULL* zurück. Das Gleiche gilt, wenn der Datensatz zwar gefunden wurde, das angegebene Feld aber keinen Inhalt hat.

Der folgende Ausdruck ermittelt den Namen des Kunden mit der *KundenNr* 6 und gibt ihn in einem Meldungsfenster aus:

```
MsgBox DLookup("Name", "Kunden", "KundenNr = 6")
```

Wenn innerhalb der Bedingung Zeichenketten auftauchen, müssen diese in einfachen Anführungszeichen angegeben werden. Die folgende Anweisung gibt beispielsweise die *KundenNr* des ersten Kunden mit dem Namen *Müller* aus:

```
MsgBox DLookup("KundenNr", "Kunden", "Name = "Müller"")
```

Alternativ können anstelle des einfachen Anführungszeichens auch normale Anführungszeichen angegeben werden. Dies kann allerdings schnell unübersichtlich werden:

```
MsgBox DLookup("KundenNr", "Kunden", "Name =  
"Müller"")
```

In den obigen Beispielen wurden für das Argument *<Expr>* immer nur einzelne Feldnamen ausgegeben. Access lässt aber auch die Angabe von Berechnungen und anderen Ausdrücken zu. Das folgende Beispiel gibt den *Namen* und den *Vornamen* des Kunden mit der *KundenNr* 6 aus:

```
MsgBox DLookup("Name & ", " & Vorname", "Kunden", _  
"KundenNr = 6")
```



Hinweis



Angabe von
Ausdrücken



In diesem Beispiel fügt der Ausdruck

"Name & ", " & Vorname"

die Inhalte der Felder *Name* und *Vorname* mit einem Komma als Trennzeichen aneinander.

Andere Domänen-
funktionen

Neben der *DLookup*-Funktion unterstützt Access eine Reihe weiterer Domänen-
funktionen:

Domänenfunktionen

Funktion	Beschreibung
<i>DAvg</i> -Funktion	Ermittelt den Durchschnittswert des angegebenen Feldes.
<i>DCount</i> -Funktion	Ermittelt die Anzahl der Datensätze.
<i>DFirst</i> -Funktion	Ermittelt den Inhalt des ersten Datensatzes.
<i>DLast</i> -Funktion	Ermittelt den Inhalt des letzten Datensatzes.
<i>DMax</i> -Funktion	Ermittelt den größten Wert des angegebenen Feldes.
<i>DMin</i> -Funktion	Ermittelt den kleinsten Wert des angegebenen Feldes.
<i>DStDev</i> -, <i>DStdDevP</i> -Funktionen	Ermitteln die Standardabweichung des angegebenen Feldes.
<i>DSum</i> -Funktion	Ermittelt die Summe der Feldinhalte.
<i>DVar</i> -, <i>DVarP</i> -Funktionen	Berechnen die Varianz des angegebenen Feldes.

Tabelle 5.2

Alle Domänenfunktionen haben die gleiche Syntax wie die *DLookup*-Funktion:

<Funktionsname> (<Expr>, <Domain> [, <Criteria>])

Der einzige Unterschied besteht in der Interpretation der unter <Criteria> angegebenen Bedingung. Wurde bei der *DLookup*-Funktion immer nur ein einzelner Datensatz ausgewählt, berücksichtigen die in der Tabelle aufgeführten Funktionen jeweils alle Datensätze, die der angegebenen Bedingung entsprechen. Eine Ausnahme sind hier lediglich die Funktionen *DFirst* und *DLast*.

Die *DSum*-Funktion ermöglicht so z. B. die Summenberechnung eines Datenfeldes, wobei nur die zu der Bedingung passenden Datensätze berücksichtigt werden.



Eine Besonderheit muss bei der *DCount*-Funktion für die Berechnung der Datensatzanzahl beachtet werden. Access zählt hierbei nämlich nur die Datensätze, deren unter *<Expr>* angegebenes Datenfeld *nicht* den Inhalt *NULL* aufweist. Wenn Sie also die Gesamtzahl ermitteln möchten, geben Sie hier am besten den Namen des Primärschlüsselfeldes an, dessen Inhalt nie *NULL* ist.

Datenzugriff mit den DAO-Objekten

Standardmäßig sind in Access XP die ADO-Objekte aktiviert. Um die Beispiele dieses Abschnitts nachvollziehen zu können, verwenden Sie die Beispieldatenbank *Kapitel5.mdb*. Wenn Sie eine neue Datenbank anlegen, gehen Sie folgendermaßen vor:

- Legen Sie ein neues Modul an.
- Wählen Sie den Menüpunkt *Extras > Verweise* der VBA-Entwicklungsumgebung aus.
- Entfernen Sie die Markierung des Eintrags *Microsoft ActiveX Data Object 2.1 Library*.
- Markieren Sie den Eintrag *Microsoft DAO 3.6 Object Library*.
- Bestätigen Sie mit *OK*.
- Anschließend sind die DAO-Objekte aktiviert.

Die DAO-Objekthierarchie für den Datenbankzugriff ist etwas komplexer als die von Formularen und Steuerelementen. Wenn Sie eine Datenbank betrachten, kann diese mehrere Tabellen und Abfragen enthalten. Jede Tabelle enthält mehrere Datenfelder und Indizes. Jedem Index ist wiederum ein Datenfeld zugewiesen, welches das zu indizierende Feld bestimmt.

Die *Abbildung 5.1* enthält die Objekte, die für den DAO-Datenzugriff mit Access relevant sind.

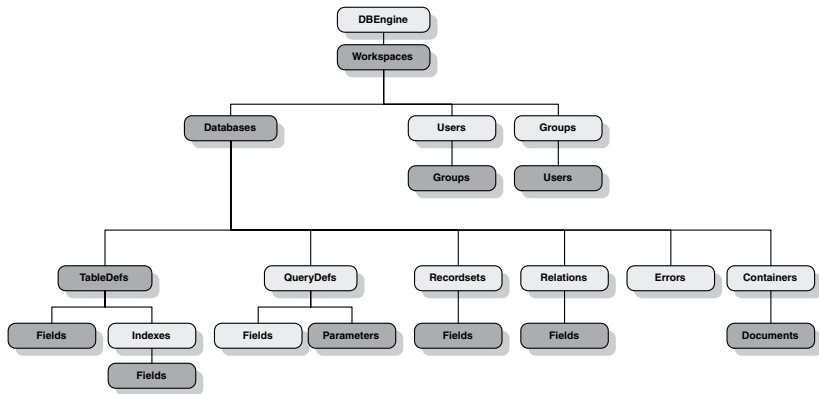
Bei den Objekten dieser grafisch dargestellten Hierarchie handelt es sich mit Ausnahme des *DBEngine*-Objekts um *Auflistungen*, die Sie bereits im vorhergehenden Kapitel im Zusammenhang mit Formularen und Berichten kennen gelernt haben. So erlaubt beispielsweise die *TableDefs*-Auflistung den Zugriff auf die Struktur aller Tabellen der aktuellen Datenbank.

An oberster Stelle befindet sich das *DBEngine*-Objekt. Dieses Objekt ist eine Ausnahme, da es als Wurzel der Hierarchie keine Auflistung, sondern ein einzelnes, von Access fest vorgegebenes Objekt ist. Das Objekt repräsentiert hierbei die JET-Datenbank-Engine (*JET = Joint Engine Technology*) von Access, die für die komplette Verwaltung aller Daten verantwortlich ist. Dazu gehören beispielsweise Tabellen und Abfragen.



Abb. 5.1

Die DAO-Objekthierarchie



Die Datenbank-Engine stellt mehrere *Workspace*-Objekte zu Verfügung, deren Zugriff die *Workspaces*-Auflistung erlaubt. Ein *Workspace*-Objekt ist eine Arbeitsgruppe, in der das Sicherheitssystem von Access zusammen mit Benutzern und Benutzergruppen verwaltet wird. Sie lernen *Workspace*-Objekte im Kapitel 12 Zugriff auf das Sicherheitssystem noch ausführlicher kennen.

Database-Objekte

Die nächste Ebene stellen die *Database*-Objekte dar, die jeweils eine bestimmte Datenbank repräsentieren. Da es von VBA aus möglich ist, mehrere Datenbanken gleichzeitig zu öffnen, können sich in der zugehörigen *Databases*-Auflistung auch mehrere *Database*-Objekte befinden. An erster Position (Positionsnummer 0) befindet sich grundsätzlich immer die aktuelle Datenbank, mit der Sie gerade in Access arbeiten. Diese Datenbank wird auch als *Benutzerdatenbank* bezeichnet.

Ähnliches gilt für die aktuelle Arbeitsumgebung (*Workspace*-Objekt). Die aktuelle Arbeitsumgebung befindet sich in der *Workspaces*-Auflistung ebenfalls an der Positionsnummer 0. Um auf die aktuelle Benutzerdatenbank zuzugreifen, kann der folgende Ausdruck verwendet werden:

```
DBEngine.Workspaces(0).Databases(0)
```

Da diese Schreibweise viel Tipparbeit erfordert, erlaubt Access auch eine kürzere:

```
DBEngine(0)(0)
```

Bei dieser Schreibweise wird der Name der beiden Auflistungen *Workspaces* und *Databases* einfach ausgelassen. Dies ist möglich, da diese beiden Auflistungen jeweils die Standardauflistungen der übergeordneten Objekte sind. In Abbildung 5.1 sind alle diese Standardauflistungen dunkel eingefärbt.

CurrentDb

Noch kürzer wird der Zugriff auf die aktuelle Datenbank durch den Einsatz der Funktion *CurrentDb*. Hierbei handelt es sich um eine spezielle Abkürzung, die die DAO-Objekthierarchie teilweise umgeht. Der Vorteil ist, dass die Objekte einer Datenbank, auf der Sie mittels *CurrentDb*-Funktion verweisen, auf dem aktuellsten Stand sind. Dies ist bei Verwendung des *DBEngine(o)(o)*-Ausdrucks nicht immer der



Fall, deshalb wird in diesem Buch stets die *CurrentDB*-Funktion eingesetzt. Dies ist auch die offizielle Empfehlung der Firma Microsoft.

Database-Objekte

Zur Handhabung von *Database*-Objekten stellt Access einen Objektdatentyp mit dem Namen *Database* zu Verfügung. Einer mit diesem Datentyp deklarierten Variablen können Sie also mithilfe der *Set*-Anweisung eine Referenz auf eine Datenbank zuweisen:

```
Dim MeineDatenbank As Database
```

```
Set MeineDatenbank = DBEngine.Workspaces(0).Databases(0)
```

Mit der Objektvariablen *MeineDatenbank* können Sie nun innerhalb von Access arbeiten. Der folgende Ausdruck verwendet beispielsweise die *Name*-Eigenschaft des *Database*-Objekts, um den Dateinamen der Datenbank inklusive dem Dateipfad in einem Meldungsfenster auszugeben:

```
MsgBox MeineDatenbank.Name
```

Wie bereits erwähnt, lassen sich von VBA aus mehrere Datenbanken gleichzeitig öffnen. Zu diesem Zweck stellen *Workspace*-Objekte eine Methode mit dem Namen *OpenDatabase* zur Verfügung, die das Öffnen und Bereitstellen der Datenbank erledigt. Die Methode erwartet eine Zeichenkette als Parameter, die den Dateinamen und ggf. den Dateipfad der Datenbankdatei enthält. Das Ergebnis der Methode ist ein Objekt des Typs *Database*, das auf die geöffnete Datenbank verweist. Die folgende Beispielprozedur öffnet die Datenbank mit dem Namen *Nordwind.mdb*, die sich im Unterverzeichnis *Beispiele* des Access-Installationsverzeichnis befindet:

```
Sub DBTestprozedur1()
```

```
Dim MeineDatenbank As DATABASE
```

```
Set MeineDatenbank = DBEngine.Workspaces(0). _  
    OpenDatabase("C:\Programme\Microsoft _  
    Office\Office\Beispiele\Nordwind.MDB")
```

```
End Sub
```

Wenn Sie beim Öffnen der Datenbank keinen Pfad angeben, durchsucht Access das aktuelle Verzeichnis nach der Datenbank. Dieses können Sie mithilfe der *CurDir*-Funktion ermitteln und mit den Anweisungen *ChDir* bzw. *ChDrive* verändern.

Die geöffnete Datenbank wird von Access automatisch der *Databases*-Auflistung hinzugefügt, so dass nun auch hierüber ein Verweis auf diese Datenbank möglich ist. Die folgende Beispielprozedur öffnet wieder die Beispieldatenbank *Nordwind.mdb* und gibt anschließend eine Liste aller geöffneten Datenbanken (also auch der Benutzerdatenbank) nacheinander in mehreren Meldungsfenstern aus:

OpenDatabase-
Methode



Hinweis





```
Sub DBTestprozedur2()  
  
Dim EineDatenbank As DATABASE  
Dim EineArbeitsumgebung As Workspace  
Dim AnzahlDatenbanken As Integer  
Dim i As Integer  
  
Set EineArbeitsumgebung = DBEngine.Workspaces(0)  
  
Set EineDatenbank = EineArbeitsumgebung.OpenDatabase _  
    ("C:\ACCESS\Beispiele\Nordwind.mdb")  
  
AnzahlDatenbanken = _  
    EineArbeitsumgebung.Databases.Count  
  
For i = 0 To AnzahlDatenbanken - 1  
  
    MsgBox EineArbeitsumgebung.Databases(i).Name  
  
Next i  
  
End Sub
```

Schließen einer Datenbank

Access schließt die geöffnete Datenbank automatisch, wenn die Lebensdauer der betreffenden *Database*-Variablen beendet ist. Bei lokalen Variablen ist dies beispielsweise bei der Beendigung der Prozedur der Fall. In der obigen Beispielprozedur *DBTestprozedur2* wird die Datenbank *Nordwind.mdb* also nach Abarbeitung der Schleife wieder geschlossen und damit aus der *Databases*-Auflistung entfernt.

Das Schließen der Datenbank kann allerdings auch vorzeitig erfolgen, indem Sie die *Close*-Methode auf das *Database*-Objekt anwenden:

```
EineDatenbank.Close
```

Das vorzeitige Schließen dient vor allem dazu, die von der Datenbank belegten Systemressourcen wieder freizugeben. Die Methode wird daher oft bei globalen *Database*-Variablen angewendet, die nicht mehr benötigt werden.

Eine Datenbank wird ebenfalls automatisch geschlossen, wenn Sie der betreffenden Objektvariablen mithilfe der *OpenDatabase*-Methode einfach eine andere Datenbank zuweisen.

Hinweis



Beachten Sie, dass Sie die *Close*-Methode nicht auf ein *Database*-Objekt anwenden können, das auf die aktuelle Benutzerdatenbank verweist. Das Schließen der Benutzerdatenbank ist nur mithilfe der Methode *CloseCurrentDatabase* des *Application*-Objekts möglich:

```
Application.CloseCurrentDatabase
```



Recordset-Objekte

Der Zugriff auf die Daten einer Datenbank erfolgt innerhalb von VBA mithilfe von *Recordset*-Objekten. Da diese Objekte nichts anderes als eine Gruppe von Datensätzen repräsentieren, werden sie auch als *Datensatzgruppen* bezeichnet. Die Datensätze einer Datensatzgruppe können entweder direkt aus einer Tabelle stammen oder das Ergebnis einer Abfrage sein. Ähnlich wie Formulare haben auch Datensatzgruppen einen Datensatzzeiger, der auf den aktuellen Datensatz zeigt. Mithilfe spezieller Methoden kann dieser Datensatzzeiger auf andere Datensätze verschoben werden, so dass die Bearbeitung jedes einzelnen Datensatzes möglich ist.

Access stellt drei verschiedene Typen von Datensatzgruppen zur Verfügung. Der Typ *Tabelle* repräsentiert direkt eine Tabelle. Bei diesem Typ enthält die Datensatzgruppe alle Datensätze, die auch in der Tabelle enthalten sind. Auch die Reihenfolge der Datensätze stimmt mit der in der Tabelle überein. Datensatzgruppen des Typs *Tabelle* zeichnen sich dadurch aus, dass eine Suche nach einem bestimmten Datensatz sehr zügig erfolgen kann. Dieser Vorteil muss allerdings damit erkauft werden, dass die Suche nur innerhalb von Datenfeldern ausgeführt werden kann, die mit einem Index versehen sind.

Änderungen, die Sie an den Datensätzen einer Datensatzgruppe vornehmen, spiegeln sich direkt in der zugrunde liegenden Tabelle wider. Das Gleiche gilt für den umgekehrten Fall, wenn von einer anderen Stelle aus Änderungen an der Tabelle vorgenommen wurden. Eine Datensatzgruppe des Typs *Tabelle* ist damit ein direktes Abbild einer Tabelle.

Der zweite Typ von Datensatzgruppen wird als *Dynaset* bezeichnet. Ein Dynaset ist nichts anderes als das Ergebnis einer Abfrage. Die Datensätze dieser Datensatzgruppe können dementsprechend auf mehreren verknüpften Tabellen basieren oder z.B. beliebig sortiert und gefiltert sein. Auch die Suche nach bestimmten Datensätzen ist unabhängig von den definierten Indizes in allen Datenfeldern möglich. Aus diesem Grund werden oft *Recordset*-Objekte des Typs *Dynaset* verwendet, bei denen eigentlich auch Datensatzgruppen des Typs *Tabelle* eingesetzt werden könnten. Die höhere Flexibilität bei der Suche wird dann allerdings durch eine geringere Suchgeschwindigkeit erkauft.

Wie bei Access üblich, können die Datensätze eines Abfrageergebnisses verändert werden, wobei Access diese Änderungen in den zugrunde liegenden Tabellen berücksichtigt. Dies trifft auch für *Recordset*-Objekte des Typs *Dynaset* zu.

Der dritte und letzte Typ von Datensatzgruppen hat den Namen *Snapshot*. Wie dieser Name bereits vermuten lässt, handelt es sich bei einem Snapshot um eine Momentaufnahme einer Gruppe von Datensätzen. Um diese zu erstellen, kopiert Access alle gewünschten Datensätze aus den Tabellen in die *Snapshot*-Datensatzgruppe. Da dieser Vorgang natürlich sehr arbeitsintensiv ist, sollten Snapshots nur dann eingesetzt werden, wenn sie wirklich benötigt werden. Dies ist beispielsweise bei zeitaufwändigen Auswertungen in Mehrbenutzerumgebungen der Fall, bei denen Sie Änderungen an den Datensätzen während der Auswertung ausschließen möchten.

Tabelle

Dynaset

Snapshot





Da Access eine Kopie der Datensätze anlegt, wirken sich Änderungen an einem Snapshot nicht auf die zugrunde liegende Tabelle aus. Auch auf dem umgekehrten Weg ziehen Änderungen an den zugrunde liegenden Tabellen keine Änderungen am Snapshot nach sich. Diese letzte Eigenschaft ist – wie bereits erwähnt – z. B. bei Auswertungen erwünscht.

Anlegen von Datensatzgruppen

Zur Handhabung von Datensatzgruppen stellt Access den Objektdatentyp *Recordset* zur Verfügung, der die drei Typen *Tabelle*, *Dynaset* und *Snapshot* unterstützt. Im Gegensatz zu älteren Access-Versionen muss also nicht mit drei verschiedenen Objektdatentypen gearbeitet werden.

OpenRecordset-
Methode

Das Anlegen einer Datensatzgruppe wird mithilfe der *OpenRecordset*-Methode bewerkstelligt, die von allen *Database*-Objekten zur Verfügung gestellt wird. Die Methode erwartet einen Parameter, der die Datenquelle – also die gewünschte Tabelle oder Abfrage – angibt:

```
<Database-Objekt>.OpenRecordset (<Datenquelle>)
```

Das Ergebnis dieser Methode ist ein *Recordset*-Objekt, das alle Datensätze der Datenquelle enthält. Die folgende Beispielprozedur legt eine Datensatzgruppe an, die auf der Tabelle *Kunden* basiert:

```
Sub DBTestprozedur3 ()

Dim AktuelleDB As Database
Dim MeineDG As Recordset

Set AktuelleDB = CurrentDb

Set MeineDG = AktuelleDB.OpenRecordset ("Kunden")

End Sub
```

Access erkennt in diesem Fall automatisch, dass es sich bei der Datenquelle *Kunden* um eine Tabelle handelt und legt daher eine Datensatzgruppe des Typs *Tabelle* an. Als Datenquelle können Sie allerdings auch den Namen einer Abfrage angeben, die Sie vorher über das Datenbankfenster von Access angelegt haben. In diesem Fall erstellt Access automatisch ein *Recordset*-Objekt des Typs *Dynaset*.

Eine weitere Alternative besteht darin, dass Sie direkt einen SQL-Text angeben:

```
Set MeineDG = AktuelleDB.OpenRecordset ("SELECT * _  
Kunden ORDER BY Name")
```

In diesem Fall legt Access wieder automatisch ein *Recordset*-Objekt des Typs *Dynaset* an. Zusätzlich werden die einzelnen Datensätze in der Datensatzgruppe gemäß der *ORDER BY*-Klausel nach dem Datenfeld *Name* sortiert.





Als Abfrage darf im Zusammenhang mit der *OpenRecordset*-Methode ausschließlich eine Auswahlabfrage angegeben werden (*SELECT*-Anweisung). Wenn Sie eine Aktionsabfrage ausführen möchten – die keine Datensätze zum Ergebnis hat, sondern lediglich Datensätze verändert – verwenden Sie die *Execute*-Methode des *Database*-Objekts. Der betreffende SQL-Text bzw. der Name der gewünschten Abfrage muss der Methode hierbei als Parameter übergeben werden. Die Anweisung



Hinweis

```
AktuelleDB.Execute ("DELETE * FROM Kunden")
```

löscht beispielsweise (ohne Sicherheitsabfrage!) alle Kunden aus der Kundentabelle.

Beim Erstellen einer Datensatzgruppe kann der Typ bei Bedarf auch explizit angegeben werden. Dies ist beispielsweise dann erforderlich, wenn Sie auf der Basis einer Tabelle eine Datensatzgruppe des Typs *Dynaset* erstellen möchten. Zu diesem Zweck besitzt die *OpenRecordset*-Methode einen zweiten Parameter mit dem Namen *Type*:

```
<Database-Objekt>.OpenRecordset (<Name>, <Type>)
```

Für *<Type>* kann einer der folgenden numerischen Werte angegeben werden:

Mögliche Angaben für den Parameter *<Type>*

Wert	Konstante	Typ der Datensatzgruppe
1	<i>dbOpenTable</i>	Tabelle
2	<i>dbOpenDynaset</i>	Dynaset
4	<i>dbOpenSnapshot</i>	Snapshot

Tabelle 5.3

Zu jedem der drei Werte stellt Access zusätzlich eine Konstante zur Verfügung, deren Einsatz die Lesbarkeit des Programmcodes erleichtert. Diese Konstanten können anstelle der Werte angegeben werden. Um beispielsweise auf der Grundlage der Tabelle *Kunden* eine *Dynaset*-Datensatzgruppe anzulegen, verwenden Sie die folgende Anweisung:

```
Set MeineDG = AktuelleDB.OpenRecordset ("Kunden",  
dbOpenDynaset)
```

Auch Snapshots lassen sich mithilfe des *<Type>*-Parameters erstellen. In diesem Fall geben Sie einfach die Konstante *dbOpenSnapshot* an:

```
Set MeineDG = AktuelleDB.OpenRecordset ("Kunden",  
dbOpenSnapshot)
```





Hinweis

Nicht möglich ist das Erstellen einer Datensatzgruppe des Typs *Tabelle*, wenn unter *<Name>* der Name einer Abfrage oder direkt ein SQL-Text angegeben ist. In diesem Fall tritt ein Laufzeitfehler auf!

In einigen Fällen kann es notwendig sein, den Typ einer Datensatzgruppe nachträglich ausfindig zu machen. Dies ist beispielsweise dann der Fall, wenn Sie einer Prozedur ein *Recordset*-Objekt als Parameter übergeben. Damit die Manipulationen an der Datensatzgruppe korrekt durchgeführt werden können (z.B. die Suche nach einem bestimmten Datensatz), muss der Typ bekannt sein. Dieser lässt sich mithilfe der *Type*-Eigenschaft ermitteln, der von allen *Recordset*-Objekten zur Verfügung gestellt wird. Der Inhalt dieser Eigenschaft ist einer der drei Werte 1 (Tabelle), 2 (Dynaset) oder 4 (Snapshot). Die folgende Prozedur erwartet beim Aufruf ein *Recordset*-Objekt und gibt den Typ der Datensatzgruppe in einem Meldungsfenster aus:

```
Sub RecordsetTypAnzeigen (R As Recordset)

Dim Meldung As String

Select Case R.Type

    Case 1
        Meldung = "Tabelle"
    Case 2
        Meldung = "Dynaset"
    Case 4
        Meldung = "Snapshot"

End Select

Meldung = "Der Typ der Datensatzgruppe ist " & _
        & Meldung & " "

MsgBox Meldung

End Sub
```

Der Aufruf könnte von der Prozedur *DBTestprozedur3* aus über die folgende Anweisung erfolgen:

```
RecordsetTypAnzeigen MeineDG
```

Schließen einer Datensatzgruppe

Genau wie Objekte des Typs *Database* werden auch *Recordset*-Objekte automatisch geschlossen, wenn die Lebensdauer der betreffenden Objektvariablen beendet ist. Das Schließen kann aber auch mithilfe der *Close*-Methode vorzeitig erfolgen. Die folgende Beispielprozedur zeigt dies:



```
Sub DBTestprozedur4()  
  
Dim MeineDG As Recordset  
Dim AktuelleDB As DATABASE  
  
    Set AktuelleDB = CurrentDb  
    Set MeineDG = AktuelleDB.OpenRecordset("Kunden")  
  
    MeineDG.Close  
  
End Sub
```

Alle Datensatzgruppen, die während einer Datenbanksitzung geöffnet sind, werden von Access innerhalb der *Recordset*-Auflistung verwaltet. Für den Anwender ist der Zugriff auf diese Auflistung in der Praxis allerdings nicht erforderlich, da Access für alle notwendigen Funktionen spezielle Methoden zur Verfügung stellt.

Hinweis

Verschieben des Datensatzzeigers

Wie bereits erwähnt, besitzt jede einzelne Datensatzgruppe einen individuellen Datensatzzeiger, der den aktuellen Datensatz kennzeichnet. Nur dieser aktuelle Datensatz erlaubt das Auslesen oder Verändern von Feldinhalten. Wenn Sie auf mehrere Datensätze zugreifen möchten, müssen Sie den Datensatzzeiger nacheinander auf die gewünschten Datensätze positionieren. Zu diesem Zweck stellt Access vier *Move*-Methoden zur Verfügung, die die Positionierung ermöglichen:

Die *Move*-Methoden

Methode	Beschreibung
<i>MoveFirst</i>	Positioniert auf den ersten Datensatz.
<i>MoveLast</i>	Positioniert auf den letzten Datensatz.
<i>MoveNext</i>	Positioniert auf den folgenden Datensatz.
<i>MovePrevious</i>	Positioniert auf den vorhergehenden Datensatz.

Tabelle 5.4

Von der Wirkung sind die vier *Move*-Methoden mit den Navigationsschaltflächen vergleichbar, die Sie vom Wechseln des aktuellen Datensatzes in Formularen kennen. Das folgende Beispiel legt eine Datensatzgruppe auf Basis der Tabelle *Kunden* an und positioniert anschließend auf den letzten Datensatz dieser Gruppe:

```
Sub DBTestprozedur5()  
  
Dim AktuelleDB As Database  
Dim MeineDG As Recordset
```





```
Set AktuelleDB = CurrentDb
Set MeineDG = AktuelleDB.OpenRecordset ("Kunden")

MeineDG.MoveLast
```

End Sub

Hinweis



Wenn Sie eine Datensatzgruppe mithilfe der *OpenRecordset*-Methode anlegen, positioniert Access den Datensatzzeiger automatisch auf den ersten Datensatz. Dies ist allerdings nur der Fall, wenn die Datensatzgruppe mindestens einen Datensatz enthält.

EOF-, BOF-Eigenschaften



Eine Besonderheit tritt auf, wenn der Datensatzzeiger momentan auf den letzten Datensatz zeigt und Sie die *MoveNext*-Methode ausführen. Obwohl hier kein Datensatz mehr vorhanden ist, führt Access die Positionierung ohne Fehlermeldung aus. Gleichzeitig wird der Datensatzzeiger hinter den letzten Datensatz positioniert und ist damit ungültig. Wenn nun ein Zugriff auf den (nicht vorhandenen) Datensatz erfolgt oder erneut die *MoveNext*-Methode ausgeführt wird, bricht Access das Programm mit einem Laufzeitfehler ab.

Um diesen Zustand zu ermitteln, stellen Datensatzgruppen die *EOF*-Eigenschaft zur Verfügung (*EOF* = **End Of File**). Diese Eigenschaft besitzt normalerweise den Wert *False*. Sollte der Datensatzzeiger allerdings hinter den letzten Datensatz positioniert werden, besitzt sie den Wert *True*.

Genauso verhält es sich mit der Eigenschaft *BOF* (**Bottom Of File**). Diese besitzt den Wert *True*, sobald sich der Datensatzzeiger vor dem ersten Datensatz befindet. Das folgende Beispiel verwendet die *MoveNext*-Methode im Zusammenhang mit der *EOF*-Eigenschaft, um die Namen aller Kunden der Datensatzgruppe *MeineDG* im Direktfenster auszugeben. Innerhalb der Prozedur erfolgt außerdem der Zugriff auf das Datenfeld *Name*. Die genaue Vorgehensweise beim Zugriff auf Datenfelder lernen Sie im folgenden Abschnitt noch kennen.

```
Sub DBTestprozedur6 ()

Dim AktuelleDB As Database
Dim MeineDG As Recordset

Set AktuelleDB = CurrentDb
Set MeineDG = AktuelleDB.OpenRecordset ("Kunden")

Do While MeineDG.EOF = False

    Rem Zugriff auf das Datenfeld "Name"
    Debug.Print MeineDG.Fields!Name

    MeineDG.MoveNext

Loop

End Sub
```



In diesem Beispiel wird eine *Do...Loop*-Schleife verwendet, bei der die Bedingung bereits beim Eintritt in die Schleife überprüft wird. Wenn nun die Tabelle *Kunden* und damit auch die Datensatzgruppe *MeineDG* keinen einzigen Datensatz enthält, setzt Access die *EOF*-Eigenschaft (und auch die *BOF*-Eigenschaft) von vornherein auf den Wert *True*. In diesem Fall wird die Schleife also nicht durchlaufen. Wäre dies nicht der Fall, würde der Zugriff auf das Datenfeld *Name* oder aber auch die Anwendung der *MoveNext*-Methode zu einem Fehler führen.

Eine weitere Möglichkeit zur Iteration über alle Datensätze bietet die *RecordCount*-Eigenschaft von Datensatzgruppen. Diese Eigenschaft enthält die Anzahl der Datensätze in der Datensatzgruppe. Die *RecordCount*-Eigenschaft ist aber mit Vorsicht zu genießen: Bei Datensatzgruppen des Typs *Tabelle* oder *Dynaset* kann sich die Anzahl der Datensätze ständig verändern, wenn – z.B. in Mehrbenutzerumgebungen – von anderer Stelle Datensätze hinzugefügt oder gelöscht werden.

Ein anderes Problem besteht darin, dass die *RecordCount*-Eigenschaft nur beim Typ *Tabelle* auf Anhieb den korrekten Wert liefert. Bei Datensatzgruppen des Typs *Dynaset* und *Snapshot* wird nur die Anzahl der Datensätze angegeben, auf die bis jetzt zugegriffen wurde. Im folgenden Beispiel gibt die *RecordCount*-Eigenschaft also unabhängig von der Anzahl der Datensätze stets den Wert 1 zurück:

```
Sub DBTestprozedur7()  
  
Dim AktuelleDB As Database  
Dim MeineDG As Recordset  
  
    Set AktuelleDB = CurrentDb  
    Set MeineDG = AktuelleDB.OpenRecordset("Kunden", _  
                                           dbOpenDynaset)  
  
    Debug.Print MeineDG.RecordCount  
  
End Sub
```

Um hier die Gesamtzahl aller Datensätze zu ermitteln, müssen Sie zuerst den letzten Datensatz der Datensatzgruppe mithilfe der *MoveLast*-Methode zum aktuellen Datensatz machen. Anschließend enthält die *RecordCount*-Eigenschaft die richtige Anzahl. Das folgende Beispiel nutzt dies, um wieder alle Kundennamen im Direktfenster auszugeben:

```
Sub DBTestprozedur8()  
  
Dim AktuelleDB As Database  
Dim MeineDG As Recordset  
Dim Anzahl As Long  
Dim i As Long  
  
    Set AktuelleDB = CurrentDb  
    Set MeineDG = AktuelleDB.OpenRecordset("Kunden", _  
                                           dbOpenDynaset)
```

RecordCount-
Eigenschaft





```

MeinedG.MoveLast
Anzahl = MeinedG.RecordCount
MeinedG.MoveFirst

For i = 1 To Anzahl

    Debug.Print MeinedG!Name

    MeineDG.MoveNext

Next i

End Sub

```

Beachten Sie, dass der Datentyp der *RecordCount*-Eigenschaft *Long* ist und die Anzahl dementsprechend in einer *Long*-Variablen gespeichert wird. *Integer*-Variablen sind hierzu ungeeignet, da sie nur einen Maximalwert von ca. 32.000 unterstützen. Eine Tabelle kann aber wesentlich mehr Datensätze besitzen.

Hinweis



Sollte die Datensatzgruppe keine Datensätze beinhalten, hat die *RecordCount*-Eigenschaft den Wert 0.

Die folgende Funktion kann eingesetzt werden, um immer die richtige Anzahl der Datensätze einer Datensatzgruppe zu ermitteln:

```

Function AnzahlDatensätze(R As Recordset) As Long

Dim Lesezeichen As String

If R.Type = dbOpenTable Then
    AnzahlDatensätze = R.RecordCount
Else
    If R.RecordCount = 0 Then
        AnzahlDatensätze = 0
    Else
        Lesezeichen = R.Bookmark
        R.MoveLast
        AnzahlDatensätze = R.RecordCount
        R.Bookmark = Lesezeichen
    End If
End If

End Function

```

Zuerst wird der Typ der Datensatzgruppe untersucht. Da die *RecordCount*-Eigenschaft bei *Recordset*-Objekten des Typs *Tabelle* immer den richtigen Wert zurückgibt, kann dieser direkt als Ergebnis der Funktion zurückgegeben werden. Handelt es sich dagegen um ein *Dynaset* oder einen *Snapshot*, wird zunächst überprüft, ob überhaupt ein Datensatz vorhanden ist. Wenn kein Datensatz vorhanden ist (die *RecordCount*-Eigenschaft hat in diesem Fall das Ergebnis 0), wird der Wert 0



zurückgegeben. Sind dagegen Datensätze vorhanden, wird auf den letzten Datensatz positioniert und das nun korrekte Ergebnis der *RecordCount*-Funktion als Ergebnis zurückgegeben.

Eine Besonderheit der Funktion besteht darin, dass abschließend mithilfe eines Lesezeichens wieder auf denjenigen Datensatz positioniert wird, der beim Start der Funktion aktuell war. Mehr über den Umgang mit Lesezeichen erfahren Sie im Abschnitt *Lesezeichen*.

Neben den vier vorgestellten Methoden zur Positionierung des Datensatzzeigers stellt Access noch eine weitere Methode mit dem Namen *Move* zur Verfügung. Diese Methode ermöglicht es, den Zeiger in einem Schritt um mehrere Datensätze zu verschieben. Dazu muss beim Aufruf der Methode ein numerischer Parameter angegeben werden, der die Anzahl der zu überspringenden Datensätze angibt. Positive Werte bewirken hierbei einen Sprung zum Tabellenende hin, negative Werte in die entgegengesetzte Richtung. Sollte ein Sprung über das Tabellenende bzw. den Tabellenanfang hinaus erfolgen, setzt Access die *EOF*- bzw. *BOF*-Eigenschaft auf den Wert *True*. Das folgende Beispiel überspringt einen Datensatz und positioniert damit direkt auf den übernächsten Datensatz:

```
MeineDG.Move 2
```

Zugriff auf Datenfelder

Jede Datensatzgruppe besitzt eine Auflistung mit dem Namen *Fields*, die den Zugriff auf die einzelnen Datenfelder erlaubt. Mit der *Fields*-Auflistung arbeiten Sie wie mit jeder anderen Auflistung. Die folgenden Beispiele beziehen sich auf das bereits bekannte *Recordset*-Objekt mit dem Namen *MeineDG*. Der Zugriff auf das Datenfeld *Name* ist mithilfe der folgenden beiden Varianten möglich:

```
MeineDG.Fields!Name  
MeineDG.Fields("Name")
```

Sollte der Feldname Leer- oder Sonderzeichen enthalten, muss dieser bei Verwendung der ersten Variante innerhalb eckiger Klammern [...] angegeben werden.

Da es sich bei der *Fields*-Auflistung um die Standardauflistung von *Recordset*-Objekten handelt, können auch die folgenden Abkürzungen verwendet werden:

```
MeineDG!Name  
MeineDG("Name")
```

Um nun den Inhalt eines bestimmten Datenfeldes zu ermitteln, muss die *Value*-Eigenschaft eines Feldes verwendet werden. Das folgende Beispiel hat den Inhalt des Feldes *Name* zum Ergebnis:

```
MeineDG.Fields!Name.Value
```



Hinweis



**Die *Fields*-
Auflistung**



Hinweis





Ähnlich wie bei Steuerelementen gibt es auch hier wieder eine Kurzschreibweise, bei der die *Value*-Eigenschaft komplett ausgelassen wird:

```
MeineDG.Fields!Name
```

Noch kürzer wird der Ausdruck, wenn Sie den Namen der Auflistung ebenfalls auslassen:

```
MeineDG!Name
```

Hinweis



Ein oft gemachter Fehler ist, dass bei dem Zugriff auf einen Feldinhalt mit dem Namen *Name* anstelle des Ausrufungszeichens fälschlicherweise ein Punkt angegeben wird:

```
MeineDG.Name
```

In diesem Fall gibt Access nicht den Inhalt des Datenfeldes *Name* aus, sondern greift auf die *Name*-Eigenschaft des *Recordset*-Objekts zu!

Hinweis



Bevor Sie den Inhalt eines Feldes dauerhaft verändern können, müssen Sie den Datensatz mittels der *Edit*-Methode für die Bearbeitung vorbereiten. Mehr dazu erfahren Sie im Abschnitt *Ändern, Hinzufügen und Löschen von Datensätzen*.

Zugriff über die Feldposition



Neben dem Zugriff über den Feldnamen kann mithilfe der *Fields*-Auflistung auch über die Position des Feldes innerhalb der Auflistung Bezug genommen werden. Die folgende Beispielprozedur listet die Feldnamen und die entsprechenden Inhalte des aktuellen Datensatzes einer Datensatzgruppe im Direktfenster auf:

```
Sub FeldinhalteAuflisten (R As Recordset)
```

```
Dim i As Integer
```

```
For i = 0 To R.Fields.Count - 1
```

```
    Debug.Print R(i).Name, R(i).Value
```

```
Next i
```

```
End Sub
```

Mithilfe der Schleife werden alle Datenfelder des aktuellen Datensatzes nacheinander ausgewählt. Die Anweisung innerhalb der Schleife gibt schließlich nacheinander den Feldnamen und den Inhalt des Feldes aus. In der ausführlichen Schreibweise könnte diese Anweisung auch folgendermaßen aussehen:

```
Debug.Print R.Fields(i).Name, R.Fields(i).Value
```



Alternativ kann auch eine *For Each*-Schleife eingesetzt werden:

```
Sub FeldinhalteAuflisten2(R As Recordset)

Dim EinFeld As Field

    For Each EinFeld In R.Fields

        Debug.Print EinFeld.Name, EinFeld.Value

    Next EinFeld

End Sub
```

Zum Start einer der beiden Prozeduren müssen Sie nun lediglich eine *Recordset*-Variable angeben. Beachten Sie hierbei, dass der Datensatzzeiger dieser Datensatzgruppe auf einen gültigen Datensatz zeigen muss, da der Zugriff auf die Datenfelder sonst zu einem Fehler führt. Dies können Sie überprüfen, indem Sie zu Beginn der Prozedur die Werte der *EOF*- und *BOF*-Eigenschaften auswerten:

```
If R.EOF = True Or R.BOF = True Then Exit Sub
```

Wenn Sie den Inhalt eines Feldes einer VBA-Variablen zuweisen möchten, müssen Sie darauf achten, dass der betreffende Felddatentyp mit dem VBA-Datentyp kompatibel ist. Um diesen Problemen aus dem Wege zu gehen, wird normalerweise der Datentyp *Variant* verwendet. Ein weiterer Vorteil ist, dass es bei Datenfeldern mit dem Inhalt *NULL* keine Probleme gibt.

Zuweisung an Variablen

Wenn Sie lieber einen der anderen Grunddatentypen vorziehen, etwa um die Arbeitsgeschwindigkeit zu optimieren, müssen Sie die passenden Datentypen gemäß folgender Tabelle auswählen:

Kompatible Datentypen

Felddatentyp	Feldgröße	VBA-Grunddatentyp
<i>AutoWert</i>	Long Integer	Long
	Replikations-ID	Variant
<i>Währung</i>	-	Currency
<i>Datum/Zeit</i>	-	Variant
<i>Memo</i>	-	String
<i>Zahl</i>	Byte	Byte
	Integer	Integer
	Long Integer	Long



Kompatible Datentypen (Forts.)

Felddatentyp	Feldgröße	VBA-Grunddatentyp
	Single	Single
	Double	Double
<i>OLE-Objekt</i>	-	String (eingeschränkt)
<i>Text</i>	-	String
<i>Ja/Nein</i>		Boolean
<i>Hyperlink</i>	-	String

Tabelle 5.5

Hinweis



Der Zugriff auf Felder des Typs *OLE-Objekt* ist von VBA aus nur eingeschränkt möglich. Bei Feldern des Typs *OLE-Objekt* handelt es sich eigentlich um Datenfelder zur Speicherung so genannter **BLOBs** (**B**inary **L**arge **O**bjects). BLOBs sind nichts anderes als eine lange Sequenz einzelner Bytes, die abhängig vom Kontext unterschiedlich interpretiert werden können. Da Zeichenketten des Typs *String* im Prinzip den gleichen Aufbau besitzen, lassen sich damit BLOBs innerhalb einer Variablen speichern. Die Bearbeitung der Zeichenkette und damit der Daten ist ebenfalls möglich. Sie müssen hierbei jedoch berücksichtigen, dass Access innerhalb von VBA den Unicode als Zeichensatz verwendet, bei dem ein Zeichen durch zwei Byte dargestellt wird. Zur Bearbeitung einer Zeichenkette auf Byte-Ebene stellt Access spezielle Funktionen zur Verfügung, die mit dem Buchstaben *B* (für Byte) enden: *AscB*, *ChrB*, *LeftB*, *MidB*, *RightB*.

Durchsuchen von Datensatzgruppen

Bei der Suche nach einem bestimmten Datensatz innerhalb einer Datensatzgruppe muss grundsätzlich zwischen *Recordset*-Objekten des Typs *Tabelle* und *Recordset*-Objekten des Typs *Dynaset* oder *Snapshot* unterschieden werden.

Bei *Dynaset*- und *Snapshot*-Datensatzgruppen stellen *Recordset*-Objekte die folgenden vier *Find*-Methoden zur Verfügung:

Die *Find*-Methoden

Methode	Funktion
<i>FindFirst</i> <Criteria>	Bewegt den Datensatzzeiger zum ersten Datensatz, der die Bedingung erfüllt.
<i>FindLast</i> <Criteria>	Bewegt den Datensatzzeiger zum letzten Datensatz, der die Bedingung erfüllt.



Die Find-Methoden (Forts.)

Methode	Funktion
<i>FindNext</i> <Criteria>	Setzt den Datensatzzeiger auf den nächsten Datensatz, der die Bedingung erfüllt.
<i>FindPrevious</i> <Criteria>	Setzt den Datensatzzeiger auf den vorherigen Datensatz, der die Bedingung erfüllt.

Tabelle 5.6

Alle vier Methoden erwarten als Parameter eine Bedingung, die die gesuchten Datensätze auswählt. Bei dieser Bedingung handelt es sich um eine Zeichenkette, die den *WHERE*-Teil eines SQL-Textes enthält. Die folgende Tabelle enthält einige gültige Bedingungen:

Beispiele für Bedingungen der Find-Methoden

Beispiel-Bedingung

Name = "Müller"

Name Like "M*ier"

Name = "Müller" *AND Postleitzahl* = "40595"

Gehalt < 30000

KundenNr = 4567

Tabelle 5.7

Bei der Angabe der Bedingungen innerhalb von VBA ergibt sich das Problem, dass innerhalb der Bedingungen Zeichenketten verwendet werden (z. B. *Müller*). Solche Zeichenketten, die in eine andere Zeichenkette eingebettet sind, müssen in diesem Fall nicht innerhalb normaler Anführungszeichen ("..."), sondern innerhalb einfacher Anführungszeichen ('...') angegeben werden. Eine Alternative besteht darin, dass Sie – wie in dem folgenden Beispiel – anstelle der einfachen Anführungszeichen je ein doppeltes Anführungszeichen angeben:

```
"Name = ""Müller"" "
```

Die beiden Methoden *FindFirst* und *FindLast* suchen immer den ersten bzw. letzten Datensatz der Datensatzgruppe, der den angegebenen Bedingungen entspricht und positionieren den Datensatzzeiger auf diesen Datensatz. Sie sind damit unabhängig von der aktuellen Position des Datensatzzeigers. Der Ausdruck

```
MeinedG.FindFirst "Name = "Müller" "
```



Hinweis





sucht also nach dem ersten Datensatz, dessen Feld *Name* den Inhalt *Müller* besitzt. Die Methoden *FindNext* und *FindPrevious* suchen dagegen relativ vom aktuellen Datensatz aus den nächsten bzw. vorhergehenden Datensatz. Sollten also mehrere Datensätze den angegebenen Bedingungen entsprechen, können Sie mithilfe dieser Methoden alle passenden Datensätze nacheinander zum aktuellen Datensatz machen und bearbeiten.

NoMatch

Informationen darüber, ob die letzte Suche nach einem Datensatz erfolgreich war, stellt die *NoMatch*-Eigenschaft des betreffenden *Recordset*-Objekts zur Verfügung. Konnte kein Datensatz gefunden werden, besitzt diese Eigenschaft den Inhalt *True*. Beachten Sie, dass in diesem Fall kein aktueller Datensatz mehr existiert! Bei erfolgloser Suche müssen Sie den Datensatzzeiger also erst wieder mithilfe der *Move*- oder der *Find*-Methode neu positionieren, bevor Sie auf einen Datensatz zugreifen!

Das folgende Beispiel durchsucht alle Datensätze der Tabelle *Kunden* nach dem Namen *Müller* und gibt jeweils den Vornamen und die Postleitzahl im Direktfenster aus. Beachten Sie hierbei, dass auf der Basis der Tabelle *Kunden* eine *Dynaset*-Datensatzgruppe angelegt wird!

```
Sub DBTestprozedur9()

Dim AktuelleDB As Database
Dim MeineDG As Recordset
Dim Bedingung As String

Set AktuelleDB = CurrentDb
Set MeineDG = AktuelleDB.OpenRecordset _
    ("Kunden", dbOpenDynaset)

Bedingung = "Name = "Müller""

MeineDG.FindFirst Bedingung

Do While MeineDG.NoMatch = False

    Debug.Print MeineDG!Vorname, _
        MeineDG!Postleitzahl

    MeineDG.FindNext Bedingung

Loop

End Sub
```

Mithilfe der *FindFirst*-Methode wird zunächst der erste Datensatz gesucht, der der Bedingung entspricht. In die nachfolgende *Do...Loop*-Schleife wird nur eingetreten, wenn diese erste Suche bereits erfolgreich war. Innerhalb der Schleife wird der *Vorname* und die *Postleitzahl* des gefundenen Datensatzes ausgegeben und mithilfe der *FindNext*-Methode der nächste passende Datensatz gesucht. Bei jedem Durch-



lauf der Schleife überprüft die Prozedur mithilfe der *NoMatch*-Eigenschaft, ob die letzte Suche erfolgreich war. Wenn dies nicht der Fall ist und damit kein weiterer passender Datensatz vorhanden ist, beendet Access die Schleife.

Bei der Suche nach Datensätzen in *Recordset*-Objekten des Typs *Tabelle* können die *Find*-Methoden nicht angewendet werden. Speziell für diese *Recordset*-Typen stellt Access die *Seek*-Methode zur Verfügung. Der Nachteil der *Seek*-Methode gegenüber den *Find*-Methoden besteht darin, dass nur in Datenfeldern gesucht wird, für die beim Tabellenentwurf ein Index definiert wurde.

Da eine Tabelle mehrere Indizes besitzen kann, muss vor dem Einsatz der *Seek*-Methode zunächst einmal der gewünschte Index ausgewählt werden. Dies lässt sich mithilfe der *Index*-Methode bewerkstelligen, die nur von *Recordset*-Objekten des Typs *Tabelle* unterstützt wird:

```
<Datensatzgruppe>.Index = <Indexname>
```

Welchen Namen ein bestimmter Index hat, lässt sich beim Tabellenentwurf frei festlegen. Eine Übersicht erhalten Sie, indem Sie die betreffende Tabelle in der Entwurfsansicht öffnen und den Menüpunkt *Ansicht > Indizes* auswählen. In dem gleichnamigen Dialogfenster werden nun alle Indizes mit ihren Namen aufgelistet.

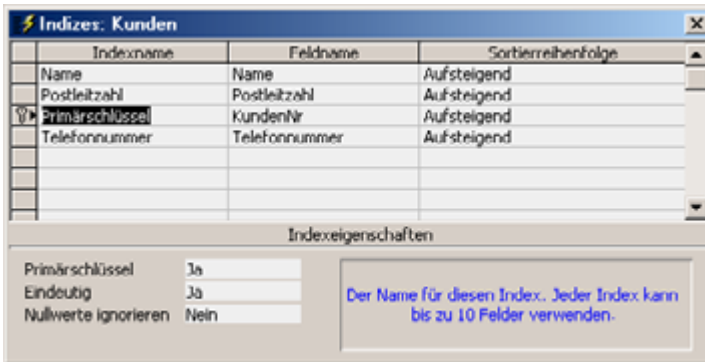


Abb. 5.2

Das Fenster *Indizes*

Wenn Sie beim Entwurf von Tabellen ein oder mehrere Felder als Primärschlüssel definieren, vergibt Access automatisch den Indexnamen *PrimaryKey*. Bei einzelnen indizierten Datenfeldern weist Access dem Index den Namen des Datenfeldes zu.

Sie sollten sich auf diese Regeln allerdings nicht verlassen, da der Indexname beim Tabellenentwurf frei verändert werden kann. Ein gutes Beispiel hierfür sind Tabellen, die Sie mit dem Tabellenassistenten von Access angelegt haben. Hier besitzt der Index des Primärschlüsselfeldes nicht den Namen *PrimaryKey*, sondern *Primärschlüssel*.



Hinweis



Welche Indizes eine Tabelle unterstützt, lässt sich innerhalb von VBA mithilfe der *Indexes*-Auflistung ausfindig machen. Diese Auflistung wird von Objekten des Typs *TableDef* zur Verfügung gestellt, die Sie noch im *Kapitel 10 Design von Datenbank-objekten* kennen lernen.

Wenn Sie einmal einen Indexnamen ausfindig gemacht haben, können Sie ihn mithilfe der *Index*-Eigenschaft aktivieren. Der folgende Ausdruck definiert beispielsweise für die Datensatzgruppe *MeineDG* den Index *Name*:

```
MeineDG.Index = "Name"
```

Hinweis



Beachten Sie, dass direkt nach dem Anlegen der Datensatzgruppe kein Index aktiv ist. Die Datensätze werden in diesem Fall gemäß ihrer Eingabereihenfolge angeordnet.

Seek-Methode

Nachdem der Index aktiviert wurde, kann dieses Feld mithilfe der *Seek*-Methode durchsucht werden. Zu diesem Zweck muss ähnlich wie bei den *Find*-Methoden eine Bedingung angegeben werden, die allerdings in die zwei Komponenten *Vergleichsoperator* und *Schlüssel* unterteilt ist:

```
<Datensatzgruppe>.Seek <Comparison>, <Key>
```

Als Vergleichsoperator muss unter *<Comparison>* eine der Zeichenketten *<*, *<=*, *=*, *>=* oder *>* angegeben werden. Die Operatoren können sowohl bei Zahlen als auch bei Zeichenketten eingesetzt werden. Bei den Operatoren *>*, *>=* und *=* wird die Suche von oben mit dem ersten, bei den Operatoren *<* und *<=* dagegen von unten mit dem letzten Datensatz begonnen.

Der Feldinhalt, nach dem gesucht wird (*Key*), muss vom gleichen Datentyp sein wie das zu durchsuchende Datenfeld, welches Sie vorher über die *Index*-Eigenschaft ausgewählt haben. Bei Zeichenketten überprüft Access hierbei nicht den vollen Feldinhalt, sondern lediglich, ob der Anfang der Zeichenkette mit dem Schlüssel übereinstimmt. Die folgenden Anweisungen würden beispielsweise den ersten Datensatz der Datensatzgruppe *MeineDG* auswählen, bei dem der Inhalt des Feldes *Name* mit der Zeichenkette *Mül* beginnt:

```
MeineDG.Index = "Name"  
MeineDG.Seek "=", "Mül"
```

Das nächste Beispiel sucht den Kunden mit der *KundenNr* 6:

```
MeineDG.Index = "Primärschlüssel"  
MeineDG.Seek "=", 6
```

War die letzte Suche nicht erfolgreich, wird genau wie bei den *Find*-Methoden die *NoMatch*-Eigenschaft der Datensatzgruppe auf den logischen Wert *True* gesetzt.

Mithilfe der *Seek*-Methode kann immer nur der erste bzw. letzte passende Datensatz ausfindig gemacht werden. Ein schrittweises Positionieren auf alle passenden Datensätze wie beim Einsatz der *Find*-Methoden ist nicht möglich.

Hinweis



Wenn Sie einen Mehrfachindex durchsuchen möchten, können Sie die einzelnen Schlüsselwerte hinter dem Vergleichsoperator angeben:

```
<Datensatzgruppe>.Seek <Comparison>, <Key1>, <Key2>,  
..., <Key13>
```

Sortieren und Filtern

Beim Positionieren des Datensatzzeigers mithilfe der *Move*-Methoden spielt die Reihenfolge der Datensätze innerhalb der Datensatzgruppe eine Rolle. Wenn Sie eine Datensatzgruppe des Typs *Tabelle* anlegen, werden die Datensätze hier gemäß dem aktiven Index angeordnet.

Sie können aber bei bereits erstellten *Recordset*-Objekten des Typs *Tabelle* den Index nachträglich wechseln. In diesem Fall ordnet Access die Datensätze gemäß dieses Indizes an und positioniert den Datensatzzeiger auf den ersten Datensatz. Die Aktivierung eines Indizes haben Sie bereits im vorhergehenden Abschnitt kennen gelernt. Das folgende Beispiel sortiert die Tabelle *Kunden* gemäß dem Index *Name*:

```
Sub DBTestprozedur10 ()  
  
Dim AktuelleDB As Database  
Dim MeineDG As Recordset  
  
Set AktuelleDB = CurrentDb  
Set MeineDG = AktuelleDB.OpenRecordset("Kunden", _  
    dbOpenTable)  
  
MeineDG.Index = "Name"  
  
End Sub
```

Die Reihenfolge der Datensätze in einer Datensatzgruppe des Typs *Dynaset* oder *Snapshot* kann nicht nachträglich verändert werden. Es ist allerdings möglich, bereits beim Erstellen der Datensatzgruppe eine bestimmte Reihenfolge vorzugeben.

Am einfachsten lässt sich dies bewerkstelligen, indem Sie beim Öffnen der Datensatzgruppe einfach einen SQL-Text mit einer Sortieranweisung angeben. Das folgende Beispiel legt eine neue Datensatzgruppe auf der Grundlage der Tabelle *Kunden* an, in der alle Kunden nach ihren Postleitzahlen sortiert sind:

```
Sub DBTestprozedur11 ()  
  
Dim AktuelleDB As Database  
Dim MeineDG As Recordset  
Dim SQL As String  
  
SQL = "SELECT * FROM Kunden ORDER BY Postleitzahl"
```

Datensatzgruppen
des Typs *Tabelle*

Datensatzgruppen
der Typen *Dynaset*
und *Snapshot*





```
Set AktuelleDB = CurrentDb
Set MeineDG = AktuelleDB.OpenRecordset (SQL,
dbOpenDynaset)
```

```
End Sub
```

Alternativ dazu können Sie natürlich den Namen einer Abfrage aus dem Datenbankfenster angeben, die ebenfalls die Datensätze in der gewünschten Reihenfolge zum Ergebnis hat.

Die zweite Möglichkeit zum Sortieren einer Datensatzgruppe geht davon aus, dass Sie bereits eine Datensatzgruppe besitzen und die Datensätze dieser Datensatzgruppe in eine neue Reihenfolge bringen möchten. Wie bereits erwähnt, ist es aber nicht möglich, nachträglich Änderungen an der Reihenfolge durchzuführen. Die Lösung besteht nun darin, dass Sie einfach eine zweite Datensatzgruppe anlegen, die auf der ersten Datensatzgruppe basiert. Bei der Anlage dieser zweiten Datensatzgruppe können Sie nun eine andere Reihenfolge bestimmen.

Damit dies technisch möglich ist, stellen alle Datensatzgruppen genau wie *Database*-Objekte die *OpenRecordset*-Methode zur Verfügung. Wird diese Methode auf eine Datensatzgruppe angewendet, darf allerdings kein Parameter angegeben werden. Das folgende Beispiel legt zuerst eine Datensatzgruppe *R1* auf Basis der Tabelle *Kunden* an. Anschließend wird aus dieser Datensatzgruppe eine zweite Datensatzgruppe *R2* erzeugt:

```
Sub DBTestprozedur12 ()

Dim AktuelleDB As Database
Dim R1 As Recordset
Dim R2 As Recordset

Set AktuelleDB = CurrentDb

Set R1 = AktuelleDB.OpenRecordset ("Kunden", _
dbOpenDynaset)

Set R2 = R1.OpenRecordset ()

End Sub
```

Eine Datensatzgruppe, die aus einer anderen Datensatzgruppe hervorgeht, besitzt immer automatisch den Typ *Dynaset*.



Hinweis

Die *Sort*-Eigenschaft

Die zweite Datensatzgruppe mit dem Namen *R2* enthält nun die gleichen Datensätze wie die Datensatzgruppe *R1*. Auch die Sortierreihenfolge stimmt mit der ursprünglichen Datensatzgruppe überein. Um nun die Datensätze der zweiten Datensatzgruppe zu sortieren, muss der ersten Datensatzgruppe *R1* noch vor dem Erstellen der zweiten Datensatzgruppe *R2* mithilfe der *Sort*-Eigenschaft ein Sortierkriterium zugewiesen werden. Dieses Sortierkriterium wirkt sich dann erst beim Erstellen der zweiten Datensatzgruppe aus.



Als Sortierkriterium muss der *Sort*-Eigenschaft der *ORDER BY*-Teil einer SQL-Anweisung zugewiesen werden. Um beispielsweise eine aufsteigende Sortierung nach dem Feld *Name* vorzunehmen, weisen Sie der *Sort*-Eigenschaft einfach die Zeichenkette *Name* zu. Eine absteigende Sortierung erreichen Sie über die Zeichenkette *Name DESC*. Das folgende Beispiel nimmt eine entsprechende Sortierung der Datensatzgruppe R2 vor:

```
Sub DBTestprozedur13 ()

Dim AktuelleDB As Database
Dim R1 As Recordset
Dim R2 As Recordset

    Set AktuelleDB = CurrentDb

    Set R1 = AktuelleDB.OpenRecordset("Kunden", _
        dbOpenDynaset)

    R1.Sort = "Name"

    Set R2 = R1.OpenRecordset()

End Sub
```

Etwas schneller erledigen Sie diese Aufgabe, indem Sie nur eine Variable des Typs *Recordset* verwenden. Durch die Zuweisung am Ende der folgenden Prozedur wird die alte unsortierte Datensatzgruppe von der neuen sortierten Datensatzgruppe überschrieben:

```
Sub DBTestprozedur14 ()

Dim AktuelleDB As Database
Dim R As Recordset

    Set AktuelleDB = CurrentDb

    Set R = AktuelleDB.OpenRecordset("Kunden", _
        dbOpenDynaset)
    R.Sort = "Name"
    Set R = R.OpenRecordset()

End Sub
```

Neben der Sortierung über die *Sort*-Eigenschaft ist auch das Filtern einer Datensatzgruppe mithilfe der *Filter*-Eigenschaft möglich. Die Vorgehensweise ist hierbei identisch mit der Technik zum Sortieren einer Datensatzgruppe. Der *Filter*-Eigenschaft einer Datensatzgruppe muss dazu lediglich der *WHERE*-Teil einer SQL-Abfrage zugewiesen werden. Mithilfe der *OpenRecordset*-Methode legen Sie schließlich eine neue Datensatzgruppe an, in der dann nur die gewünschten Datensätze enthalten

Filtern von Datensätzen



sind. Daraus ergibt sich auch, dass ein Filter nicht mehr zurückgenommen werden kann. Gegebenenfalls müssen Sie wieder auf die ursprünglichen Daten zurückgreifen.

Es folgen einige Beispielbedingungen, die Sie der *Filter*-Eigenschaft zuweisen können:

Beispielbedingungen zum Einsatz mit der Filter-Eigenschaft

Beispielbedingungen

"[Name] = 'Meier'"

"[Gehalt] > 10000"

"[Name] = 'Meier' And [Gehalt] > 10000"

Tabelle 5.8

Auch die gleichzeitige Verwendung der Eigenschaften *Filter* und *Sort* ist möglich, wie das folgende Beispiel zeigt:

```
Sub DBTestprozedur15 ()  
  
Dim AktuelleDB As Database  
Dim R As Recordset  
  
Set AktuelleDB = CurrentDb  
  
Set R = AktuelleDB.OpenRecordset ("Kunden", _  
    dbOpenDynaset)  
  
R.Sort = "Name"  
R.Filter = "Postleitzahl Like '4*'"  
  
Set R = R.OpenRecordset()  
  
End Sub
```

Die Datensatzgruppe *R* enthält alle Kunden nach Namen sortiert, deren Postleitzahlen mit der Zahl 4 beginnen.

Lesezeichen

Bei der Navigation durch eine Datensatzgruppe mithilfe der *Move*- oder der *Find*-Methoden ist es manchmal sinnvoll, einen bestimmten Datensatz zu markieren, um diesen anschließend ohne großen Such- oder Positionierungsaufwand wieder zum aktuellen Datensatz zu machen. Zu diesem Zweck unterstützt Access *Lesezeichen*, mit denen Sie jeweils den aktuellen Datensatz kennzeichnen können.



Lesezeichen werden in Variablen des Typs *String* gespeichert. Wenn Sie für den aktuellen Datensatz ein Lesezeichen anlegen, erstellt Access automatisch eine Zeichenkette, die diesen Datensatz eindeutig identifiziert. Der Inhalt dieser Zeichenkette ist meist eine zufällig wirkende Anhäufung von Zeichen und für den Programmierer nicht von Interesse.

Um diese Zeichenkette anzufordern, stellen *Recordset*-Objekte aller drei Typen die *Bookmark*-Eigenschaft zur Verfügung. Der Inhalt dieser Eigenschaft ist die besagte Zeichenkette, die sich auf den aktuellen Datensatz der Datensatzgruppe bezieht. Das folgende Beispiel legt ein Lesezeichen für den ersten Kunden an, der den Namen *Müller* hat. Dazu wird zuerst mithilfe der *FindFirst*-Methode der passende Datensatz gesucht. Sollte Access fündig werden (die *NoMatch*-Eigenschaft besitzt also den Wert *False*), wird mithilfe der *Bookmark*-Eigenschaft die Zeichenkette abgefragt und in der Variablen *Lesezeichen* gespeichert.

```
Sub DBTestprozedur16 ()

Dim AktuelleDB As Database
Dim R As Recordset
Dim Lesezeichen As String

Set AktuelleDB = CurrentDb

Set R = AktuelleDB.OpenRecordset("Kunden", _
    dbOpenDynaset)

R.FindFirst "Name = '" & Müller & "'"

If R.NoMatch = False Then

    Lesezeichen = R.Bookmark

End If

End Sub
```

Sollte kein aktueller Datensatz existieren, weil beispielsweise in der Datensatzgruppe überhaupt keine Datensätze vorhanden sind, führt das Abfragen des Lesezeichens zu einem Laufzeitfehler!

Um wieder auf einen per Lesezeichen markierten Datensatz zu positionieren, weisen Sie der *Bookmark*-Eigenschaft die vorher gespeicherte Zeichenkette erneut zu:

```
R.Bookmark = Lesezeichen
```

Sobald Sie die Datensatzgruppe schließen (z.B. per *Close*-Methode oder durch Zuweisung einer anderen Datenquelle), verlieren alle Lesezeichen ihre Gültigkeit. Dies gilt selbst dann, wenn Sie anschließend wieder eine identische Datensatzgruppe anlegen.

Die *Bookmark*-
Eigenschaft



Hinweis



Positionieren auf
einen Datensatz



Hinweis





Lesezeichen
austauschen

Die Einschränkungen beim Einsatz von Lesezeichen gehen allerdings noch weiter: Es ist nicht möglich, ein Lesezeichen zwischen zwei verschiedenen Datensatzgruppen auszutauschen, auch wenn diese vom Inhalt her identisch sind. Um trotzdem Lesezeichen zwischen zwei oder mehreren Datensatzgruppen austauschen zu können, erlaubt Access das Klonen von Datensatzgruppen mithilfe der *Clone*-Methode. Beim Klonen erstellt Access aus der Original-Datensatzgruppe eine zweite Datensatzgruppe, die jedoch sehr nah an die erste Datensatzgruppe geknüpft ist.

Datensatzgruppen klonen

Von der Anwendung her ist die *Clone*-Methode identisch mit der *OpenRecordset*-Methode. Das folgende Beispiel legt eine Datensatzgruppe *R1* auf Basis der Tabelle *Kunden* an. Anschließend wird mithilfe der *Clone*-Eigenschaft aus dieser ersten Datensatzgruppe eine zweite erstellt:

```
Sub DBTestprozedur17 ()

Dim AktuelleDB As Database
Dim R1 As Recordset
Dim R2 As Recordset

Set AktuelleDB = CurrentDb

Set R1 = AktuelleDB.OpenRecordset("Kunden", _
    dbOpenDynaset)
Set R2 = R1.Clone()

End Sub
```

Zwischen den beiden Datensatzgruppen können nun uneingeschränkt Lesezeichen ausgetauscht werden. Um beispielsweise in der Datensatzgruppe *R2* den gleichen Datensatz wie in der Datensatzgruppe *R1* auszuwählen, kann der folgende Ausdruck verwendet werden:

```
R2.Bookmark = R1.Bookmark
```

Hinweis



Access erlaubt das Einbinden von Tabellen fremder Datenbanken. Bei diesen Datenbanken kann es vorkommen, dass Lesezeichen generell nicht unterstützt werden. Im Einzelfall können Sie dies überprüfen, indem Sie vor der Verwendung der *Bookmark*-Eigenschaft die *Bookmarkable*-Eigenschaft auswerten. Werden Lesezeichen unterstützt, enthält diese Eigenschaft den Wert *True*; andernfalls ist der Inhalt *False*. Wenn Sie ausschließlich mit Access-Tabellen arbeiten, ist die Auswertung der *Bookmarkable*-Eigenschaft nicht notwendig, da Access-Tabellen Lesezeichen uneingeschränkt unterstützen.



Zugriff auf die Datensatzgruppen von Formularen

Auch Formularen liegt eine Datensatzgruppe zugrunde, die die innerhalb des Formulars angezeigten Datensätze enthält. Eine wichtige Eigenschaft ist in diesem Zusammenhang die *RecordsetClone*-Eigenschaft die von *Form*-Objekten – also von Formularen – zur Verfügung gestellt wird. Diese Eigenschaft enthält eine Kopie der Datensatzgruppe, die dem Formular zugrunde liegt. Das folgende Beispiel legt eine entsprechend geklonte Datensatzgruppe mit dem Namen *R* an, die auf dem (geöffneten) Formular *Kunden* basiert. Anschließend werden die Namen aller Datenfelder dieser Datensatzgruppe im Direktfenster ausgegeben:

```
Sub DBTestprozedur18()

Dim R As Recordset
Dim EinFeld As Field

DoCmd.OpenForm "Kunden"

Set R = Forms!Kunden.RecordsetClone

For Each EinFeld In R.Fields

    Debug.Print EinFeld.Name

Next EinFeld

End Sub
```

Neben der *RecordsetClone*-Eigenschaft unterstützen Formulare auch die *Bookmark*-Eigenschaft, die sich immer auf den momentan aktiven Datensatz bezieht. Mithilfe dieser Eigenschaft ist es möglich, ein Formular mit einer Datensatzgruppe zu synchronisieren, indem einfach Lesezeichen zwischen dem Formular und der (mithilfe der *RecordsetClone*-Eigenschaft geklonten) Datensatzgruppe ausgetauscht werden. Die folgende Prozedur macht sich dies zunutze, um innerhalb eines beliebigen geöffneten Formulars auf einen bestimmten Datensatz zu positionieren:

```
Function FormularDatensatzAuswählen(FormularObjekt _
    As Form, Bedingung As String) As Boolean

On Error GoTo Err

Dim R As Recordset

"Das Standardergebnis der Funktion ist False
"Sollte der Datensatz während der Abarbeitung der
"Funktion gefunden werden, wird das Ergebnis in
"True geändert
FormularDatensatzAuswählen = False

"Datenatzgruppe des Formulars anfordern
```

RecordsetClone-
Eigenschaft

Bookmark-
Eigenschaft von
Formularen





```
Set R = FormularObjekt.RecordsetClone

"Überprüfung, ob die Datensatzgruppe Lesezeichen
"unterstützt
If R.Bookmarkable = False Then
    FormularDatensatzAuswählen = False
    Exit Function
End If

"Bildschirmaktualisierung unterdrücken
FormularObjekt.Painting = False

"Datensatz suchen
R.FindFirst Bedingung

"Wurde der Datensatz gefunden?
If R.NoMatch = True Then
    "Daten des Formulars aktualisieren
    FormularObjekt.Requery
    "Neue aktualisierte Datensatzgruppe anfordern
    Set R = FormularObjekt.RecordsetClone
    R.FindFirst Bedingung "Erneut suchen
    "Überprüfung, ob nun ein Datensatz gefunden wurde
    If R.NoMatch = False Then "Datensatz gefunden?
        "Auf gefundenen Datensatz positionieren
        FormularObjekt.Bookmark = R.Bookmark
        "True als Erfolgsmeldung der Funktion zurückgeben
        FormularDatensatzAuswählen = True
    End If
Else
    FormularObjekt.Bookmark = R.Bookmark
    FormularDatensatzAuswählen = True
End If

"Bildschirmaktualisierung zulassen
FormularObjekt.Painting = True

Exit Function

"Fehlerbehandlungsroutine

Err:

FormularDatensatzAuswählen = False
FormularObjekt.Painting = True
Exit Function

End Function
```



Nachdem die Prozedur auf der Grundlage des gewünschten Formulars eine Datensatzgruppe geklont hat, wird mithilfe der *FindFirst*-Methode nach dem ersten passenden Datensatz gesucht. Sollte die Suche nicht zum gewünschten Erfolg führen, geht die Prozedur davon aus, dass die Daten innerhalb des Formulars nicht mehr aktuell sind. Nach einer Aktualisierung des Formulars mithilfe der *Requery*-Methode wird eine neue Datensatzgruppe geklont und die Suche wiederholt. Sollte die Suche erfolgreich sein, gibt die Funktion *True* zurück. Andernfalls ist das Ergebnis *False*.

Um beispielsweise in dem Formular *Kunden* auf den Kunden mit der *KundenNr* 6 zu positionieren, rufen Sie die Prozedur wie folgt auf:

```
Sub DBTestprozedur19()

Dim SucheErfolgreich As Boolean

DoCmd.OpenForm "Kunden"

SucheErfolgreich = FormularDatensatzAuswählen _
    (Forms!Kunden, "KundenNr = 6")

End Sub
```

Innerhalb der Prozedur *FormularDatensatzAuswählen* wird eine Formulareigenschaft mit dem Namen *Painting* verwendet. Wird dieser Eigenschaft der Wert *False* zugewiesen, unterdrückt Access alle Bildschirmausgaben für dieses Formular. In der Prozedur wird sie verwendet, damit durch das Aktualisieren des Formulars und die damit verbundene Aktualisierung des Fensterinhalts keine unnötigen Zeitverzögerungen entstehen.



Hinweis

Ändern, Hinzufügen und Löschen von Datensätzen

Bevor Sie Änderungen an dem aktuellen Datensatz einer Datensatzgruppe vornehmen, müssen Sie diese Absicht zunächst einmal Access mitteilen. Zu diesem Zweck stellen *Recordset*-Objekte des Typs *Tabelle* und *Dynaset* die *Edit*-Methode zur Verfügung. Diese Methode bewirkt, dass Access den aktuellen Datensatz in einen Zwischenspeicher kopiert, in dem dann Änderungen an den Feldinhalten vorgenommen werden können. Sie als Entwickler merken von diesem Vorgang nichts.

Wenn Sie die Änderungen abgeschlossen haben, muss der Inhalt des aktuellen Datensatzes durch den Inhalt des Zwischenspeichers ersetzt werden. Diesen Vorgang erledigt Access für Sie, wenn Sie die *Update*-Methode ausführen.

Die beiden Methoden *Edit* und *Update* stehen nur bei Datensatzgruppen des Typs *Tabelle* und *Dynaset* zur Verfügung, da an Snapshots keine Änderungen vorgenommen werden können!



Die *Edit*-Methode



Hinweis





Das folgende Beispiel sucht den ersten Datensatz der Datensatzgruppe *R*, bei dem das Feld *Name* den Inhalt *Meier* hat. Bei diesem Datensatz wird anschließend die Postleitzahl in *40595* geändert:

```
Sub DBTestprozedur20 ()

Dim AktuelleDB As Database
Dim R As Recordset

Set AktuelleDB = CurrentDb

Set R = AktuelleDB.OpenRecordset("Kunden", _
    dbOpenDynaset)

R.FindFirst "Name = "Meier""

If R.NoMatch = False Then

    R.Edit
    R!Postleitzahl = "40595"
    R.Update

End If

End Sub
```

Hinweis



Sollten Sie den aktuellen Datensatz der Datensatzgruppe wechseln, bevor Sie beim Bearbeiten dieses Datensatzes die *Update*-Methode ausführen, gehen alle Änderungen verloren!

Mehrbenutzerumgebungen



In einer Mehrbenutzerumgebung – beispielsweise im Netzwerk – kann es notwendig sein, den kompletten Datensatz während der Bearbeitung bis zur Ausführung der *Update*-Methode für den Zugriff durch andere Benutzer zu sperren. Um dies zu erreichen, setzen Sie einfach die *LockEdits*-Eigenschaft auf den Wert *True*. Besitzt diese Eigenschaft dagegen die Einstellung *False*, sperrt Access den Datensatz nur für eine kurze Zeit während der Abarbeitung der *Update*-Methode. Diese Einstellung eignet sich daher gut für zeitaufwändigere Manipulationen, bei denen der Datensatz nicht während der ganzen Zeit gesperrt sein soll.

Standardmäßig besitzt die *LockEdits*-Eigenschaft den Wert *True*. Der bearbeitete Datensatz wird also während der kompletten Bearbeitung gesperrt.

Sollten Sie selbst einmal "Opfer" eines gesperrten Datensatzes werden, tritt während der Ausführung der *Edit*-Methode ein Laufzeitfehler auf. Diesen Fehler können Sie mithilfe der *On Error*-Anweisung abfangen und entsprechend darauf reagieren.



Hinzufügen eines Datensatzes

Das Hinzufügen eines Datensatzes läuft auf eine ähnliche Weise wie das Ändern eines Datensatzes ab. Anstelle der *Edit*-Methode muss lediglich die *AddNew*-Methode verwendet werden. Die *AddNew*-Methode erzeugt einen leeren Datensatz im Zwischenspeicher, den Sie nun editieren können. Die *Update*-Methode bewirkt dann schließlich, dass dieser Datensatz der Datensatzgruppe endgültig hinzugefügt wird. Bei Datensatzgruppen des Typs *Tabelle* ordnet Access den neuen Datensatz gemäß des aktuellen Index ein.

Bei Datensatzgruppen des Typs *Dynaset* wird er lediglich an das Ende der Datensatzgruppe angehängt. Die Position des aktuellen Datensatzes bleibt also unverändert. Wenn Sie allerdings eine Positionierung auf den neuen Datensatz wünschen, wenden Sie folgenden Ausdruck an:

```
<Datensatzgruppe>.Bookmark =  
<Datensatzgruppe>.LastModified
```

Die *LastModified*-Eigenschaft enthält ein Lesezeichen, das den zuletzt geänderten oder hinzugefügten Datensatz der Datensatzgruppe markiert. Wenn Sie dieses Lesezeichen der *Bookmark*-Eigenschaft zuweisen, nimmt Access eine Positionierung auf diesen Datensatz vor.

Genau wie beim Ändern eines Datensatzes geht der neue Datensatz verloren, wenn Sie vor der Ausführung der *Update*-Methode den Datensatz wechseln.

Wenn Sie über die *AddNew*-Methode einen neuen Datensatz anlegen, füllt Access die betreffenden Felder automatisch mit den Standardwerten, die Sie beim Entwurf der Tabelle definiert haben.

Löschen eines Datensatzes

Zum Löschen des aktuellen Datensatzes stellt Access die *Delete*-Methode zur Verfügung, die den momentan aktuellen Datensatz löscht. Die Anwendung dieser Methode wirkt sich bei Datensatzgruppen des Typs *Tabelle* und *Dynaset* unterschiedlich aus:

Nachdem Sie den aktuellen Datensatz aus einem *Recordset*-Objekt des Typs *Tabelle* gelöscht haben, entfernt Access den Datensatz aus der Datensatzgruppe. Eine Folge davon ist, dass der Datensatzzeiger der Datensatzgruppe ungültig wird. Bevor Sie also weitere Zugriffe auf Datensätze vornehmen, müssen Sie einen anderen Datensatz zum aktuellen Datensatz machen.

Bei *Dynaset*-Datensatzgruppen wird der Datensatz nicht direkt gelöscht, sondern lediglich als gelöscht markiert. Der Datensatzzeiger bleibt weiterhin auf diesem Datensatz, obwohl er nun als "ungültig" gilt und ein weiterer Zugriff einen Laufzeitfehler auslösen würde. Wenn Sie den Datensatzzeiger nach dem Löschvorgang von dem Datensatz wegbewegen, löscht Access den Datensatz komplett aus der Datensatzgruppe.

Die *AddNew*-
Methode

LastModified



Hinweis

Die *Delete*-Methode

Löschen in *Dynaset*-
Datensatzgruppen





Das folgende Beispiel löscht die drei ersten Datensätze der Tabelle *Kunden*:

```
Sub DBTestprozedur21 ()

Dim AktuelleDB As Database
Dim R As Recordset
Dim i As Integer

Set AktuelleDB = CurrentDb

Set R = AktuelleDB.OpenRecordset("Kunden", _
    dbOpenTable)

If R.EOF = True Then Exit Sub "Keine Datensätze
                             "vorhanden

For i = 1 To 3

    R.Delete

    R.MoveNext

    "Wenn keine Datensätze mehr vorhanden sind,
    "wird die Schleife verlassen

If R.EOF Then Exit For

Next i

End Sub
```

Hinweis



Das Löschen von Datensätzen aus Datensatzgruppen des Typs *Snapshot* ist nicht möglich!

Auch beim Löschen eines Datensatzes kann es Probleme geben, wenn dieser gesperrt ist. In diesem Fall tritt während der Ausführung der *Delete*-Methode ein Laufzeitfehler auf.

Das folgende Beispiel zeigt noch einmal die Manipulation von Datensätzen anhand eines Beispiels zum Suchen und Ersetzen von Feldinhalten innerhalb einer Datensatzgruppe. Der Prozedur *FeldinhalteErsetzen* muss beim Aufruf eine Datensatzgruppe des Typs *Tabelle* oder *Dynaset* übergeben werden. Außerdem muss das zu durchsuchende Feld sowie der alte und der neue Text angegeben werden. Da der Vorgang etwas länger dauern kann, wird die Sanduhr mithilfe der *Hourglass*-Aktion aktiviert, die den Mauszeiger ersetzt. Am Ende der Prozedur wird diese wieder abgeschaltet:

```
Sub FeldinhalteErsetzen (R As Recordset, Feld As _
    String, Alt As String, Neu As String)
```





```
If R.RecordCount = 0 Then Exit Sub
DoCmd.Hourglass True
R.MoveFirst

Do While Not R.EOF

    R.Edit
    R(Feld) = TextErsetzen(R(Feld), ALt, Neu)
    R.Update

    R.MoveNext

Loop
DoCmd.Hourglass False
```

End Sub

Die folgende Testprozedur zeigt den Einsatz der *FeldinhalteErsetzen*-Prozedur anhand eines Beispiels. Hierbei wird das Feld *Name* nach dem Text *GMBH* durchsucht, der durch *GmbH* ersetzt wird:

```
Sub DBTestprozedur22 ()

Dim AktuelleDB As Database
Dim R As Recordset

Set AktuelleDB = CurrentDb

Set R = AktuelleDB.OpenRecordset("Kunden", _
    dbOpenTable)

FeldinhalteErsetzen R, "Name", "GMBH", "GmbH"
```

End Sub

Das Durchsuchen und Ersetzen des Feldinhalts erledigt die Funktion *TextErsetzen*. Die Wirkungsweise dieser Funktion können Sie im Direktfenster ausprobieren. Wenn Sie hier den Ausdruck

```
Print TextErsetzen ("Willi Meier", "Willi", "Dieter")
```

angeben, ersetzt Access den Vornamen *Willi* durch den Text *Dieter*. Den neuen Text "Dieter Meier" gibt die Funktion als Ergebnis zurück.

```
Function TextErsetzen(ByVal Text As Variant, Suchen _
    As String, Ersetzen As String) As Variant
```

```
Dim Position As Integer
Dim Länge As Integer
```



Die Funktion
TextErsetzen




```

If IsNull(Text) Then Exit Function

Länge = Len(Suchen)

Position = 0

Do
    Position = InStr(Position + 1, Text, Suchen)
    If Position = 0 Then Exit Do
    Text = Left(Text, Position - 1) + Ersetzen _
        + Mid(Text, Position + Länge)
Loop

TextErsetzen = Text

End Function

```

Fehler beim Datenzugriff

Error-Objekt

Um auf Fehler, die im Zusammenhang mit dem Zugriff auf Daten auftreten, besser reagieren zu können, stellt Access eine spezielle Auflistung mit dem Namen *Errors* zur Verfügung. Innerhalb dieser Auflistung wird der zuletzt aufgetretene Datenzugriffsfehler durch ein oder mehrere *Error*-Objekte repräsentiert.

Wenn im Zusammenhang mit einem Datenzugriff ein Laufzeitfehler auftritt, legt Access in der *Errors*-Auflistung in der Regel nur ein einziges *Error*-Objekt ab, das den aufgetretenen Fehler genauer beschreibt. Besonders beim Datenzugriff per ODBC-Treiber können sich hier aber auch mehrere *Error*-Objekte ansammeln. Diese Objekte repräsentieren Fehler, die in den einzelnen Zugriffsebenen auftreten. Die einzelnen Objekte sind hierbei nach den Zugriffsebenen sortiert. Das erste Objekt ist der tiefsten Ebene zugeordnet.

Eigenschaften des
Error-Objekts

Informationen über ein *Error*-Objekt erhalten Sie mithilfe der Eigenschaften *Number*, *Description* und *Source*:

Eigenschaften des *Error*-Objekts

Eigenschaft	Beschreibung
<i>Number</i>	Die Nummer des aufgetretenen Fehlers.
<i>Description</i>	Eine kurze Beschreibung des Fehlers.
<i>Source</i>	Das Objekt, in dem der Fehler aufgetreten ist.

Tabelle 5.9



Das folgende Beispiel löst einen Laufzeitfehler aus, indem direkt nach dem Öffnen einer Datensatzgruppe zweimal auf den vorhergehenden Datensatz positioniert wird. Der Laufzeitfehler wird in einer Fehlerbehandlungsroutine abgefangen. Hierbei werden die Inhalte der drei Eigenschaften jedes einzelnen *Error*-Objekts im Direktfenster ausgegeben.

```
Sub FehlerTest()

On Error GoTo Err

Dim R As Recordset
Dim EinFehler As Error

    Set R = CurrentDb.OpenRecordset("Kunden")

    R.MovePrevious
    R.MovePrevious

Exit Sub

Err:

For Each EinFehler In DBEngine.Errors

    Debug.Print EinFehler.Number, _
                EinFehler.Description, _
                EinFehler.Source

Next EinFehler

End Sub
```

Folgende Information erscheint nach Ausführung der Prozedur im Direktfenster:

```
3021          Kein aktueller Datensatz.    DAO.Recordset
```

Datenzugriff mit ADO

Zur weiteren Vereinheitlichung des Zugriffs auf Datenquellen unterschiedlicher Art und Herkunft hat Microsoft unter dem Namen *OLE-DB* eine Reihe von COM-Schnittstellen entwickelt. Eine solche OLE-DB-Datenquelle (ein so genannter OLE-DB-Provider) kann im einfachsten Fall eine simple Textdatei, eine relationale Datenbank oder auch ein ganzes Cluster von Datenbanken sein. Darüber hinaus ist es auch denkbar, heutzutage noch gar nicht bekannte Datenspeichermodelle über einen OLE-DB-Provider abzubilden und über die OLE-DB-Schnittstellen zur Verfügung zu stellen. Diese OLE-DB-Schnittstellen sind jedoch stellenweise recht komplex und lassen sich nicht immer direkt in Visual-Basic-Code verwenden.





Als Bindeglied zwischen der „nackten“ COM-Welt und der VB(A)-Welt dienen die **Active(X) Data Objects (ADO)**, die ein leicht vereinfachtes Programmiermodell für den Zugriff auf OLE-DB-Provider bieten.

Das ADO-Programmiermodell umfasst (im Gegensatz zum in früheren Access-Versionen verwendeten DAO-Objektmodell) nur einige wenige Objekte und Auflistungen in einer recht flachen Hierarchie. Über ADO sind asynchrone Datenzugriffe möglich, was auch durch die Bearbeitung der von verschiedenen Objekten ausgelösten Ereignisse (*Events*) unterstützt wird. Auf die Ereignisse wird hier jedoch nicht näher eingegangen, da sie in Access nur in Ausnahmefällen wirklich benötigt werden.

Die Beispielprozeduren zu diesem Abschnitt finden Sie in der Datenbank *Kapitel5_ADO.mdb*.

Das Objektmodell

Das ADO-Objektmodell setzt sich aus den folgenden Komponenten zusammen:

Objekte:

- *Connection*
- *Command*
- *Parameter*
- *Recordset*
- *Field*
- *Error*
- *Property*

Auflistungen:

- *Errors*
- *Fields*
- *Parameters*
- *Properties*

Ereignisse (*Connection*-Objekt):

- *BeginTransComplete, CommitTransComplete, RollbackTransComplete*
- *WillConnect, ConnectComplete, Disconnect*



- *WillExecute, ExecuteComplete*
- *InfoMessage*

Ereignisse (*Recordset*):

- *FetchProgress, FetchComplete*
- *WillChangeField, FieldChangeComplete*
- *WillMove, MoveComplete, EndOfRecordset*
- *WillChangeRecord, RecordChangeComplete*
- *WillChangeRecordset, RecordsetChangeComplete*

Auf Daten zugreifen

Im Prinzip läuft der Zugriff auf eine Datenquelle immer nach einem einfachen Schritt-für-Schritt-Schema ab:

- Herstellen einer Verbindung (*Connection*-Objekt) zu einer Datenquelle. Optional: Einleiten einer Transaktion.
- Optional: Anlegen eines Objekts, das eine SQL-Anweisung repräsentiert (*Command*-Objekt).
- Optional: Festlegen von Spalten, Tabellen und Werten für die SQL-Anweisung als variable Parameter (*Parameters*-Auflistung).
- Ausführen eines Befehls (*Connection*-, *Command*-, *Recordset*-Objekte).
- Speichern der zurückgegebenen Zeilen (Datensätze) in einem Cache (*Recordset*).
- Optional: Anlegen einer Ansicht des Caches zum Sortieren, zum Filtern der Daten und zur Navigation (*Recordset*).
- Bearbeiten der Daten: Hinzufügen, Löschen oder Ändern von Zeilen und Spalten (*Recordset*).
- Falls erforderlich, Datenquelle mit den Änderungen im Cache aktualisieren (*Recordset*).
- Falls eine Transaktion eingeleitet worden ist, Änderungen akzeptieren oder verwerfen; Transaktion abschließen (*Connection*).





Eine Datenverbindung herstellen (*Connection*)

Damit Sie auf eine Datenquelle zugreifen können, müssen Sie zunächst eine Verbindung zu ihr herstellen. Die zu verbindende Datenquelle wird in einem *Connection-String* festgelegt. Dieser kann je nach Datenquelle bzw. OLE-DB-Provider unterschiedlich aussehen. Standardmäßig wird eine Verbindung über die *Open*-Methode eines *Connection*-Objekts hergestellt:

```
Connection.Open <ConnectionString>, <UserID>,  
<Passwort>, <OpenOptions>
```

Sie können aber auch eine Abkürzung wählen, indem Sie die *Open*-Methode eines *Recordset*-Objekts aufrufen und dabei zugleich die Verbindung näher spezifizieren:

```
Recordset.Open <Quelle>, <ActiveConnection>,  
<CursorType>, <LockType>, <Optionen>
```

Als *ActiveConnection* können Sie sowohl einen literalen Connection-String, eine String-Variable oder auch ein bereits vorhandenes *Connection*-Objekt, das eine bereits geöffnete Verbindung repräsentiert, übergeben. Letzteres kann auch die von einer bereits in Access geöffneten Datenbank zur Verfügung gestellte aktive Verbindung (*Connection-Eigenschaft* des *CurrentProject*-Objekts) sein:

```
Sub ConnectionÖffnen()  
  
Dim con As ADODB.Connection  
  
Set con = Application.CurrentProject.Connection  
  
End Sub
```

Neben dieser Abkürzung können Sie alternativ beim *Connection*-Objekt lediglich die reine Datenquellenangabe im Connection-String übergeben und die übrigen benötigten Angaben wie UserID oder Passwort sowohl in den entsprechenden Eigenschaften des *Connection*-Objekts oder als Parameter setzen.

Einen Befehl festlegen (*Command*)

Im nächsten Schritt legen Sie zur Ausführung einer Anweisung an die Datenquelle ein *Command*-Objekt an. Hier können Sie zum einen in einer Datenbank vorhandene Tabellen oder Abfragen einsetzen oder auf gespeicherte SQL-Prozeduren zurückgreifen. Meistens werden Sie jedoch eine auszuführende SQL-Anweisung als String übergeben:

```
Sub CommandAnlegen()  
  
Dim cmd As ADODB.Command  
  
Set cmd = New ADODB.Command
```





```
cmd.CommandText = "SELECT * FROM Kunden"
```

End Sub

Nach der Ausführung gibt das *Command*-Objekt ein *Recordset*-Objekt zurück, das die abgefragten Datensätze (Tabellenzeilen) enthält.

Eine parametrisierte SQL-Abfrage sähe dementsprechend so aus:

```
cmd.CommandText = "SELECT * FROM Kuden WHERE Name = ?"
```

Den konkreten Parameter legen Sie fest, indem Sie ein *Parameter*-Objekt anlegen, dann in dessen Eigenschaften den Parameter spezifizieren und das Objekt schließlich in die *Parameter*-Auflistung einfügen. Soll beispielsweise nach dem Kundenamen *Meier* gesucht werden, würden Sie das *Parameter*-Objekt wie folgt anlegen:

```
Dim prm As ADODB.Parameter

Set prm = New ADODB.Parameter
prm.Name = "Name"
prm.Type = adVarChar
prm.Direction = adInput
prm.Size = 40
prm.Value = "Meier"
cmd.Parameters.Append prm
```

Auch hier gibt es wieder eine Alternative. Sie können der *Parameter*-Auflistung des *Command*-Objekts ein *Parameter*-Objekt über die *Append*-Methode der Auflistung direkt anfügen und dabei den Parameter spezifizieren:

```
cmd.Parameters.Append cmd.CreateParameter _
    "Name", adVarChar, adInput, 40, "Meier"
```

Bei der Verwendung von parametrisierten SQL-Anweisungen gibt es allerdings eine Einschränkung. Sie müssen dabei die *Execute*-Methode des *Command*-Objekts ausführen. Jedoch können Sie dann keinen Cursortyp und keine *Lock*-Optionen für das *Recordset*-Objekt festlegen.

Den Befehl ausführen

Ein *Recordset*-Objekt mit den gewünschten Datensätzen erhalten Sie schließlich, indem Sie entweder die *Execute*-Methode des *Connection*-Objekts oder des *Command*-Objekts oder die *Open*-Methode eines explizit angelegten *Recordset*-Objekts ausführen:

```
Connection.Execute(<CommandText>, <RecordsAffected>, _
    <Options>)
Command.Execute(<RecordsAffected>, <Parameters>, _
    <Options>)
```





```
Recordset.Open <Source>, <ActiveConnection>, _  
    <CursorType>, <LockType>, <Options>
```

Zur Ausführung jeder dieser drei Alternativen wird eine Verbindung zu einer Datenquelle benötigt:

- *Connection.Execute*: Es wird die im *Connection*-Objekt selbst festgelegte Verbindung verwendet.
- *Command.Execute*: Es wird das von der *ActiveConnection*-Eigenschaft des *Command*-Objekts zurückgegebene *Connection*-Objekt verwendet.
- *Recordset.Open*: Es wird der angegebene *ConnectionString* oder das von der *ActiveConnection*-Eigenschaft des *Recordset*-Objekts zurückgegebene *Connection*-Objekt verwendet.

Die drei Alternativen unterscheiden sich auch darin, wie der auszuführende Befehl festgelegt wird:

- *Connection.Execute*: Hier ist der Befehl ein String.
- *Command.Execute*: Hier wird der Befehl in der Eigenschaft *CommandText* festgelegt. Im Befehl enthaltene Parameter (?) werden im Argument *Parameters* (einem Variant-Array) spezifiziert.
- *Recordset.Open*: Hier wird der Befehl im *Source*-Argument übergeben. Dies kann sowohl ein String als auch ein *Command*-Objekt sein.

Alle drei Alternativen haben jeweils ihre Vor- und Nachteile:

- Die *Execute*-Methoden sind in erster Linie für Befehle gedacht, die keine Daten zurückliefern sollen. Sie sind jedoch nicht darauf beschränkt.
- Die Aufrufe beider *Execute*-Methoden werden schnell ausgeführt. Sie liefern jedoch nur schreibgeschützte *Forward Recordset*-Objekte zurück.
- Beim Aufruf von *Command.Execute* können leicht wiederverwendbare parametrisierte Befehle verwendet werden.
- Beim Aufruf der *Open*-Methode des *Recordset*-Objekts können der Cursortyp (die Strategie des Datenzugriffs) und der *Lock*-Typ festgelegt werden.

Sowohl geöffnete *Connection*-Objekte als auch *Command*-Objekte werden automatisch geschlossen, wenn die Lebensdauer der betreffenden Objektvariablen beendet ist. Das Schließen kann aber auch mithilfe der *Close*-Methode vorzeitig erfolgen. Dabei wird das Objekt jedoch nicht aus dem Speicher entfernt. Sie können daher nach dem Schließen verschiedene Eigenschaften ändern und das betreffende Objekt anschließend erneut öffnen. Um ein Objekt vollständig zu eliminieren, setzen Sie seine Objektvariable auf *Nothing*.



```
Sub ObjekteSchließen()

Dim Rs As ADODB.Recordset

Set Rs = New ADODB.Recordset

Rs.Open "SELECT * FROM Kunden", _
        CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic

"Datensätze bearbeiten ..."
While Not Rs.EOF

    Debug.Print Rs!Name

    Rs.MoveNext

Wend

Rs.Close

End Sub
```

Bearbeiten der Datensätze

Der Zugriff auf die Daten einer Datenquelle erfolgt innerhalb von VBA mithilfe von *Recordset*-Objekten, die Sie entweder explizit anlegen oder die von einer *Execute*-Methode zurückgegeben werden. Da diese Objekte nichts anderes als eine Gruppe von Datensätzen repräsentieren, werden sie auch als „Datensatzgruppen“ bezeichnet. Ähnlich wie Formulare haben auch Datensatzgruppen einen Datensatzzeiger, der auf den aktuellen Datensatz zeigt. Mithilfe spezieller Methoden kann dieser Datensatzzeiger auf andere Datensätze verschoben werden, so dass die Bearbeitung jedes einzelnen Datensatzes möglich ist.

Sie können mittels der *Move*-Methoden (*MoveFirst*, *MoveLast*, *MovePrevious*, *MoveNext*) oder der *Find*-Methode durch die Datensätze navigieren. Dabei ist immer genau ein Datensatz der jeweils aktuelle Datensatz, den Sie gerade bearbeiten können.

Über die *Filter*-Eigenschaft können Sie festlegen, welche Datensätze tatsächlich zur Verfügung stehen sollen, sozusagen für Sie „sichtbar“ und damit navigierbar sind.

Das *Recordset*-Objekt stellt eine *Fields*-Auflistung zur Verfügung, die die abgefragten Felder der Datensätze enthält. Auf die Werte der Felder eines Datensatzes greifen Sie über die (Standard-)Eigenschaft *Value* der in dieser Auflistung enthaltenen *Field*-Objekte zu. Bei den Methoden *GetRows* und *Update* können Sie alternativ dazu auf alle Felder zugleich zugreifen.





Zugriff auf Datensätze

Die eigentliche Bearbeitung der von einer Datenquelle abgefragten Datensätze erfolgt über das *Recordset*-Objekt.

Verschieben des Datensatzzeigers

Wie bereits erwähnt, besitzt jede einzelne Datensatzgruppe (*Recordset*) einen individuellen Datensatzzeiger, der den aktuellen Datensatz kennzeichnet. Nur dieser aktuelle Datensatz erlaubt das Auslesen oder Verändern von Feldinhalten. Wenn Sie auf mehrere Datensätze zugreifen möchten, müssen Sie den Datensatzzeiger nacheinander auf die gewünschten Datensätze positionieren. Zu diesem Zweck stehen die vier *Move*-Methoden zur Verfügung:

Die *Move*-Methoden



Methode	Beschreibung
<i>MoveFirst</i>	Positioniert auf den ersten Datensatz.
<i>MoveLast</i>	Positioniert auf den letzten Datensatz.
<i>MoveNext</i>	Positioniert auf den folgenden Datensatz.
<i>MovePrevious</i>	Positioniert auf den vorhergehenden Datensatz.

Tabelle 5.10



Von der Wirkung sind die vier *Move*-Methoden mit den Navigationsschaltflächen vergleichbar, die Sie vom Wechseln des aktuellen Datensatzes in Formularen kennen. Das folgende Beispiel legt eine Datensatzgruppe auf der Grundlage der Tabelle *Kunden* in einer Datenbank *Stammdaten* an und positioniert anschließend auf den letzten Datensatz dieser Gruppe.

```
Sub MoveBeispiel()  
  
Dim Rs As ADODB.Recordset  
  
Set Rs = New ADODB.Recordset  
  
Rs.Open "SELECT * FROM Kunden", _  
        CurrentProject.Connection, _  
        adOpenDynamic, adLockOptimistic  
  
Debug.Print Rs!Name
```





```
Rs.MoveLast
Debug.Print Rs!Name
```

```
Rs.Close
```

```
End Sub
```

Der Datensatzzeiger eines *Recordset*-Objekts wird zu Beginn automatisch auf den ersten Datensatz positioniert, wenn die Datensatzgruppe mindestens einen Datensatz enthält.

Eine Besonderheit tritt auf, wenn der Datensatzzeiger momentan auf den letzten Datensatz zeigt und Sie die *MoveNext*-Methode ausführen. Obwohl hier kein Datensatz mehr vorhanden ist, wird die Positionierung ohne Fehlermeldung ausgeführt. Gleichzeitig wird der Datensatzzeiger hinter den letzten Datensatz positioniert und ist damit ungültig. Wenn nun ein Zugriff auf den (nicht vorhandenen) Datensatz erfolgt oder erneut die *MoveNext*-Methode ausgeführt wird, wird ein Fehler ausgelöst. Ein ähnliches Verhalten zeigt die Methode *MovePrevious*. Zeigt der Datensatzzeiger auf den ersten Datensatz und wird die Methode *MovePrevious* ausgeführt, wird der Datensatzzeiger vor den ersten Datensatz positioniert. Ein Zugriff auf den (auch hier nicht vorhandenen) Datensatz oder ein erneuter Aufruf der Methode *MovePrevious* löst ebenfalls einen Fehler aus.

Eigenschaften *EOF* und *BOF*

Um diesen Zustand zu ermitteln, stellt das *Recordset*-Objekt die *EOF*-Eigenschaft zur Verfügung (*EOF* = **End Of File**). Diese Eigenschaft gibt normalerweise den Wert *False* zurück. Sollte der Datensatzzeiger allerdings hinter den letzten Datensatz positioniert werden, besitzt sie den Wert *True*.

Genauso verhält es sich mit der Eigenschaft *BOF* (**Begin Of File**). Sie gibt den Wert *True* zurück, sobald sich der Datensatzzeiger vor dem ersten Datensatz befindet.

Im folgenden Beispiel werden die Datensätze eines *Recordset*-Objekts so lange durchlaufen und auf das Feld *Name* zugegriffen, bis der letzte Datensatz erreicht worden ist:

```
Do While Rs.EOF = False
    Debug.Print Rs.Fields("Name")
    Rs.MoveNext
Loop
```

Hier wird eine *Do...Loop*-Schleife verwendet, bei der die Bedingung bereits beim Eintritt in die Schleife überprüft wird. Falls das *Recordset*-Objekt keinen einzigen Datensatz enthalten sollte, wird die *EOF*-Eigenschaft (und auch die *BOF*-Eigenschaft) von vornherein auf *True* gesetzt. In diesem Fall wird die Schleife erst gar nicht durchlaufen. Wäre dies nicht der Fall, würde der Zugriff auf das Feld *Name* oder auch der Aufruf der *MoveNext*-Methode einen Fehler ausgeben.





Die Eigenschaft *RecordCount*

Eine weitere Möglichkeit zur Iteration über alle Datensätze bietet die Eigenschaft *RecordCount* des *Recordset*-Objekts. Diese Eigenschaft enthält die Anzahl der Datensätze der Datensatzgruppe. Sie ist jedoch mit Vorsicht zu handhaben, da sie nur unter bestimmten Bedingungen ein verwertbares Ergebnis liefern kann.

Kann ADO die Anzahl der Datensätze nicht genau bestimmen, wird -1 zurückgegeben.

Unterstützt das *Recordset*-Objekt vorausschauende Positionierung oder Bookmarks (d.h. ein Aufruf der Methode *Supports* mit den Argumenten *adApproxPosition* oder *adBookmark* gibt *True* zurück), enthält die *RecordCount*-Eigenschaft die tatsächliche Anzahl der Datensätze, unabhängig davon, ob das *Recordset*-Objekt bereits vollständig gefüllt ist. Wird die vorausschauende Positionierung nicht unterstützt, kann die Verwendung der *RecordCount*-Eigenschaft eine erhebliche Ressourcenbelastung zur Folge haben, da erst alle Datensätze empfangen werden müssen, um einen korrekten Wert für *RecordCount* ermitteln zu können.

Beachten Sie, dass der Datentyp der *RecordCount*-Eigenschaft *Long* ist und die Anzahl dementsprechend in einer *Long*-Variablen gespeichert werden muss. *Integer*-Variablen sind hierzu ungeeignet, da sie nur einen Maximalwert von ca. 32.000 unterstützen. Eine Tabelle kann jedoch wesentlich mehr Datensätze enthalten.

Zugriff auf Datenfelder

Das *Recordset*-Objekt verfügt über die *Fields*-Auflistung, die den Zugriff auf die einzelnen Datenfelder erlaubt. Mit der *Fields*-Auflistung arbeiten Sie wie mit jeder anderen Auflistung. Der Zugriff erfolgt folgendermaßen:

```
Recordset.Fields("<Feldname>")
```

Um nun den Inhalt eines bestimmten Datenfeldes zu ermitteln, muss die *Value*-Eigenschaft verwendet werden. Das folgende Beispiel gibt den Inhalt eines Feldes *Name* zurück:

```
Wert = Rs.Fields("Name").Value
```

Neben dem Zugriff über den Feldnamen kann mithilfe der *Fields*-Auflistung auch über die Position eines Feldes innerhalb der Auflistung Bezug genommen werden. Das folgende Beispiel listet die Feldnamen und die entsprechenden Inhalte des aktuellen Datensatzes eines *Recordset*-Objekts im Testfenster auf:

```
Sub Feldzugriff()
```

```
Dim Rs As ADODB.Recordset  
Dim i As Integer
```

```
Set Rs = New ADODB.Recordset
```





```
Rs.Open "SELECT * FROM Kunden", _
        CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic

With Rs.Fields

    For i = 0 To .Count - 1
        Debug.Print .Item(i).Name, .Item(i).Value
    Next

End With

Rs.Close

End Sub
```

Alternativ kann auch eine *For...Each*-Schleife eingesetzt werden:

```
Sub Feldzugriff2()

Dim Rs As ADODB.Recordset
Dim Feld As ADODB.Field

Set Rs = New ADODB.Recordset

Rs.Open "SELECT * FROM Kunden", _
        CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic

For Each Feld In Rs.Fields
    Debug.Print Feld.Name, Feld.Value
Next

Rs.Close

End Sub
```

Beachten Sie, dass der Datensatzzeiger des *Recordset*-Objekts auf einen gültigen Datensatz zeigen muss, da der Zugriff auf die Datenfelder sonst einen Fehler auslösen würde. Sie können dies überprüfen, indem Sie zuvor die Werte der Eigenschaften *EOF* und *BOF* auswerten:

```
Sub Feldzugriff2()

Dim Rs As ADODB.Recordset
Dim Feld As ADODB.Field

Set Rs = New ADODB.Recordset
```



```

Rs.Open "SELECT * FROM Kunden", _
        CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic

If Not (Rs.EOF Or Rs.BOF) Then

    For Each Feld In Rs.Fields
        Debug.Print Feld.Name, Feld.Value
    Next

End If

Rs.Close

End Sub

```

Zuweisung an Variablen

Wenn Sie den Inhalt eines Feldes einer VBA-Variablen zuweisen möchten, müssen Sie darauf achten, dass der betreffende Felddatentyp mit dem VBA-Datentyp kompatibel ist. Um diesen Problemen aus dem Weg zu gehen, wird normalerweise der Datentyp *Variant* verwendet. Ein weiterer Vorteil dieses Datentyps ist, dass es bei Datenfeldern mit dem Inhalt Null keine Probleme gibt.

Wenn Sie lieber (oder besser!) einen der anderen Grundtypen vorziehen – beispielsweise zur Optimierung der Arbeitsgeschwindigkeit –, müssen Sie die passenden Datentypen gemäß folgender Tabelle auswählen:

Kompatible Datentypen

Felddatentyp	Feldgröße	VBA-Grunddatentyp
<i>AutoWert</i>	Long Integer	Long
	Replikations-ID	Variant
<i>Währung</i>	–	Currency
<i>Datum/Zeit</i>	–	Date
<i>Memo</i>	–	String
<i>Zahl</i>	Byte	Byte
	Integer	Integer
	Long Integer	Long



Kompatible Datentypen (Forts.)

Felddatentyp	Feldgröße	VBA-Grunddatentyp
	Single	Single
	Double	Double
<i>OLE-Objekt</i>	–	Variant (oder indirekt Byte()-Array)
<i>Text</i>	–	String
<i>Ja/Nein</i>	–	Boolean
<i>Hyperlink</i>	–	String

Tabelle 5.11

Der Zugriff auf Felder des Typs *OLE-Objekt* ist von VBA aus nur eingeschränkt möglich. Bei Feldern des Typs *OLE-Objekt* handelt es sich eigentlich um Datenfelder zur Speicherung so genannter *BLOBs* (**B**inary **L**arge **O**bjects). BLOBs sind nichts anderes als eine lange Sequenz einzelner Bytes, die abhängig vom Kontext unterschiedlich interpretiert werden können. Daher können sie nicht direkt einem eindeutigen Datentyp zugewiesen werden. Weisen Sie den Inhalt eines BLOBs zunächst einer *Variant*-Variablen und anschließend deren Inhalt einem zuvor dynamisch dimensionierten *Byte*-Array zu, um auf die einzelnen Bytes zugreifen zu können:

```
Dim Blob As Variant
Dim Bytes() As Byte

Blob = Rs.Fields("IrgendeinBLOB").Value
Bytes = Blob
```

Durchsuchen von Datensatzgruppen

Eine Datensatzgruppe kann nach einem bestimmten Datensatz durchsucht werden. Hierzu dient die Methode *Find*, der als Kriterium ein String übergeben wird, der dem *WHERE*-Teil einer SQL-Abfrage entspricht.

```
Rs.Find <Kriterium>, <Ueberspringen>, <Suchrichtung>,
<Start>
```



Die folgende Tabelle zeigt einige Möglichkeiten für gültige Suchkriterien:

Beispiele für Suchkriterien der *Find-Methode*

Beispielkriterien

Name = "Müller"

Name Like "M*ier"

Name = "Müller" *AND* *Postleitzahl* = "40595"

Gehalt < 30000

KundenNr = 4567

Tabelle 5.12

Beim Erstellen des Suchkriteriums in VBA ergibt sich das Problem, dass dort Zeichenketten verwendet werden (z.B. *Müller*). Solche Zeichenketten, die in eine andere Zeichenkette eingebettet sind, dürfen in diesem Fall nicht innerhalb normaler Anführungszeichen ("..."), sondern müssen innerhalb einfacher Anführungszeichen ('...' bzw. Apostroph-Zeichen) angegeben werden. Eine Alternative besteht darin, dass Sie – wie in dem folgenden Beispiel – zwei doppelte Anführungszeichen angeben:

```
Name = " "Müller" "
```

Das optionale Argument *Ueberspringen* der *Find-Methode* gibt an, wie viele Datensätze ab der aktuellen Position des Datensatzzeigers übersprungen werden sollen, bevor mit der Suche begonnen wird.

Im optionalen Argument *Suchrichtung* geben Sie an, in welcher Richtung die Datensätze von der aktuellen Position des Datensatzzeigers aus durchsucht werden sollen. Mögliche Werte sind *adSearchForward* und *asSearchBackward*.

Im ebenfalls optionalen Argument *Start* können Sie einen Bookmark übergeben, ab dem die Datensätze – anstelle der aktuellen Position des Datensatzzeigers – durchsucht werden sollen.

Wird der gewünschte Datensatz gefunden, wird der Datensatzzeiger des *Recordset*-Objekts auf diesen gesetzt, andernfalls wird er ans Ende gesetzt.

```
Sub Suchen()
```

```
Dim Rs As ADODB.Recordset
```

```
Set Rs = New ADODB.Recordset
```





```
Rs.Open "SELECT * FROM Kunden", _
        CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic

With Rs

    .Find "Name Like 'M*'"

    If Not .EOF Then

        "Gefunden
        Debug.Print !Name

    Else

        Debug.Print "Der Datensatz wurde _
        nicht gefunden!"

    End If

End With

Rs.Close

End Sub
```

Sortieren von Datensätzen

Beim Positionieren des Datensatzzeigers mithilfe der *Move*-Methoden spielt die Reihenfolge der Datensätze innerhalb des *Recordset*-Objekts eine Rolle. Die tatsächliche Reihenfolge der Datensätze wird von der Datenquelle festgelegt und kann bei unterschiedlichen Datenquellentypen voneinander abweichen. Meistens werden die Datensätze in der Reihenfolge der Eingabe abgelegt und zurückgegeben, wenn nicht schon beim Einfügen neuer Datensätze in die Datenquelle definierte Regeln verarbeitet werden.

Der einfachste Weg, eine sortierte Reihenfolge der Datensätze zu erhalten, ist die Angabe eines Sortierkriteriums bereits im SQL-Abfrage-String. Dann entspricht bereits die physische Reihenfolge der Datensätze im *Recordset*-Objekt dem Sortierkriterium. Beispiel:

```
Dim SQL As String

Set Rs = New ADODB.Recordset

SQL = "SELECT * FROM Kunden ORDER BY Postleitzahl"

Rs.Open SQL, CurrentProject.Connection, _
        adOpenDynamic, adLockOptimistic
```





Auf gleiche Weise kann jedoch auch die Sortierreihenfolge für den Zugriff auf die Datensätze des *Recordset*-Objekts nachträglich beliebig geändert werden. Dazu wird der *Sort*-Eigenschaft des *Recordset*-Objekts ein String mit einer durch Kommas getrennten Aufzählung der Feldnamen zugewiesen, nach denen sortiert werden soll. Diese Methode ist jedoch nur bei bestimmten OLE-DB-Providern möglich und außerdem sehr langsam. Der Standard-JET-Provider unterstützt diese Methode beispielsweise nicht.

Filtern von Datensätzen

Gerade bei der Verarbeitung von Daten aus einer nicht lokal vorhandenen Datenquelle kann die Filterung der einmal in ein *Recordset*-Objekt übertragenen Datensätze nicht nur Zeit sparen, sondern auch den Verbindungsweg zur Datenquelle von unnötigem Übertragungsverkehr entlasten. Solange die eigentlichen Auswahl- und Abfragekriterien nicht grundsätzlich geändert werden und eine einmal übertragene Datensatzmenge lediglich weiter spezifiziert und eingegrenzt werden soll, ist es meistens günstiger und effizienter, die Datensätze eines *Recordset*-Objekts anhand weiterer, nachträglich spezifizierter Kriterien zu filtern und die gefilterte Datensatzgruppe weiterzuverarbeiten.

Eine Filterung ändert nicht die physisch in einem *Recordset*-Objekt vorhandene Anzahl der Datensätze. Es werden lediglich die nicht dem Filterkriterium entsprechenden Datensätze ausgeblendet. Das Filterkriterium kann jederzeit geändert und auch wieder aufgehoben werden.

Als Filterkriterium kann ein String mit einer *SQL-WHERE*-Bedingung gesetzt werden. Der Aufbau der *SQL-WHERE*-Bedingung erfolgt nach den gleichen Regeln wie für das Suchkriterium der *Find*-Methode.

Das folgende Beispiel filtert alle Kunden mit dem Anfangsbuchstaben *M* und gibt diese im Direktfenster aus:

```
Sub Filter()  
  
Dim Rs As ADODB.Recordset  
Dim SQL As String  
  
Set Rs = New ADODB.Recordset  
  
SQL = "SELECT * FROM Kunden"  
  
Rs.Open SQL, CurrentProject.Connection, _  
    adOpenDynamic, adLockOptimistic  
  
With Rs  
  
    .Filter = "Name Like 'M*'"  
  
While Not .EOF
```





```

        Debug.Print !Name

        .MoveNext

    Wend

End With

Rs.Close

End Sub

```

Beachten Sie, dass sich mit dem Setzen eines Filters eine ganze Reihe von Eigenschaften des betreffenden *Recordset*-Objekts ändern können.

Zunächst wird mit dem Setzen des Filters der Datensatzzeiger auf den ersten Datensatz der nach dem Filterkriterium verbleibenden Datensätze gesetzt. Wird der Filter zurückgesetzt (das kann auch durch Setzen der Konstanten *adFilterNone* anstelle eines Filterkriteriums erfolgen), wird der Datensatzzeiger auf den ersten Datensatz im *Recordset*-Objekt gesetzt.

Dementsprechend können sich auch die von den Eigenschaften *AbsolutePosition*, *AbsolutePage*, *RecordCount* (siehe obenstehendes Beispiel) und *PageCount* zurückgegebenen Werte ändern.

Ändern von Datensätzen

Zum Ändern von Datensätzen ist es erforderlich, dass das betreffende *Recordset* als dynamisches *Recordset* geöffnet ist. Dies erledigt beispielsweise die folgende Anweisung:

```

Dim Rs As ADODB.Recordset

Rs.Open "SELECT * FROM Mitarbeiter", con, _
        adOpenDynamic, adLockOptimistic

```

Die Änderung von Feldinhalten eines Datensatzes eines *Recordset*-Objekts ist eine einfache Angelegenheit. Sie ändern einfach die gewünschten Feldinhalte und rufen anschließend die *Update*-Methode des *Recordset*-Objekts auf. Der Datensatzzeiger des *Recordset*-Objekts bleibt nach dem Aufruf der *Update*-Methode unverändert.

```

Sub Datensatzänderung()

Dim Rs As ADODB.Recordset
Dim SQL As String

Set Rs = New ADODB.Recordset

SQL = "SELECT * FROM Kunden WHERE Name = 'Meier'"

```





```
Rs.Open SQL, CurrentProject.Connection, _  
        adOpenDynamic, adLockOptimistic
```

```
With Rs
```

```
    !Land = "Deutschland"  
    .Update
```

```
End With
```

```
Rs.Close
```

```
End Sub
```

Ändern Sie den Datensatzzeiger oder fügen Sie einen neuen Datensatz ein, nachdem Sie Feldinhalte des aktuellen Datensatzes geändert haben, ohne zuvor die *Update*-Methode aufzurufen, so wird der geänderte Datensatz automatisch aktualisiert. Wollen Sie dagegen die Änderungen verwerfen, müssen Sie zuerst die *CancelUpdate*-Methode aufrufen.

```
Sub Datensatzänderung2()
```

```
Dim Rs As ADODB.Recordset  
Dim SQL As String
```

```
Set Rs = New ADODB.Recordset
```

```
SQL = "SELECT * FROM Kunden WHERE Name = "Müller""
```

```
Rs.Open SQL, CurrentProject.Connection, _  
        adOpenDynamic, adLockOptimistic
```

```
With Rs
```

```
    !Land = "Deutschland"
```

```
    If MsgBox("Änderungen speichern?", vbYesNo) = _  
        vbYes Then  
        .Update  
    Else  
        .CancelUpdate  
    End If
```

```
End With
```

```
Rs.Close
```

```
End Sub
```



Über die Eigenschaft *EditMode* können Sie feststellen, ob Feldinhalte geändert wurden und ob der Datensatz bereits mit einem Aufruf der *Update*-Methode gesichert worden ist. Gibt die Eigenschaft *EditMode* den Wert *adEditInProgress* zurück, sind die Änderungen noch nicht gesichert worden. Ist der Wert *adEditNone*, sind die Änderungen gesichert.

Anstatt Feldinhalte eines oder mehrerer Felder einzeln nacheinander zu ändern, können Sie die Änderungen auch zugleich als Argumente der *Update*-Methode zuweisen. Dabei übergeben Sie in einem Variant-Array zuerst die Namen aller Felder, die geändert werden sollen, und anschließend in einem zweiten Variant-Array die neuen Werte für diese Felder. Beim Aufruf der *Update*-Methode mit diesen Variant-Arrays als Argumente bleibt die *EditMode*-Eigenschaft unverändert.

```
Sub Datensatzänderung3()

Dim Rs As ADODB.Recordset
Dim SQL As String

Set Rs = New ADODB.Recordset

SQL = "SELECT * FROM Kunden WHERE Name = "Müller""

Rs.Open SQL, CurrentProject.Connection, _
    adOpenDynamic, adLockOptimistic

With Rs

    .Update Array("Land", "Telefonnummer"), _
        Array("Deutschland", "0211-000000")

End With

Rs.Close

End Sub
```

Wichtig ist, dass in beiden Arrays die Zuordnung der Felder korrekt ist und auch beide Arrays die gleiche Anzahl an Elementen enthalten. Ein Fehler wird nur dann ausgelöst, wenn die Anzahl in beiden Arrays nicht übereinstimmt oder wenn eine falsche Zuordnung der Feldnamen und der neuen Werte dazu führen sollte, dass einem Feld ein Wert eines für dieses Feld ungültigen Datentyps zugewiesen würde.

Beachten Sie auch, dass eine Änderung der Feldinhalte über diese Variante des *Update*-Aufrufs nicht mithilfe der *CancelUpdate*-Methode rückgängig gemacht werden kann.





Hinzufügen von Datensätzen

Mit einem Aufruf der Methode *AddNew* fügen Sie einen neuen leeren Datensatz in das *Recordset*-Objekt ein, der nun zum aktuellen Datensatz des *Recordset*-Objekts wird. Danach können Sie die Felder des Datensatzes mit Inhalten füllen und anschließend die *Update*-Methode aufrufen, um den Datensatz zu aktualisieren. Auch nach dem Aufruf der *Update*-Methode bleibt der neue Datensatz der aktuelle Datensatz des *Recordset*-Objekts.

Beim Hinzufügen neuer Datensätze ist es ebenfalls wichtig, dass es sich bei dem *Recordset* um ein dynamisches *Recordset* handelt. Ansonsten tritt bei der Ausführung der *AddNew*-Methode ein Fehler auf.

Nach dem Aufruf der *AddNew*-Methode gibt die Eigenschaft *EditMode* so lange den Wert *adEditAdd* zurück, bis der Datensatz mit dem Aufruf der *Update*-Methode gesichert worden ist.

Auch hier gilt wie bei der *Update*-Methode, dass eine Änderung des Datensatzzeigers oder das erneute Einfügen eines Datensatzes die bis zu diesem Zeitpunkt vorgenommenen Zuweisungen von Feldinhalten automatisch aktualisiert.

Auch bei der *AddNew*-Methode können Sie wie bei der *Update*-Methode alternativ die Namen der zu füllenden Felder und die Werte für diese Felder zugleich als Argumente in zwei Variant-Arrays übergeben. Hier gelten die gleichen Regeln wie bei der *Update*-Methode. Auch wird hierbei die *EditMode*-Eigenschaft nicht auf *adEditAdd* gesetzt.

```
Sub DatensätzeHinzufügen()  
  
Dim Rs As ADODB.Recordset  
Dim SQL As String  
  
Set Rs = New ADODB.Recordset  
  
SQL = "Kunden"  
  
Rs.Open SQL, CurrentProject.Connection, _  
        adOpenDynamic, adLockOptimistic  
  
With Rs  
    .AddNew  
        !Name = "Wagner"  
        !Vorname = "Hans"  
        !Postleitzahl = "40211"  
        !Ort = "Düsseldorf"  
    .Update  
  
End With
```



```
Rs.Close
```

```
End Sub
```

Löschen von Datensätzen

Mit einem Aufruf der Methode *Delete* des *Recordset*-Objekts löschen Sie standardmäßig (oder mit Angabe des Arguments *adAffectCurrent*) den aktuellen Datensatz.

Geben Sie als Argument den Wert *adAffectAll* an, werden alle Datensätze des *Recordset*-Objekts gelöscht.

Haben Sie den aktuellen Datensatz gelöscht, zeigt der Datensatzzeiger zunächst weiterhin auf diesen Datensatz. Da er jedoch durch das Löschen bereits ungültig geworden ist, löst jeder Versuch, auf seine Felder zuzugreifen, einen Fehler aus. Wenn der Datensatzzeiger von dem gelöschten Datensatz wegbewegt worden ist, kann nicht mehr auf ihn zugegriffen werden.

```
Sub DatensätzeLöschen()
```

```
Dim Rs As ADODB.Recordset
```

```
Dim SQL As String
```

```
Set Rs = New ADODB.Recordset
```

```
SQL = "SELECT * FROM Kunden Where Name = "Wagner""
```

```
Rs.Open SQL, CurrentProject.Connection, _  
adOpenDynamic, adLockOptimistic
```

```
Rs.Delete
```

```
Rs.Close
```

```
End Sub
```

Fehler beim Datenzugriff

Um auf Fehler, die im Zusammenhang mit dem Zugriff auf Daten auftreten, besser reagieren zu können, stellt ADO eine spezielle Auflistung mit dem Namen *Errors* zur Verfügung. Innerhalb dieser Auflistung werden die zuletzt aufgetretenen Fehler beim Datenzugriff durch separate *Error*-Objekte repräsentiert.

Wenn im Zusammenhang mit einem ADO-Datenzugriff ein Fehler seitens der Datenquelle auftritt, wird mindestens ein *Error*-Objekt in der *Errors*-Auflistung abgelegt. Bei manchen Datenquellen werden auch mehrere *Error*-Objekte abgelegt, wenn datenquellenspezifisch mehrere Fehler im Zusammenhang mit einer ein-





zigen ADO-Operation auftreten sollten. Die *Errors*-Auflistung wird gelöscht und durch eine neue *Errors*-Auflistung ersetzt, wenn bei der nächsten ADO-Operation erneut Datenquellenfehler auftreten sollten.

Beachten Sie, dass die *Errors*-Auflistung nur für seitens der Datenquelle auftretende Fehler zuständig ist. ADO-interne Fehler oder Syntaxfehler u.Ä. lösen im Rahmen der normalen VBA-Fehlerbehandlung Fehler aus.

Das *Error*-Objekt

Jedes *Error*-Objekt verfügt über einen Satz an Eigenschaften, die Informationen über die Art und Quelle eines Fehlers enthalten.

- Die Eigenschaft *Description* enthält die Fehlerbeschreibung im Klartext.
- Die Eigenschaft *Number* gibt eine fehlerspezifische Fehlernummer zurück.
- Die Eigenschaft *Source* identifiziert das Objekt, das den Fehler ausgelöst hat.
- Die Eigenschaften *SQLState* und *NativeError* liefern zusätzliche Informationen von SQL-Datenquellen.

Sie können durch die *Errors*-Auflistung wie durch jede Auflistung iterieren, die darin enthaltenen *Error*-Objekte auswerten und entsprechend reagieren.

```
Dim Fehler As ADODB.Error

For Each Fehler in Errors
    Debug.Print Fehler.Description
Next
```

Einführung in SQL

In den vorangehenden Abschnitten dieses Kapitels haben Sie gelernt, wie Sie das *Recordset*-Objekt benutzen und mit Datensatzgruppen arbeiten können. Dabei kommen immer wieder Abfragen vor, die entweder als Abfragen im Datenbankfenster gespeichert sein müssen oder als SQL-Text angegeben werden. Die Angabe eines SQL-Textes hat den Vorteil, dass nicht für jede Datensatzgruppe, über die Sie auf Daten zugreifen wollen, eine eigene Abfrage gespeichert werden muss. Außerdem lassen sich Sortierungen, Gruppierungen und Filter mittels SQL-Anweisung direkt mit angeben, ohne den Umweg gehen zu müssen, eine zweite Datensatzgruppe zu verwenden.

Die folgende Beispielprozedur soll den allgemeinen Zugriff per SQL-Anweisung und Verwendung eines *Recordset*-Objekts deutlich machen.



```
Sub AbfragenÜberSqlText1 ()

    Dim DB As Database
    Dim R As Recordset

    Set DB = CurrentDb
    Set R = DB.OpenRecordset ("<SQL-Anweisung>")

End Sub
```

Für den Ausdruck *<SQL-Anweisung>* muss ein gültiger SQL-Text eingetragen werden. Da das *Recordset*-Objekt eine Zeichenkette erwartet, muss die Anweisung in Anführungszeichen gesetzt werden. Das Ergebnis der Abfrage ist in der Variablen *R* gespeichert und liegt dadurch als Datensatzgruppe vor. Über die Ihnen schon bekannten Methoden können Sie nun auf einzelne Datensätze und deren Inhalte zugreifen.

Die Verwendung eines SQL-Textes als Datenherkunft hat noch einen weiteren Vorteil. Die SQL-Anweisung kann in einer Variablen vom Datentyp *String* abgelegt werden. Dadurch lassen sich Abfragen sehr flexibel halten. So können zum Beispiel durch eine *Select Case*-Anweisung oder eine *If*-Abfrage im Prozedurcode je nach eintreffendem Fall verschiedene Abfragen erzeugt oder Abfragen aus mehreren verschiedenen Strings passend zusammengefügt werden. Näheres dazu erfahren Sie am Ende des Abschnitts *Auswahlabfragen definieren*, wenn Sie bereits wissen, wie man Auswahlabfragen definiert.

Der allgemeine Aufbau der Prozedur sieht dem der vorherigen sehr ähnlich; es wird nur zusätzlich eine Variable vom Datentyp *String* definiert.

```
Sub AbfragenÜberSqlText2 ()

    Dim DB As Database
    Dim R As Recordset
    Dim SQL As String

    SQL = "<SQL-Anweisung>"
    Set DB = CurrentDb
    Set R = DB.OpenRecordset (SQL)

End Sub
```

Ein weiteres Anwendungsgebiet von SQL-Anweisungen ist das *QueryDef*-Objekt. Mit ihm lassen sich bereits bestehende Abfragen öffnen oder neue Abfragen erzeugen und im Datenbankfenster speichern. Das *QueryDef*-Objekt und seine Verwendungsmöglichkeiten lernen Sie im *Kapitel 10 Design von Datenbankobjekten* kennen.

Um SQL in den oben beschriebenen Anwendungsgebieten nutzen zu können, sind tiefe Kenntnisse von SQL nicht unbedingt erforderlich. Sie können nämlich auch eine Abfrage vollständig im Abfragefenster, d.h. im QBE-Entwurfsbereich, entwerfen und testen. Mit der Auswahl des Befehls *Ansicht > SQL-Ansicht* öffnet Access ein





Fenster, in dem es die SQL-Anweisung mit der gleichen Funktionalität anzeigt. Die Übereinstimmung zwischen dem Abfrage- und dem SQL-Fenster geht sogar noch weiter. Access setzt nämlich im SQL-Fenster gemachte Änderungen automatisch auch im Abfragefenster um und umgekehrt.

SQL-Anweisung
kopieren

Der Trick besteht nun darin, die im SQL-Fenster angezeigte Anweisung zu markieren und sie in die Zwischenablage zu kopieren. Öffnen Sie dann das Modul, wechseln Sie in die entsprechende Prozedur und fügen Sie den SQL-Text an der passenden Stelle ein. Die Abfrage, die Sie im QBE-Entwurfsbereich entworfen haben, braucht nicht gespeichert zu werden, da sie ja nur als Erstellungshilfe für den SQL-Text diente.

Aber selbst wenn Sie SQL beherrschen, kann die beschriebene Methode die Arbeit erleichtern. Wenn Sie eine umfangreiche Abfrage entwerfen wollen, die womöglich mehrere Tabellen, Bedingungen oder Sortierungen beinhaltet, wird der SQL-Text lang und unübersichtlich, wodurch sich Fehler einschleichen können. Eine solche Abfrage kann wesentlich schneller ohne viel Tipparbeit in der QBE-Entwurfsansicht erstellt werden. Die entsprechende SQL-Anweisung, die dann kopiert werden kann, erhalten Sie wieder über den Menüpunkt *Ansicht > SQL-Ansicht*.

Hinweis



Die folgenden Abschnitte setzen voraus, dass Sie bereits mit dem Abfragefenster und der Bildung von Ausdrücken mit Access vertraut sind. Alle BeispielprozEDUREN und Beispielabfragen der einzelnen Abschnitte finden Sie in der Datenbank *Kapitel5.mdb* wieder. Die Prozeduren sind im Modul *Sq1* enthalten. Die Abfragen sind von eins bis neunzehn durchnummeriert. Die Nummer einer Abfrage finden Sie im Text in der Randbemerkung neben dem entsprechenden Abfragecode.

Hinweis



Die Beispiele wurden als Abfrage im Datenbankfenster gespeichert und nicht in Form von Prozeduren. Das soll die Übersichtlichkeit erhöhen, da der für das Verständnis der einzelnen Anweisungen überflüssige Prozedurcode fehlt. Wie Sie eine SQL-Anweisung auf ein *Recordset*-Objekt anwenden, wissen Sie ja bereits. Außerdem können Sie sich das Ergebnis der Abfrage direkt in der Datenblattansicht anzeigen lassen und müssen nicht die einzelnen Felder über eine Schleife im Direktfenster ausgeben.

Auswahlabfragen definieren

Die wohl am häufigsten benutzte und benötigte Abfrage ist die Auswahlabfrage. Mit ihr lassen sich Felder aus einer Tabelle selektieren. Zusätzlich können die Datensätze eine Sortierung erhalten, gruppiert oder gefiltert werden oder Datensätze aus anderen Datenbanken ermittelt werden.

Die *Select*-
Anweisung

Den Grundstein für Auswahlabfragen bildet die *Select*-Anweisung. In der einfachsten Form werden mit ihr die Felder einer oder mehrerer Tabellen bestimmt, die die entstehende Datensatzgruppe enthalten soll. Die Syntax der *Select*-Anweisung lautet wie folgt:

```
SELECT [ALL|DISTINCT|DISTINCTROW|TOP [n PERCENT]]]  
<Auswahlliste> FROM <Tabellenausdruck>
```



Zunächst wird eine Auswahlliste der Datenfelder und Ausdrücke angegeben, die selektiert werden sollen. Wenn alle Felder in die Datensatzgruppe übernommen werden sollen, muss hier ein Sternchen (*) stehen. Mit <Tabellenausdruck> bestimmen Sie eine oder mehrere Tabellen, die als Datenherkunft dienen sollen. Die SQL-Anweisung

```
SELECT Name, Vorname, Ort FROM Kunden;
```

bildet eine Datensatzgruppe, die die Datenfelder *Name*, *Vorname* und *Ort* der Tabelle *Kunden* und die entsprechenden Informationen aus allen Datensätzen beinhaltet.

Jede SQL-Anweisung muss mit einem Semikolon abgeschlossen werden. Das gilt für die *Select*-Anweisung und auch für alle noch folgenden Abfragetypen.

Über eine Variable vom Typ *Recordset* können Sie auf die Datensätze der Abfrage zugreifen. Das entsprechende Programm sieht folgendermaßen aus:



Hinweis

```
Sub Auswahlabfrage1 ()

    Dim DB As Database
    Dim R As Recordset

    Set DB = CurrentDb

    Rem Datensatzgruppe erzeugen
    Set R = DB.OpenRecordset("SELECT Name, Vorname, _
        Ort FROM Kunden")

    Rem Datensätze im Direktfenster ausgeben
    Do While R.EOF = False

        Debug.Print R!Name, R!Vorname, R!Ort

        R.MoveNext
    Loop

End Sub
```

Um festzulegen, welche Datensätze in die Datensatzgruppe aufgenommen werden, stellt die *Select*-Anweisung verschiedene Prädikate zur Verfügung, die optional mit angegeben werden können. Es stehen die reservierten Wörter *All*, *Distinct*, *Distinctrow* und *Top* zur Verfügung. Mit dem *All*-Prädikat werden alle Datensätze in das Recordset aufgenommen. *All* ist Standardwert und wird automatisch von Access verwendet, wenn Sie kein Prädikat angeben.

All





Distinct

Sollen Wiederholungen von identischen Werten in den Datenfeldern vermieden werden, ist das Prädikat *Distinct* zu verwenden. Die Angabe von *Distinct* entspricht dem Eintrag *Ja* bei der Abfrageeigenschaft *Keine Duplikate*. Bei der Angabe von *Distinct* berücksichtigt Access einen Datensatz nur dann, wenn die in *<Auswahlliste>* aufgeführten Felder und Ausdrücke unterschiedliche Werte annehmen. Wiederholungen werden unterdrückt.

Distinctrow

Die Angabe von *Distinctrow* führt ebenfalls zur Vermeidung von Wiederholungen. Der Unterschied zu *Distinct* besteht darin, dass mit *Distinctrow* unabhängig von der Auswahlliste nur ganze Datensätze verglichen werden. Mit dem *Distinctrow*-Prädikat werden Wiederholungen in Abfragen mit 1:n-Verknüpfungen unterdrückt. Das folgende Beispiel listet alle Namen und Vornamen von Kunden auf, zu denen Rechnungen vorhanden sind. Sind einem Kunden mehrere Rechnungen zugewiesen, erscheint der Name des Kunden trotzdem nur einmal in der Datensatzgruppe. *Distinctrow* verhindert die Wiederholungen.

Abfrage 1

```
SELECT DISTINCTROW Kunden.Name, Kunden.Vorname  
FROM Kunden  
INNER JOIN Rechnungen ON Kunden.KundenNr =  
Rechnungen.KundenNr ;
```

Der Ausdruck *Inner Join ... On* definiert im Beispiel die 1:n-Verknüpfung zwischen der Kundentabelle und der Rechnungstabelle mit der Kundentabelle als Master-tabelle.

Hinweis



Distinctrow hat nur dann eine Wirkung, wenn Sie mit *<Auswahlliste>* Felder nicht aus allen verwendeten Tabellen angeben. Wenn Sie im obigen Beispiel zusätzlich aus der Rechnungstabelle Felder selektieren würden, bliebe *Distinctrow* ohne Wirkung.

Top

Mit der *Select*-Anweisung können Sie über das Prädikat *Top* auch die Spitzenwerte einer Datensatzgruppe ermitteln. So lassen sich zum Beispiel die Rechnungen mit den zehn höchsten oder sieben niedrigsten Rechnungsbeträgen herausfinden. Wenn Sie beispielsweise die fünf Rechnungen mit den höchsten Rechnungsbeträgen selektieren wollen, ist das mit der folgenden Anweisung möglich:

Abfrage 2

```
SELECT TOP 5 Rechnungen.Rechnungsbetrag  
FROM Rechnungen  
ORDER BY Rechnungen.Rechnungsbetrag DESC ;
```

Beachten Sie, dass dem reservierten Wort *Top* ein vorzeichenloser Wert vom Datentyp *Integer* folgen muss. Außerdem muss eine Sortierung verwendet werden, die über die *ORDER BY*-Klausel erzeugt wird. Wird dies nicht gemacht, gibt die *Select*-Anweisung unbestimmte Datensätze zurück. Mit aufsteigender Sortierreihenfolge erhalten Sie die niedrigsten Werte, mit absteigender Sortierreihenfolge (*Desc*) die höchsten Werte.

Percent

Durch den Zusatz *Percent* lässt sich der Wert hinter *Top* in eine prozentuale Menge umwandeln. So würden im Vergleich zu obigem Beispiel nicht mehr die fünf höchsten Rechnungsbeträge, sondern die höchsten fünf Prozent ausgegeben.



```
SELECT TOP 5 Percent Rechnungen.Rechnungsbetrag
FROM Rechnungen
ORDER BY Rechnungen.Rechnungsbetrag DESC;
```

Abfrage 3

Die *Top*-Klausel entspricht der Eigenschaft *Spitzenwerte* im Eigenschaftenfenster in der Entwurfsansicht einer Abfrage.

Der Ausdruck *<Auswahlliste>* in der *Select*-Anweisung bestimmt die Datenfelder und Ausdrücke, die Access abfragen soll. *<Auswahlliste>* hat folgende Syntax:

<Auswahlliste>

```
{ * | <Auswahlunterliste 1> [[, <Auswahlunterliste 2>]
...]}
```

Der Parameter *<Auswahlunterliste>* ist dabei wie folgt aufgebaut:

```
{<Tabellenname>.* | <Ausdruck> [AS <Spalten-Alias>]}
```

Den Tabellennamen müssen Sie stets dann bei einem Datenfeld mit angeben, wenn die Abfrage auf mehrere Tabellen zugreift und Felder aus diesen Tabellen den gleichen Namen haben. Diesen Feldern wird dann ein Punkt und der Tabellenname vorangestellt. Ansonsten kann der Tabellename weggelassen werden.

Mit dem Ausdruck *<Spalten-Alias>* können Sie einem Datenfeld oder einem berechneten Ausdruck einen neuen Namen zuordnen, über den dann auf den Inhalt zugegriffen werden kann. Geben Sie kein Alias an, wird der Feldname beibehalten. Die folgende Abfrage berechnet den Brutto-Rechnungsbetrag mithilfe der Datenfelder *Rechnungsbetrag* und *MwSt*. Später kann über den Namen *Brutto-Rechnungsbetrag* auf das Ergebnis Bezug genommen werden.

```
SELECT DISTINCTROW Rechnungen.Rechnungsbetrag,
Rechnungen.MwSt,
[Rechnungsbetrag]*[MwSt]+[Rechnungsbetrag] AS [Brutto-
Rechnungsbetrag]
FROM Rechnungen;
```

Abfrage 4

Mit dem Parameter *<Tabellenausdruck>* bestimmen Sie unter anderem die Herkunftstabellen, Abfragebedingungen und Sortierreihenfolgen. Die Reihenfolge der einzelnen Klauseln muss genau eingehalten werden. Der Ausdruck *<Tabellenausdruck>* hat die folgende Syntax:

<Tabellenausdruck>

```
<FROM-Klausel> [<WHERE-Klausel>] [<GROUP-BY-Klausel>] [
<HAVING-Klausel>] [<ORDER-BY-Klausel>]
```

Herkunftstabellen festlegen

Die Herkunftstabellen für eine Abfrage bestimmen Sie mit der *From*-Klausel, die immer mit angegeben werden muss. Alle weiteren noch behandelten Klauseln sind optional. Der Aufbau ist folgendermaßen:

From-Klausel

```
FROM <Tabellenreferenz 1> [[, <Tabellenreferenz 2>]
...] [<IN-Klausel>]
```





Für <Tabellenreferenz> können Sie folgende Ausdrücke einsetzen:

```
<Tabellenname> [AS {<Korrelationsname>|<Verknüpfte  
Tabelle>}]
```

Der Parameter <Tabellenname> kann der Name einer Tabelle, einer Abfrage, einer Unterabfrage, einer eingebundenen Tabelle oder ein Korrelationsname sein. Die Angabe von Korrelationsnamen erfolgt wie auch bei Tabellen als Literal. Der Ausdruck <Verknüpfte Tabelle> lässt sich nochmals unterteilen:

```
<Tabellenname 1> {INNER|LEFT|RIGHT} JOIN  
<Tabellenname 2> ON <Referenzfelder>
```

Mit ihm werden bei Verwendung von mehreren Tabellen die relationalen Beziehungen zwischen diesen Tabellen angegeben. <Tabellenname 1> gibt die linke Seite der Verknüpfung, also die Mastertabelle, an und <Tabellenname 2> die rechte Seite, die Detailtabelle. Der Ausdruck <Referenzfelder> legt die Referenzfelder fest, über die die Verknüpfung erfolgt. Da es sich dabei in der Regel um gleichnamige Felder aus verschiedenen Tabellen handelt, müssen den Feldnamen jeweils ein Punkt und der entsprechende Tabellenname vorangestellt werden.

Verknüpfungstypen

Die Prädikate für die Verknüpfungstypen entsprechen den über das *Beziehungen*-Fenster herstellbaren Verknüpfungen *Equi Join* und *Outer Join*.

Inner Join

Mit dem Prädikat *Innner Join* erstellen Sie eine Gleichheitsverknüpfung. Bei diesem Verknüpfungstyp enthält die entstehende Datensatzgruppe nur solche Datensätze, bei denen die Referenzfelder beider Tabellen übereinstimmen. Bei der folgenden Abfrage enthält die Datensatzgruppe nur solche Kundennamen, zu denen auch Rechnungen vorhanden sind.

Abfrage 5

```
SELECT DISTINCTROW Kunden.Name, Rechnungen.RechnungsNr  
FROM Kunden INNER JOIN Rechnungen ON Kunden.KundenNr =  
Rechnungen.KundenNr ;
```

Left Join

Neben der Gleichheitsverknüpfung bietet Access zwei Formen der Inklusionsverknüpfung (*Outer Join*) an. Bei einem *Left Join* werden alle Datensätze aus der links vom Prädikat stehenden Mastertabelle in die Datensatzgruppe übernommen. Außerdem enthält die Datensatzgruppe diejenigen Datensätze der rechts vom Prädikat stehenden Detailtabelle, deren Referenzfelder mit denen der Mastertabelle übereinstimmen.

Abfrage 6

```
SELECT DISTINCTROW Kunden.Name, Rechnungen.RechnungsNr  
FROM Kunden LEFT JOIN Rechnungen ON Kunden.KundenNr =  
Rechnungen.KundenNr ;
```

Die durch die Abfrage 6 entstehende Datensatzgruppe enthält alle Datensätze aus der Tabellen *Kunden*. Zu jedem Kunden wird dann, wenn vorhanden, die zugehörige Rechnungsnummer angezeigt.



Die zweite Möglichkeit ist die Verwendung des Prädikats *Right Join*. In diesem Fall enthält die Datensatzgruppe alle Datensätze aus der Detailtabelle. Außerdem werden die Datensätze der Mastertabelle angezeigt, denen Detaildatensätze zugeordnet sind.

 *Right Join*

```
SELECT DISTINCTROW Kunden.Name, Rechnungen.RechnungsNr
FROM Kunden RIGHT JOIN Rechnungen ON Kunden.KundenNr =
Rechnungen.KundenNr;
```

 **Abfrage 7**

Die Datensatzgruppe enthält alle Datensätze aus der Tabelle *Rechnungen* und den zu jeder Rechnung gehörenden Datensatz aus der Tabelle *Kunden*. Die Namen von Kunden, denen keine Rechnung zugeordnet ist, werden nicht in die Datensatzgruppe aufgenommen.

Wenn Sie referentielle Integrität verwenden, liefert die Gleichheitsverknüpfung mit *Inner Join* dieselben Ergebnisse wie die Inklusionsverknüpfung mit *Right Join*. Das liegt daran, dass es bei referenzieller Integrität keine Detaildatensätze gibt, denen keine Masterdatensätze zugeordnet sind.



Hinweis

Zugriff auf externe Datenbanken

Die *In*-Klausel ermöglicht Ihnen, auf Datensätze von Tabellen zuzugreifen, die nicht in der aktuellen Datenbank enthalten sind. Sie können dabei nicht nur auf Access-Tabellen Bezug nehmen, sondern auch auf alle anderen Datenbanktypen, die Access handhaben kann. Darunter fallen unter anderem dBASE-, Paradox- und Fox-Pro-Tabellen. Der Zugriff erfolgt über den Namen der Tabelle, über den Pfad, in dem die Datenbank zu finden ist, und über den Datenbanktyp. Pfad und Datenbanktyp sind jeweils in einfache (") oder doppelte (") Hochkommas zu stellen und durch ein Leerzeichen voneinander getrennt.

 *In*-Klausel

```
SELECT *
FROM Kunden IN "C:\DB" "FoxPro 2.5;" ;
```

Eine zweite Schreibweise, in der das reservierte Wort *Database* verwendet wird, ist äquivalent zu der obigen *Select*-Anweisung zu benutzen.

```
SELECT *
FROM Kunden IN "" [FoxPro 2.5; DATABASE=C:\DB;];
```

Beide *Select*-Anweisungen erzeugen jeweils eine Datensatzgruppe, die alle Datenfelder und alle Datensätze aus der FoxPro-2.5-Tabelle *Kunden.dbf* anzeigt, die im Verzeichnis C:\DB gespeichert ist.

Achten Sie darauf, dass bei der Angabe des Datenbanktyps ein zusätzliches Semikolon am Ende des Typs steht. Wird auf eine Access-Datenbank zugegriffen, braucht kein Datenbanktyp angegeben zu werden. Die Pfadangabe reicht in diesem Fall aus.



Tabelle 5.13 verdeutlicht den Zugriff auf die verschiedenen Datenbanktypen.

Zugriff auf externe Datenbanktabellen

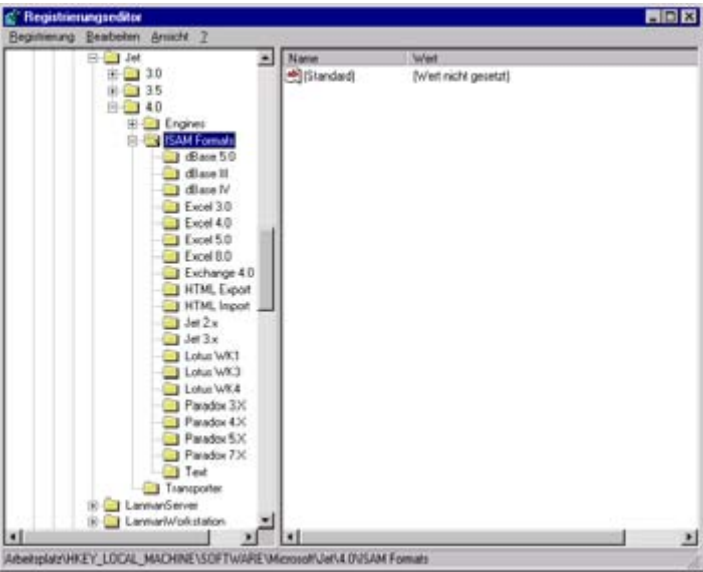
Tabellentyp	SELECT-Anweisung
Access	SELECT * FROM Kunden IN "d:\daten\lager.mdb";
dBASE III	SELECT * FROM Kunden IN "d:\daten" "dBASE III";
dBASE IV	SELECT * FROM Kunden IN "d:\daten" "dBASE IV";
FoxPro 2.0	SELECT * FROM Kunden IN "d:\daten" "FoxPro 2.0";
FoxPro 2.5	SELECT * FROM Kunden IN "d:\daten" "FoxPro 2.5";
Paradox 3.X	SELECT * FROM Kunden IN "d:\daten" "Paradox 3.X";
Paradox 4.X	SELECT * FROM Kunden IN "d:\daten" "Paradox 4.X";

Tabelle 5.13

Aufgrund der offenen ODBC-Architektur kann sich die Liste der Datenbanktypen, auf die Access zugreifen kann, ständig erweitern. Im Windows-Registrierungseditor können Sie alle auf Ihrem System verfügbaren ISAM-Formate nachsehen. Sie finden die Formate im Pfad `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Jet\3.5\ISAM Formats`.

Abb. 5.3

Im Registrierungseditor
eingetragene
ISAM-Treiber



Beachten Sie bitte, dass Sie jeweils nur auf eine externe Datenbank oder Tabelle zugreifen können. Eine externe Datenbank oder Tabelle ist dadurch gekennzeichnet, dass sie nicht in der aktuellen Datenbank enthalten ist.



Hinweis

Datensätze filtern

Wie Sie Datensätze mithilfe eines *Recordset*-Objekts filtern können, haben Sie bereits kennen gelernt. Es mussten dafür nacheinander zwei Datensatzgruppen erzeugt werden, wobei dann die letzte die gefilterten Daten enthielt. Mithilfe einer SQL-Anweisung kann das Filtern auch in einem Schritt vollzogen werden. Sie müssen mit der Methode *OpenRecordset* nur die entsprechende SQL-Anweisung angeben.

Für das Filtern von Datensätzen ist bei der *SELECT*-Anweisung die optionale *WHERE*-Klausel verantwortlich. Sie ermöglicht es, mithilfe einer Bedingung die Anzahl der anzuzeigenden Datensätze einzuschränken. Die Bedingung prüft dabei, welche Datensätze in die Datensatzgruppe aufgenommen werden und welche nicht. Die Syntax lautet wie folgt:



WHERE-Klausel

```
WHERE <Filterbedingung>
```

Die mit der *WHERE*-Klausel angegebene Bedingung entspricht der Definition von Kriterien im Abfragefenster. In *Tabelle 5.8* haben Sie schon einige Beispiele für Filterbedingungen kennen gelernt.

```
SELECT DISTINCTROW Rechnungen.RechnungsNr,
Rechnungen.Rechnungsbetrag
FROM Rechnungen
WHERE ((Rechnungen.Rechnungsbetrag < 100));
```



Abfrage 8

Die obige Abfrage nimmt alle Rechnungen in die Datensatzgruppe auf, deren Rechnungsbeträge kleiner als DM 100,- sind.

Beachten Sie bitte den Unterschied zu der ebenfalls optionalen *HAVING*-Klausel. Mit *HAVING* geben Sie eine Bedingung an, der alle Datensätze innerhalb einer Gruppierung entsprechen müssen. Die *WHERE*-Klausel selektiert die Datensätze vor der Gruppenbildung.



Hinweis

Datensätze gruppieren

Auch das Gruppieren von Datensätzen ist mit der *SELECT*-Anweisung möglich. Mit der *GROUP BY*-Klausel können Sie bis zu zehn Felder oder Ausdrücke angeben, nach denen Access die Datensatzgruppe gruppieren soll. Durch das Gruppieren werden alle Datensätze mit übereinstimmendem Gruppenkriterium zu einem Datensatz zusammengefasst.



GROUP BY-Klausel

```
GROUP BY <Ausdruck 1> [[<Ausdruck 2>] ...]
```





Alle Felder, die über die Auswahlliste der *SELECT*-Anweisung in die Datensatzgruppe aufgenommen werden, müssen in der *GROUP BY*-Klausel enthalten sein oder in einer Aggregatfunktion in der Auswahlliste vorkommen. Bei der Gruppierung werden von Access auch *NULL*-Werte berücksichtigt. Diese *NULL*-Werte bilden dann eine eigene Gruppe. In den Berechnungen der Aggregatfunktionen werden sie jedoch nicht berücksichtigt.

Auch das Selektieren von Datensätzen ist möglich. Dabei ist aber zwischen der Selektion vor und nach der Gruppenbildung zu unterscheiden. Wenn die Datensätze vor der Gruppierung selektiert werden sollen, benutzen Sie die schon oben beschriebene *WHERE*-Klausel.

Abfrage 9

```
SELECT DISTINCTROW Year ([Rechnungsdatum]) AS  
Ausdruck1, Sum(Rechnungen.Rechnungsbetrag) AS  
[Summe von Rechnungsbetrag]  
FROM Rechnungen  
WHERE ((Rechnungen.Rechnungsbetrag < 10000))  
GROUP BY Year ([Rechnungsdatum]);
```

Das Beispiel gruppiert alle Rechnungen nach der Jahreszahl des Rechnungsdatums. Die entstehende Datensatzgruppe enthält jeweils die Angabe der Jahreszahl und die Summe der Rechnungsbeträge des entsprechenden Jahres. Dabei werden für die Gruppierung nur die Datensätze berücksichtigt, bei denen der Rechnungsbetrag kleiner als DM 10.000,- ist.

Tip



Im Entwurfsbereich von Abfragen können Gruppierungen durch Einblenden der Zeile *Funktion* erzeugt werden. Dazu wählen Sie den Menüpunkt *Ansicht > Funktionen*. Über ein Kombinationsfeld kann dann für jedes Datenfeld entweder *Gruppierung* oder eine Aggregatfunktion gewählt werden.

Hinweis



Im QBE-Entwurfsbereich erzeugen Sie bei Verwendung einer Gruppierung eine *WHERE*-Klausel, indem Sie das Datenfeld für das Kriterium in eine Spalte ziehen und als Funktion *Bedingung* angeben. In der Zeile *Kriterium* definieren Sie dann die Bedingung.

Filter für eine Gruppierung definieren

HAVING-Klausel

Für die Selektion von Datensätzen innerhalb einer Gruppierung findet die *HAVING*-Klausel Anwendung. Sie selektiert die Datensätze nach einem Kriterium, nachdem die Gruppierung stattgefunden hat. Die Syntax lautet:

```
HAVING <Suchbedingung>
```

So werden zum Beispiel mit der folgenden Abfrage wieder erst alle Datensätze, bei denen der Rechnungsbetrag kleiner als DM 10.000,- ist, nach der Jahreszahl gruppiert. Anschließend werden durch die *HAVING*-Klausel nur noch die Gruppierungen angezeigt, bei denen die Summe der Rechnungsbeträge größer als DM 100,- ist.



```
SELECT DISTINCTROW Year([Rechnungsdatum]) AS  
Ausdruck1, Sum(Rechnungen.Rechnungsbetrag) AS [Summe  
von Rechnungsbetrag]  
FROM Rechnungen  
WHERE ((Rechnungen.Rechnungsbetrag < 10000))  
GROUP BY Year([Rechnungsdatum])  
HAVING (((Sum(Rechnungen.Rechnungsbetrag))>100));
```

Abfrage 10

Im QBE-Entwurfsbereich erzeugen Sie eine *HAVING*-Klausel, indem Sie bei einem Datenfeld, für das eine Gruppierung eingetragen ist, ein Kriterium definieren.



Hinweis

Datensätze sortieren

Das Sortieren von Datensatzgruppen lässt sich ebenso wie das Gruppieren mit einer *SELECT*-Anweisung leichter handhaben. Auch hier kann der Prozess in einem Schritt erledigt werden, indem der passende SQL-Text inklusive der Sortieroption bei der *OpenRecordset*-Methode angegeben wird.

Bei der *SELECT*-Anweisung übernimmt die *ORDER BY*-Klausel die Sortierung der Datensatzgruppe. Die Syntax der *ORDER BY*-Klausel lautet:

ORDER BY-Klausel

```
ORDER BY <Ausdruck 1> [{ASC|DESC}] [[, <Ausdruck 2>  
[{ASC|DESC}]] ...]
```

Sie können einen oder mehrere Ausdrücke bzw. Felder angeben. Access sortiert standardmäßig aufsteigend (A.Z, o.9). Dies entspricht der Angabe des optionalen Parameters *Asc*, der aber in diesem Fall nicht angegeben werden muss. Wenn Access absteigend sortieren soll, geben Sie den Parameter *Desc* an.

```
SELECT DISTINCTROW Year([Rechnungsdatum]) AS  
Ausdruck1, Sum(Rechnungen.Rechnungsbetrag) AS [Summe  
von Rechnungsbetrag]  
FROM Rechnungen  
WHERE ((Rechnungen.Rechnungsbetrag<10000))  
GROUP BY Year([Rechnungsdatum])  
HAVING (((Sum(Rechnungen.Rechnungsbetrag))>100))  
ORDER BY Year([Rechnungsdatum]) DESC;
```

Abfrage 11

Das Ergebnis der Abfrage entspricht dem der Abfrage 10, allerdings wird nun das Recordset zusätzlich absteigend nach Jahreszahlen sortiert.

Im QBE-Entwurfsbereich können Sie die Sortierung im Kombinationsfeld der gleichnamigen Zeile definieren. Sie haben die Wahl zwischen aufsteigender, absteigender oder keiner Sortierung.



Hinweis





Parameterabfragen

Um Abfragen flexibel zu halten und auf einfache Weise während der Laufzeit das Ergebnis zu verändern, können Parameter in Abfragen verwendet werden. Diesen Parametern muss, bevor die Abfrage ausgeführt werden kann, ein Wert zugewiesen werden. Dieser Wert ist dann das Kriterium für die Abfrage.

PARAMETERS



Zur Festlegung eines Parameters wird am Anfang der SQL-Anweisung über das Schlüsselwort *PARAMETERS* der Name und der Datentyp bestimmt. Wichtig ist, dass auf den Datentyp ein Semikolon folgt. Die Definition des Parameters wird so von der eigentlichen Abfrage getrennt. Bei Verwendung des Parameters als Kriterium in der *WHERE*-Klausel wird dieser wie ein Feld einer Tabelle gehandhabt.

Abfrage 12



```
PARAMETERS Preis Currency;
SELECT DISTINCTROW Artikel.Artikelname,
Artikel.Einzelpreis
FROM Artikel
WHERE ((Artikel.Einzelpreis > [Preis]));
```

Abfrage 12 definiert den Parameter *Preis* vom Datentyp *Währung (Currency)* und führt entsprechend dem zugewiesenen Wert die Abfrage aus. Es werden dabei alle Artikel ausgegeben, deren Preis größer ist als der Wert des Parameters.

Hinweis



Aus Parameterabfragen lässt sich nicht direkt mit einem *Recordset*-Objekt eine Datensatzgruppe bilden. Die Abfrage muss erst mit einem *QueryDef*-Objekt geöffnet oder erzeugt werden. Anschließend muss der Parameter einen Wert erhalten. Wie Sie einem Parameter in einer Abfrage per VBA einen Wert zuweisen und wie Sie herausfinden können, welche Parameter eine Abfrage besitzt, erfahren Sie im *Kapitel 9 Objektorientierte Programmierung* im Abschnitt *Abfragen designen*.

Abfragen dynamisch zusammensetzen

Noch flexibler wird eine Abfrage, wenn Sie die SQL-Anweisung während der Laufzeit aus verschiedenen Strings zusammensetzen. So kann dann wie im folgenden Beispiel nicht nur der Wert eines Parameters verändert werden, sondern auch der Vergleichsoperator, also das gesamte Kriterium. Der SQL-Text wird entsprechend der vom Anwender gemachten Eingabe konstruiert.

```
Sub AbfrageMitBenutzerdefinierterEingabe ()
```

```
Dim DB As Database
Dim R As Recordset
Dim SQL As String
Dim Ergebnis As String
```

```
Rem Eingabe des Parameters
```

```
Ergebnis = InputBox("Geben Sie bitte einen _  
Operator" & " und eine Ver-" & vbCrLf & _ _  
"gleichszeichenkette an! (Zum Beispiel >" & "H" & " " & _  
& " oder = " & "Meier" & ")")
```





```

If Ergebnis <> "" Then

    Rem Zusammenfügen des SQL-Textes
    SQL = "SELECT DISTINCTROW Kunden.Name, " _
        & "Kunden.Vorname FROM Kunden WHERE " _
        & "((Kunden.Name" & Ergebnis & "));"

    Set DB = CurrentDb
    Set R = DB.OpenRecordset(SQL)

    Rem Datensätze im Direktfenster ausgeben
    Do While R.eof = False

        Debug.Print R!Name, R!Vorname

        R.MoveNext

    Loop

End If

End Sub

```

Lautet die Eingabe beispielsweise *"Meier"*, werden alle Kunden mit dem Namen *Meier* im Direktfenster ausgegeben.

Es lassen sich aber nicht nur Strings in zusammengeführten Anweisungen verwenden, sondern auch Berechnungen, selbstgeschriebene Funktionen und Objektzugriffe.

Funktionen einbinden

Der Zugriff auf Funktionen kann auf zwei verschiedene Arten erfolgen. Dabei ist es wichtig, ob mit der Funktion auf ein Feld der entstehenden Datensatzgruppe zugegriffen wird oder auf eine Variable, die auch der Verweis auf ein Objekt, zum Beispiel ein Feld in einem Formular, sein kann.

Wird auf ein Feld der entstehenden Datensatzgruppe zugegriffen, muss der Aufruf der Funktion direkt in die Zeichenkette der SQL-Anweisung eingebettet sein, d.h. die Anweisung ist ein durchgehender String. Im folgenden Beispiel wird zu jedem Artikel aus der Datensatzgruppe mit der Funktion *Bruttopreis* der Artikelbruttopreis berechnet. Der Funktion wird dazu der Nettopreis und der Mehrwertsteuersatz jedes Artikels übergeben. Anschließend wird die Datensatzgruppe zur Kontrolle im Direktfenster ausgegeben.

```

Sub FunktionEinbinden1 ()

    Dim db As Database
    Dim R As Recordset
    Dim SQL As String

```



```

Set db = CurrentDb

SQL = "SELECT DISTINCTROW Artikel.Artikelname, _
      Artikel.Einzelpreis, _
      Bruttopreis([Einzelpreis], [MwSt]) _
      AS Bruttopreis FROM Artikel;"

Set R = db.OpenRecordset(SQL)

Rem Datensätze im Direktfenster ausgeben
Do While R.eof = False

    Debug.Print R!Artikelname, _
               Format$(R!Einzelpreis, "0.00 DM"), _
               Format$(R!Bruttopreis, "0.00 DM")

    R.MoveNext
Loop

End Sub

Function Bruttopreis(Nettopreis As Currency, _
                  MwSt As Double) As Currency

    Bruttopreis = Nettopreis * (1 + MwSt)

End Function

```

Wird der Funktion dagegen eine Variable oder ein Verweis auf ein Objekt übergeben, darf die Funktion nicht in den SQL-Text eingebettet sein, da Access die Funktion in diesem Fall als Parameter interpretiert, wodurch ein Fehler auftritt. Sie muss deshalb separat über das kaufmännische Und (&) mit der Zeichenkette verknüpft werden. Die Prozedur *FunktionEinbinden2* erzeugt eine Datensatzgruppe, die alle Artikel enthält, deren Preis größer als DM 116,- ist. Der Wert des Kriteriums wird über die Funktion *Bruttopreis* berechnet.

```

Sub FunktionEinbinden2 ()

    Dim db As Database
    Dim R As Recordset
    Dim SQL As String
    Dim Nettopreis As Currency
    Dim MwSt As Double

    Set db = CurrentDb

    Nettopreis = 100
    MwSt = .16           "> Bruttopreis = 116,00 DM

```





```
SQL = "SELECT DISTINCTROW Artikel.Artikelname,
      Artikel.Einzelpreis FROM Artikel
      WHERE ((Artikel.Einzelpreis>" _
      & Bruttopreis(Nettopreis, MwSt) & "));"

Set R = db.OpenRecordset(SQL)

Rem Datensätze im Direktfenster ausgeben
Do While R.eof = False

    Debug.Print R!Artikelname, R!Einzelpreis

    R.MoveNext
Loop

End Sub
```

Wenn Sie wie in den Beispielen eine Funktion verwenden, der mehrere Parameter übergeben werden, müssen die einzelnen Parameter durch Kommas getrennt werden. Im Gegensatz dazu muss im QBE-Entwurfsbereich das Semikolon als Trennzeichen verwendet werden. Dezimalzahlen werden nicht mit einem Punkt, sondern mit einem Komma versehen. Im QBE-Entwurfsbereich muss die Mehrwertsteuer also mit 0,16 und nicht mit 0.16 eingegeben werden.



Hinweis

Zugriff auf Objekte

Für den Verweis auf Objekte gilt dasselbe wie für das Einbinden von Funktionen, die auf Variablen zugreifen. Der Verweis muss über das &-Zeichen mit dem Rest der SQL-Anweisung verbunden werden. Wie Sie einen Verweis auf ein Formular definieren, haben Sie bereits in *Kapitel 4 Programmieren mit Objekten* gelernt.

Beim folgenden Beispiel müssen Sie, bevor Sie die Prozedur ausführen, das Formular *Artikelpreis* öffnen, einen beliebigen Preis eintragen und mit  bestätigen. Wechseln Sie dann in das Modulfenster, und führen Sie die Prozedur *Objektzugriff* durch Eingabe im Direktfenster aus. Daraufhin enthält die Datensatzgruppe alle Artikel, deren Preis größer ist als der zuvor im Formular eingetragene.

```
Sub Objektzugriff ()

    Dim db As Database
    Dim R As Recordset
    Dim SQL As String

    Set db = CurrentDb

    If forms!Artikelpreis.txt_preis <> "" Then

        SQL = "SELECT DISTINCTROW Artikel.Artikelname,
              Artikel.Einzelpreis FROM Artikel "
```





```

SQL = SQL & "WHERE ((Artikel.Einzelpreis>" & _
forms!Artikelpreis.[txt_preis] & "));"

Set R = db.OpenRecordset(SQL)

Rem Datensätze im Direktfenster ausgeben
Do While R.eof = False

    Debug.Print R!Artikelname, R!Einzelpreis

    R.MoveNext
Loop

End If

End Sub

```

Tabellenerstellungsabfragen definieren

Die **SELECT INTO**-
Anweisung

Sie kennen nun die wichtigste SQL-Anweisung – die *SELECT*-Anweisung. In einem sehr engen Zusammenhang dazu steht die *SELECT INTO*-Anweisung. Mit ihr lassen sich genau die gleichen Aktionen ausführen wie mit der *SELECT*-Anweisung. Es können Datensätze selektiert, gruppiert und sortiert werden. Der Unterschied besteht nur darin, dass das Ergebnis nicht in einer Datensatzgruppe gespeichert wird, sondern in einer neuen Tabelle. Die Syntax der *SELECT INTO*-Anweisung lautet wie folgt:

```

SELECT [ALL|DISTINCT|DISTINCTROW|TOP] <Auswahlliste>
INTO <neuer Tabellennamen> [IN-Klausel] FROM
<Datenherkunft>

```

INTO

Über das Prädikat *Into* geben Sie den Namen der Tabelle an, die erzeugt werden soll. Er wird an die Stelle des Ausdrucks *<neuer Tabellennamen>* gesetzt. Standardmäßig wird die neue Tabelle in der aktuellen Datenbank erzeugt. Über eine *IN*-Klausel lässt sich die Tabelle aber auch in einer anderen Datenbank speichern. Dabei kann das Datenbankformat frei gewählt werden. Die Syntax der *IN*-Klausel und wie Sie auf die verschiedenen Datenbankformate zugreifen, können Sie in der *Tabelle 5.9* nachschlagen.

Die neu erstellte Tabelle übernimmt die Feldnamen und die Felddatentypen der in *<Auswahlliste>* aufgeführten Felder. Andere Eigenschaften werden allerdings nicht übernommen, so dass eventuell Primärschlüssel oder Sekundärschlüssel nachträglich manuell definiert werden müssen.

Im folgenden Beispiel wird eine neue Tabelle mit dem Namen *Kunde und Rechnung* erzeugt, die alle Rechnungsnummern mit dem zugehörigen Kundennamen speichert:

Abfrage 13

```

SELECT DISTINCTROW Kunden.Name, Rechnungen.RechnungsNr
INTO [Kunde und Rechnung]

```



```
FROM Kunden INNER JOIN Rechnungen ON Kunden.KundenNr =
Rechnungen.KundenNr;
```

Anders als bei der *SELECT*-Anweisung kann bei der *SELECT INTO*-Anweisung auch eine Unterabfrage als Datenherkunft bei der *FROM*-Klausel mit angegeben werden.



Hinweis

Löschabfragen definieren

Um eine Löschabfrage zu erzeugen, stellt SQL die *DELETE*-Anweisung zur Verfügung. Mit Löschabfragen können Datensätze, die nach Kriterien selektiert werden können, aus einer vorhandenen Tabelle gelöscht werden. Die Syntax der *DELETE*-Anweisung lautet:

```
DELETE <Auswahlliste> FROM <Tabellenname> [WHERE
<Suchbedingung>]
```

Die Bedingung für die Datensätze, die gelöscht werden sollen, definieren Sie mit der Ihnen bereits bekannten *WHERE*-Klausel. Alle Datensätze, die die entstehende Datensatzgruppe enthält, werden nach dem Ausführen der Abfrage aus der bei *<Tabellenname>* angegebenen Tabelle gelöscht.

Auch in einer Löschabfrage ist die Verknüpfung von Tabellen möglich.

```
DELETE DISTINCTROW Rechnungen.*, Kunden.Name
```

```
FROM Kunden INNER JOIN Rechnungen ON Kunden.KundenNr =
Rechnungen.KundenNr
```

```
WHERE ((Kunden.Name)="Meier"));
```

Die obige Abfrage löscht alle Rechnungen zu den Kunden, die den Namen *Meier* haben. Wenn Sie alle Kunden löschen wollen, die *Meier* heißen und zu denen Rechnungen vorhanden sind, müssen Sie darauf achten, dass Sie bei der referenziellen Verknüpfung zwischen den Tabellen *Kunden* und *Rechnungen* die Löschweitergabe aktiviert haben. Ansonsten können die Kunden nicht gelöscht werden, da noch Rechnungen zu diesen Kunden vorhanden sind.

Löschabfragen löschen stets ganze Datensätze. Um die Inhalte einzelner Datenfelder zu löschen, verwenden Sie die weiter unten beschriebene Aktualisierungsabfrage.



Tipp

Das Löschen von Datensätzen ist immer endgültig. Sie sollten daher vor umfangreichen Löschoperationen unbedingt eine Sicherheitskopie der Datensätze anfertigen. Alternativ dazu können Sie das Ergebnis der Abfrage und damit die Datensätze, die gelöscht werden, auch vorher zum Beispiel in einem Formular in der Datenblattansicht anzeigen, bevor der Benutzer den Löschauftrag bestätigt oder abbricht.



Hinweis





Anfügeabfragen definieren

Die *INSERT*-Anweisung

Eine Anfügeabfrage selektiert Datensätze aus einer oder mehreren Tabellen und fügt sie an eine bereits bestehende Tabelle an. So können Sie zum Beispiel Tabellen zusammenkopieren oder Datensätze aus einer importierten Tabelle in einer Tabelle Ihrer Datenbank verwenden, indem Sie die Daten aus der externen Tabelle in die entsprechenden Felder der eigenen Tabelle einfügen. Anfügeabfragen werden unter SQL mit der *INSERT*-Anweisung erzeugt. Die Syntax lautet wie folgt:

```
INSERT INTO <Tabellenname> [<Spaltennamenliste>]
[<IN-Klausel>] {<SELECT-Anweisung>|VALUES <Werteliste>}
```

Der Ausdruck <Tabellenname> bestimmt die Zieltabelle. Es muss sich dabei um eine bereits existierende Tabelle handeln. Wenn sich die Zieltabelle in der aktuellen Datenbank befindet, genügt die Angabe des Namens. Andernfalls müssen Sie mit der *IN*-Klausel den Namen der Datei angeben, in der sich die Zieltabelle befindet.

Die Datensätze werden entsprechend der durch die *SELECT*-Anweisung entstehenden Datensatzgruppe an eine bestehende Tabelle angefügt. Das folgende Beispiel fügt die Namen der Tabelle *Kunden* an die Tabelle *Kunden-Sicherung* an. Dabei werden nur die Namen der Kunden übergeben und wieder im Feld *Name* der Tabelle *Kunden-Sicherung* eingetragen.

Abfrage 15

```
INSERT INTO [Kunden-Sicherung] ( Name )
SELECT DISTINCTROW Kunden.Name
FROM Kunden;
```

Die Inhalte von Datenfeldern lassen sich auch beliebig den Datenfeldern der Zieltabelle zuordnen. Mit der folgenden Anweisung werden an die Tabelle *Kunden-Sicherung* alle Kunden angefügt, wobei jedoch die Vornamen im Feld *Name* und die Nachnamen im Feld *Vorname* gespeichert wird.

Abfrage 16

```
INSERT INTO [Kunden-Sicherung] ( Vorname, Name )
SELECT DISTINCTROW Kunden.Name, Kunden.Vorname
FROM Kunden;
```

Mit der *SELECT INTO*-Anweisung ist es auch möglich, den Feldern einer Tabelle direkt Werte zuzuordnen. Die folgende Anweisung fügt an die Tabelle *Kunden-Sicherung* einen Datensatz mit den Feldern *Name* und *Vorname* an, die mit den entsprechenden Inhalten gefüllt werden.

Abfrage 17

```
INSERT INTO [Kunden-Sicherung] ( Vorname, Name )
VALUES ( "Hans", "Wurst" );
```

Wenn Sie diese Abfrage in der SQL-Ansicht des Abfragefensters entworfen haben und die Abfrage schließen oder in die Entwurfsansicht schalten, wandelt Access die *Values*-Klausel automatisch in eine *SELECT*-Anweisung um. Das Ergebnis der Abfrage ändert sich dadurch aber nicht. Die entsprechende Abfrage sieht wie folgt aus:

```
INSERT INTO [Kunden-Sicherung] ( Vorname, Name )
SELECT "Hans", "Wurst";
```



Beachten Sie, dass Access beim Anfügen von Datensätzen an eine bereits bestehende Tabelle die gleichen Regeln prüft wie bei der Dateneingabe. So verhindert Access etwaige Schlüsselverletzungen ebenso wie mögliche Inkompatibilitäten in den Datentypen.



Hinweis

Die mit der *INSERT INTO*-Anweisung angegebene Zieltabelle kann im QBE-Entwurfsbereich im Eigenschaftenfenster bei der Eigenschaft *Zieltabelle* eingetragen werden.



Hinweis

Aktualisierungsabfragen definieren

Für umfangreiche Änderungen in Tabellen, die einzelne Datenfelder betreffen, sind Aktualisierungsabfragen am besten geeignet. Mit solchen Abfragen können für viele Datensätze in einem Schritt die gleichen Änderungen vorgenommen werden. So lässt sich zum Beispiel der Mehrwertsteuersatz mit einer Abfrage in allen Datensätzen von 15% auf 16% ändern. Wenn Sie nur einzelne oder sehr verschiedene Veränderungen vornehmen wollen, führen Sie diese besser manuell durch. Aktualisierungsabfragen definieren Sie mit der *UPDATE*-Anweisung:

```
UPDATE <Tabellenname> SET <SET-Klausel-Liste> [WHERE
<Suchbedingung>]
```

Über den Ausdruck *<Tabellenname>* bestimmen Sie die Tabelle, deren Daten geändert werden sollen. Die *WHERE*-Klausel schränkt wie bei der *SELECT*-Anweisung die Datensätze der Datensatzgruppe, die erzeugt wird, ein. Die Syntax der *SET*-Klausel lautet wie folgt:

```
SET <Spaltenname 1> = {<Ausdruck 1>|Null}
[[<Spaltenname 2> = {<Ausdruck 2>|Null}] ...]
```

Mit der *Set*-Klausel werden die Felder der Tabelle bestimmt, die aktualisiert werden sollen. Jedem Feld wird ein Ausdruck zugeordnet, dessen Wert in das Feld eingetragen werden soll.

```
UPDATE DISTINCTROW Rechnungen
SET Rechnungen.Status =
"Bezahlt", Rechnungen.MwSt = 0.16
WHERE ((Rechnungen.Rechnungsbetrag<10));
```

Das Beispiel ermittelt zunächst alle Rechnungen, bei denen der Rechnungsbetrag kleiner als DM 10,- ist. Anschließend wird der Status auf *Bezahlt* und der Mehrwertsteuersatz auf 16% gesetzt.

Die *Update*-Anweisung

Set

Abfrage 18

Kreuztabellenabfragen definieren

Kreuztabellen eignen sich hervorragend für zwei oder mehrdimensionale Datenbestände. Der Vorteil einer Kreuztabelle besteht darin, dass sowohl für die Zeilen als auch für die Spalten der entstehenden Datensatzgruppe Gruppenkriterien definiert werden können. Die in einer Zelle der Kreuztabelle enthaltenen Daten sind

Die *TRANSFORM*-Anweisung





dadurch gekennzeichnet, dass die zugrunde liegenden Datensätze beide Gruppenkriterien erfüllen. Für die Zellen einer Kreuztabelle werden in der Regel Aggregatfunktionen definiert. Per SQL wird die Definition einer Kreuztabellenabfrage mit der *TRANSFORM*-Anweisung eingeleitet:

```
TRANSFORM <Aggregatfunktion> <SELECT-Anweisung> PIVOT
<Spaltenname> [IN <Fixwertliste>]
```

Auf *TRANSFORM* folgt die Angabe einer Aggregatfunktion, die die Berechnung für die Zellen der Kreuztabelle enthält. Die *SELECT*-Anweisung bestimmt wie gewohnt die Herkunftstabellen und die Datenfelder, nach denen gruppiert werden soll. Zusätzlich können auch noch die von der *SELECT*-Anweisung bekannten Klauseln verwendet werden, um die Datensätze beispielsweise vorher zu selektieren. Die über die *SELECT*-Anweisung erzeugten Gruppierungen bilden die Zeilen der entstehenden Kreuztabelle. Es muss deshalb immer eine *GROUP BY*-Klausel im Zusammenhang mit der *SELECT*-Anweisung auftauchen.

PIVOT

Auf den Parameter *PIVOT* darf nur der Name eines Datenfeldes folgen. Dieses Datenfeld bildet das Kriterium für die Spalten der Kreuztabelle. Das folgende Beispiel bildet eine Kreuztabelle, in der die einzelnen Kundennummern die Zeilen darstellen und alle vorkommenden Jahre aus den Rechnungsdaten die Spalten. Die Zellen der Kreuztabelle enthalten die Summe der Rechnungsbeträge der entsprechenden Kombinationen aus Kundennummer und Jahreszahl.

Abfrage 19

```
TRANSFORM Sum(Rechnungen.Rechnungsbetrag) AS [Summe
von Rechnungsbetrag]
SELECT Rechnungen.KundenNr
FROM Rechnungen
GROUP BY Rechnungen.KundenNr
PIVOT Year([Rechnungsdatum]);
```

IN

Mit der *IN*-Klausel, die in diesem Fall nicht für den Bezug zu einer externen Datenbank benutzt wird, lassen sich fixe Spaltenüberschriften erzeugen. So erhalten die mit einer *PIVOT*-Anweisung nach Quartalen gruppierten Spalten vernünftige Spaltennamen.

```
TRANSFORM ...
...
PIVOT "Quartal " & DatePart("q",[Rechnungsdatum])
IN ("1.Quartal","2.Quartal","3.Quartal","4.Quartal");
```

Übersicht der Abfragetypen

Mit den SQL-Anweisungen, die Sie in den vorangegangenen Abschnitten kennen gelernt haben, halten Sie ein nützliches Werkzeug für das Erstellen von Abfragen per VBA in der Hand, mit dem sich viele Probleme einfach bewältigen lassen. Damit Sie die Anweisungen auch leicht anwenden können, sind die einzelnen Abfragetypen und ihre Syntax in *Tabelle 5.14* zusammengestellt. So haben Sie immer einen schnellen Überblick über die Struktur einer Anweisung.



Anweisungen von Access-SQL

Anweisung	Syntax	Abfragetyp
DELETE	DELETE <Auswahlliste> FROM <Tabellenname> [WHERE <Suchbedingung>]	Löschabfrage
INSERT	INSERT INTO <Tabellenname> [<Spaltennamenliste>] [<IN-Klausel>] {<SELECT-Anweisung> VALUES <Werteliste>}	Anfügeabfrage
SELECT	SELECT [ALL DISTINCT DISTINCTROW [n PERCENT]]] <Auswahlliste> FROM <Tabellenausdruck>	Auswahlabfrage
SELECT-INTO	SELECT [ALL DISTINCT DISTINCTROW TOP] <Auswahlliste> INTO <neuer Tabellenname> [<IN-Klausel>] FROM <Datenherkunft>	Tabellen- erstellungs- abfrage
TRANSFORM	TRANSFORM <Aggregatfunktion> <SELECT-Anweisung> PIVOT <Spaltenname> [IN <Fixwertliste>]	Kreuztabellen- abfrage
UPDATE	UPDATE <Tabellenname> SET <SET-Klausel-Liste> [WHERE <Suchbedingung>]	Aktualisierungs- abfrage

Tabelle 5.14

Das Datumsformat

Besondere Beachtung verdient das Datumsformat von Access-SQL. In allen Abfragen oder SQL-Texten muss das Datum im amerikanischen Format eingegeben werden. Dabei gilt die Reihenfolge Monat – Tag – Jahr. Die einzelnen Werte werden durch Schrägstriche (/) voneinander getrennt und das gesamte Datum in Doppelkreuze (#) gestellt.

Mit einer kleinen gebrauchsfertigen Funktion können Sie ein übergebenes Datum in das SQL-Format umwandeln. Weisen Sie dazu in Ihrem Programm einer Variablen vom Datentyp *String* die Funktion mit einem Datum als Argument zu. Die Variable enthält anschließend das Datum im richtigen Format. Wird kein Datum übergeben, liefert die Funktion einen Leerstring.

```
Function DatumUmwandeln (Datumstring As  
Variant) As String
```

```
    Dim Monat As String  
    Dim Tag As String  
    Dim Jahr As String
```





```
If Datumstring <> "" Then
    Tag = Format$(Datumstring, "dd")
    Monat = Format$(Datumstring, "mm")
    Jahr = Format$(Datumstring, "yy")
    DatumUmwandeln = "#" & Monat & "/" & Tag & "/" & _
        & Jahr & "#"
Else
    DatumUmwandeln = ""
End If
```

End Function

Als Beispiel für die Verwendung der Funktion in einem SQL-Text soll die Prozedur *KriteriumDatum* dienen:

```
Sub KriteriumDatum()

    Dim DB As Database
    Dim R As Recordset
    Dim SQL As String
    Dim NeuesDatum As String

    NeuesDatum = DatumUmwandeln(Forms![Datum _
        Umwandeln].txt_datum)

    Rem SQL-Anweisung definieren
    SQL = "SELECT DISTINCTROW RechnungsNr, _
        Rechnungsdatum FROM Rechnungen "
    SQL = SQL & "WHERE ((Rechnungsdatum<" & _
        NeuesDatum & ")) _
        ORDER BY Rechnungsdatum;"

    Set DB = CurrentDb

    Set R = DB.OpenRecordset(SQL)

    Rem Datensätze im Direktfenster ausgeben
    Do While R.EOF = False

        Debug.Print R!RechnungsNr, R!Rechnungsdatum
        R.MoveNext

    Loop

    R.Close
    DB.Close

End Sub
```





Formulare programmieren



Formulare öffnen	308
Schließen von Formularen	315
Suchen nach Datensätzen	318
Navigieren in Formularen	327
Neueingabe von Datensätzen	332
Löschen von Datensätzen	334
Kontrolle der Dateneingabe	343
Filtern und Sortieren	344
Dialogfenster	353
Positionierung, Größenänderung und Aktivierung von Formularen	361
Aktivierung und Deaktivierung von Formularen	365
Zeitgeberprogrammierung	367
Abfangen von Formularfehlern	368





Aufgaben von Formularen

Die Entwicklung einer Datenbank beginnt normalerweise mit der Anlage der Tabellenstruktur, in der die einzelnen Datensätze der Datenbank abgelegt werden. Zur Eingabe und Präsentation der Daten ist die Datenblattansicht von Tabellen nicht gerade sehr komfortabel. Aus diesem Grund stellt Access Formulare zur Verfügung, deren Aufgabe die ansprechende Präsentation der Daten ist.

Die Aufgabe von Formularen geht jedoch noch weiter. So bilden Formulare die Hauptschnittstelle zwischen einer Datenbankanwendung und den Daten. Dies bedeutet, dass Formulare nicht nur der Eingabe und Präsentation von Daten, sondern auch der Vereinfachung der Bedienung dienen. Hinter einem Formular steht daher nicht selten ein umfangreicher Programmcode, der die Manipulation der Daten komfortabel macht.

Für den Entwickler sind Formulare ebenfalls die zentralen Objekte der Anwendungsentwicklung. Fast alle in der Praxis benötigten Ereignisse gehen nämlich von Formularen und den darin enthaltenen Steuerelementen aus. Da Access ereignisorientiert programmiert wird, spielt dies natürlich eine wichtige Rolle.

In diesem Kapitel lernen Sie den Umgang und die Steuerung von Formularen mithilfe von VBA-Code kennen. Das Thema ist vor allem der praktische Umgang mit den Ereignissen, die von Formularen ausgelöst werden können. Als Datenzugriffsmodell wird DAO verwendet.

Alle Beispiele dieses Kapitels finden Sie in der Datenbank *Kapitel6.mdb*.

Formulare öffnen

Eine der wohl am häufigsten benötigten Aktionen ist das Öffnen von Formularen. Access stellt zu diesem Zweck das Datenbankfenster zur Verfügung, mit dem sich jedes Formular durch einen einfachen Doppelklick auf den Namen öffnen lässt.

Die gleiche Wirkung können Sie auch direkt von VBA aus erzielen. Im *Kapitel 2 Schnellstart VBA* haben Sie zu diesem Zweck bereits die *OpenForm*-Methode kennen gelernt, deren Aufgabe im Öffnen eines Formulars besteht. Zur Erinnerung folgt noch einmal die Syntax dieser Methode:

```
DoCmd.OpenForm <FormName> [,<View>] [,<FilterName>]  
[,<WhereCondition>] [,<DataMode>] [,<WindowMode>]  
[,<OpenArgs>]
```

In der einfachsten Ausprägung muss nur ein einziges Argument – nämlich der Name des zu öffnenden Formulars – angegeben werden. Das folgende Beispiel öffnet das Formular mit dem Namen *Kunden*:

```
DoCmd.OpenForm FormName:="Kunden"
```



Eine Anweisung dieser Art wird häufig im Zusammenhang mit Schaltflächen eingesetzt, bei denen ein Klick zum Öffnen eines bestimmten Formulars führt. Sie haben dies bereits im zweiten Kapitel kennen gelernt. Mittels des Code-Generators haben Sie eine Ereignisprozedur angelegt, mit deren Hilfe Sie auf das *Klick*-Ereignis (engl. *Click*-Ereignis) reagieren. Innerhalb der Ereignisprozedur haben Sie dann zum Öffnen des Formulars die *OpenForm*-Methode eingesetzt.

Das *Klick*-Ereignis

Klickt der Anwender nun auf die Schaltfläche, löst Access das *Click*-Ereignis aus und startet die Ereignisprozedur. Hier wird wiederum die *OpenForm*-Methode ausgeführt und schließlich das gewünschte Formular geöffnet.

Die Beispieldatenbank *Kapitel6.mdb* enthält ein Formular mit dem Namen *Hauptmenü*. Hier befindet sich eine Schaltfläche mit der Beschriftung *Kundenformular öffnen*, deren *Beim Klicken*-Ereignis mit einer entsprechenden Ereignisprozedur verknüpft ist. Die Prozedur hat den folgenden Aufbau:

```
Private Sub btnKunden_Click()  
  
    DoCmd.OpenForm FormName:="Kunden"  
  
End Sub
```

Ein Klick auf die Schaltfläche führt nun dazu, dass Access das Kundenformular öffnet.

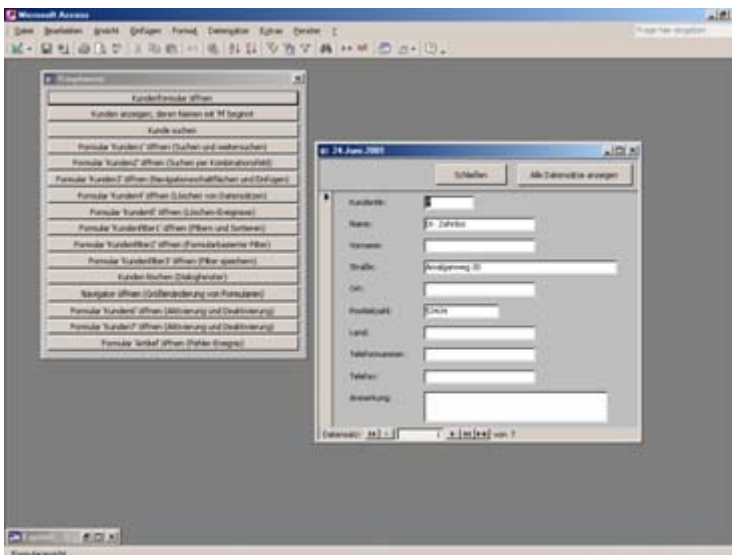


Abb. 6.1

Das Formular *Kunden* in der Formularansicht

Wenn Sie die Prozedur begutachten möchten, öffnen Sie das Formular *Hauptmenü* in der Entwurfsansicht. Aktivieren Sie anschließend das Eigenschaftenfenster der Schaltfläche und positionieren Sie die Einfügemarke in das Eingabefeld *Beim Klicken* der Schaltfläche. Rechts neben dem Eingabefeld erscheint nun die *Editor*-Schaltfläche.



Hinweis



che, mit der Sie die Ereignisprozedur aktivieren. Alternativ können Sie auch den Menüpunkt *Ereignis* des Kontextmenüs der Schaltfläche auswählen. Da es sich bei dem *Click*-Ereignis um das Standardereignis von Schaltflächen handelt, führt dies zum selben Ergebnis.

Ansicht und Datenmodus festlegen

Formularansicht

Neben dem Formularnamen können der *OpenForm*-Methode noch einige weitere Argumente übergeben werden. Besonders wichtig sind in diesem Zusammenhang die beiden Argumente *<View>* und *<Datamode>*. Unter *<View>* legen Sie beispielsweise fest, ob Access das Formular in der Formular- oder der Datenblattansicht öffnet. Die folgende Tabelle enthält eine Übersicht über alle Konstanten, die Sie für *<View>* angeben können:

Angaben für den Parameter <View>

Konstante	Beschreibung
<i>acNormal</i>	Öffnet das Formular in der Formularansicht. Diese Einstellung ist die Standardeinstellung, falls Sie das optionale Argument <i><View></i> nicht angeben.
<i>acDesign</i>	Öffnet das Formular in der Entwurfsansicht
<i>acPreview</i>	Öffnet das Formular in der Seitenansicht. Die Seitenansicht zeigt die Vorschau des Formulars, wie es beim Ausdruck auf einem Drucker ausgegeben wird.
<i>acFormDS</i>	Öffnet das Formular in der Datenblattansicht. Die einzelnen Datensätze werden hier in Form einer Tabelle angezeigt.

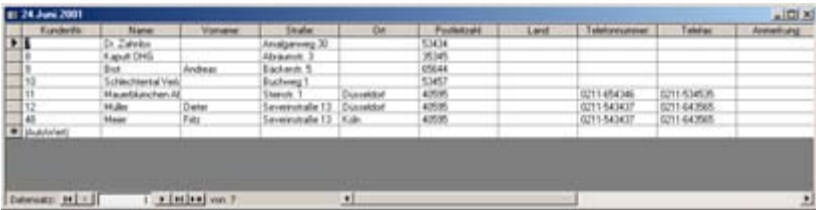
Tabelle 6.1

Um das Formular *Kunden* beispielsweise in der Datenblattansicht zu öffnen, ergänzen Sie die *OpenForm*-Methode einfach um die Konstante *acFormDS*:

```
DoCmd.OpenForm FormName:="Kunden", View:=acFormDS
```

Abb. 6.2

Das Formular *Kunden* in der Datenblattansicht



Der zweite wichtige Parameter mit dem Namen *<DataMode>* bestimmt, welche Möglichkeiten der Benutzer bei der Bearbeitung der Daten innerhalb des Formulars hat. Wenn Sie das Formular beispielsweise im *Nur Lesen*-Modus öffnen, kann der Anwender weder vorhandene Datensätze bearbeiten noch neue Datensätze hinzufügen. Die folgende Tabelle enthält alle möglichen Einstellungen:

Datenmodus

Angaben für das Argument *DataMode*

Konstante	Beschreibung
<i>acFormAdd</i>	Es können nur neue Datensätze hinzugefügt werden. Das Betrachten oder Bearbeiten alter Datensätze ist nicht möglich. Der Anwender hat allerdings die Möglichkeit, seine neu eingegebenen Datensätze zu bearbeiten.
<i>acFormEdit</i>	Die Datensätze können wie gewohnt bearbeitet werden. Auch das Hinzufügen neuer Datensätze ist möglich.
<i>acFormPropertySettings</i>	Diese Einstellung ist die Standardeinstellung, wenn das Argument <i><DataMode></i> ausgelassen wird. Es werden die Einstellungen der Formulareigenschaften <i>AllowEdits</i> , <i>AllowDeletions</i> , <i>AllowAdditions</i> und <i>DataEntry</i> verwendet.
<i>acFormReadOnly</i>	Die Daten des Formulars können nur gelesen werden. Eine Bearbeitung ist nicht möglich.

Tabelle 6.2

Wenn Sie bei der *OpenForm*-Anweisung eine der Konstanten *acFormAdd*, *acFormEdit* oder *acFormReadOnly* verwenden, werden die Formulareigenschaften *AllowEdits*, *AllowDeletions*, *AllowAdditions* und *DataEntry* für diese eine Aktion überschrieben und somit nicht berücksichtigt.

Das folgende Beispiel öffnet das Formular *Kunden* wieder in der Datenblattansicht. Außerdem wird der Datenmodus "Nur-Lesen" (*acReadOnly*) aktiviert:

```
DoCmd.OpenForm FormName:="Kunden", View:=acFormDS,  
DataMode:= acFormReadOnly
```

Filter und Bedingungen

Zwei weitere wichtige Argumente sind die beiden Parameter *<FilterName>* und *<WhereCondition>*. Wie die Namen dieser Parameter schon vermuten lassen, ermöglichen sie das Filtern der Datenquelle. Dies bedeutet, dass Sie innerhalb des Formulars nur bestimmte Datensätze auswählen können.





Unter *<FilterName>* kann der Name einer Abfrage als Zeichenkette angegeben werden. Über diese Abfrage können nun die Datensätze mithilfe geeigneter Bedingungen eingeschränkt – also gefiltert werden. Hierbei muss darauf geachtet werden, dass die Abfrage alle Datenfelder enthält, die von dem Formular benötigt werden.

Sortieren von Datensätzen



Auch das Sortieren von Datensätzen ist mithilfe des *FilterName*-Arguments möglich. Geben Sie hierzu einfach den Namen einer Abfrage an, die die Datensätze in der gewünschten Reihenfolge sortiert.

Bedingung



Wenn Sie auf das Sortieren der Datensätze verzichten möchten, können Sie alternativ auch auf das Argument *<WhereCondition>* zurückgreifen. Hier geben Sie einfach den *WHERE*-Teil einer SQL-Abfrage als Zeichenkette an. Access zeigt in dem Formular dann die Datensätze an, die der Bedingung entsprechen. Das Anlegen einer Abfrage ist in diesem Fall also nicht extra erforderlich.

Das folgende Beispiel öffnet wieder das Formular *Kunden*. Diesmal werden allerdings nur Kunden angezeigt, deren Namen mit dem Buchstaben *M* beginnen:

```
DoCmd.OpenForm FormName:="Kunden", _  
WhereCondition:="Name LIKE 'M*'"
```

In dem Formular *Hauptmenü* der Beispieldatenbank finden Sie die Schaltfläche *Kunden anzeigen, deren Namen mit "M" beginnen*. Die *Click*-Ereignisprozedur dieser Schaltfläche enthält die obige Anweisung.

Aufheben eines Filters oder einer Bedingung



Ein Filter oder eine Bedingung, die Sie auf die beschriebene Art und Weise angewendet haben, lässt sich rückgängig machen, indem Sie das Formular schließen und anschließend – diesmal ohne Angabe einer Bedingung – wieder öffnen.

Eine andere Möglichkeit besteht darin, dass Sie die Methode *ShowAllRecords* des *DoCmd*-Objekts ausführen. Diese Methode erwartet keine Parameter und hebt alle Filter und Bedingungen des momentan aktiven Formulars auf. Sie ist damit identisch mit der *AnzeigenAlleDatensätze*-Aktion.

Eine Besonderheit der Methode ist, dass sie sich immer auf das momentan aktive Formular bezieht. Um also sicherzustellen, dass die Methode auf das richtige Formular angewendet wird, muss dieses vorher aktiviert werden. Dies bewerkstelligen Sie mithilfe der *SetFocus*-Methode, die von allen *Form*-Objekten (also alle Formulare) zur Verfügung gestellt wird. Das Aufheben des Filters bzw. der Bedingung könnte folgendermaßen aussehen:

```
Forms!Kunden.SetFocus  
DoCmd.ShowAllRecords
```

Alle Datensätze anzeigen



Der ideale Einsatzort für diese beiden Anweisungen ist eine Schaltfläche, die Sie direkt im Kundenformular unterbringen. Geben Sie der Schaltfläche die Beschriftung *Alle Datensätze anzeigen*. Anschließend legen Sie für das *Click*-Ereignis eine Ereignisprozedur an. Die Angabe der *SetFocus*-Methode ist hierbei allerdings nicht



erforderlich, da das Formular *Kunden* automatisch den Fokus erhält, wenn der Benutzer die Schaltfläche *Alle Datensätze anzeigen* anklickt. Die komplette Ereignisprozedur hat damit den folgenden Aufbau:

```
Private Sub btn_AlleAnzeigen_Click ()

    DoCmd.ShowAllRecords

End Sub
```

Wenn Sie nun das Formular mit einer Bedingung oder einem Filter öffnen, genügt ein Klick auf diese Schaltfläche, um wieder alle Datensätze anzeigen zu lassen. Sie finden diese Schaltfläche im Formular *Kunden* der Beispieldatenbank.

Kontrollieren des Öffnens von Formularen

Sobald ein Formular geöffnet wird – sei es durch die *OpenForm*-Methode oder die Schaltfläche *Öffnen* des Datenbankfensters – löst Access für das betreffende Formular das *Open*-Ereignis aus, auf das Sie mithilfe der *BeimÖffnen*-Ereigniseigenschaft reagieren. Sie finden diese Ereigniseigenschaft im Eigenschaftenfenster aller Formulare.

Wenn Sie mithilfe des Code-Generators eine Ereignisprozedur für dieses Ereignis anlegen, erstellt dieser im Formularmodul den folgenden Prozedurrumpf:

```
Private Sub Form_Open (Cancel As Integer)

End Sub
```

Beim Start der Ereignisprozedur übergibt Access dieser einen Parameter mit dem Namen *Cancel*, den Sie bereits im *Kapitel 3 Die Programmiersprache VBA* kennen gelernt haben. Da der Parameter per *Call by Reference* übergeben wird, können Sie den Inhalt der Variablen während der Ausführung der Prozedur verändern und Access damit eine Information zukommen lassen.

 **Cancel-Parameter**

Im Zusammenhang mit dem *Open*-Ereignis hat der *Cancel*-Parameter die Aufgabe, das Öffnen eines Formulars bei Bedarf zu verhindern. Zu diesem Zweck muss der Variablen *Cancel* lediglich der logische Wert *True* zugewiesen werden. Wenn Sie den Inhalt der Variablen dagegen nicht verändern, behält sie ihren Standardwert *False* und das Formular kann wie gewohnt geöffnet werden.

Wenn das Öffnen des Formulars mithilfe der *Cancel*-Variable verhindert wird, spricht man auch vom *Abbruch des Open-Ereignisses*.

 **Hinweis**

In der Praxis wird das *Öffnen*-Ereignis oft dazu verwendet, bestimmte Initialisierungsaufgaben zu erledigen, die das Formular betreffen. Dies kann beispielsweise das Füllen eines Datenfeldes oder das Aus- und Einblenden bestimmter Steuerelemente sein. Die folgende Ereignisprozedur blendet beispielsweise das aktuelle Datum in die Titelzeile des Formulars ein:



```
Private Sub Form_Open (Cancel As Integer)

    Me.Caption = Format(Date, "dd.mmmm.yyyy")

End Sub
```

Formularbeschriftung ändern

In dem obigen Beispiel wird der Ausdruck *Me.Caption* dazu verwendet, um auf die *Caption*-Eigenschaft (deutsch: *Beschriftung*-Eigenschaft) des aktuellen Formulars Bezug zu nehmen und dieser das formatierte Tagesdatum zuzuweisen. Anstelle des Ausdrucks *Me* hätte auch der vollständige Verweis auf das Formular *Kunden* angegeben werden können:

```
Forms!Kunden.Caption
```

Der Einsatz von *Me* ist allerdings vorteilhafter, da der Name des Formulars nicht im Programmcode angegeben werden muss. Wenn Sie das Formular später also umbenennen möchten, brauchen Sie den Programmcode nicht anzupassen.

Hinweis



Beachten Sie, dass die *Me*-Eigenschaft immer nur in Formularen (bzw. Berichten) angewendet werden kann und hierbei stets auf das Formular verweist, in dessen Formularmodul der Ausdruck *Me* eingesetzt wird! Die beiden Ausdrücke

```
Me.Caption
```

und

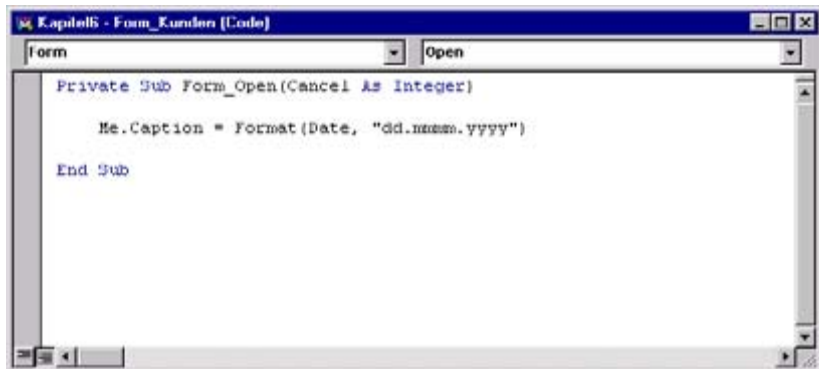
```
Forms!Kunden.Caption
```

sind also nur dann gleichwertig, wenn Sie den ersten Ausdruck im Modul des Formulars *Kunden* einsetzen!

Sie finden die Ereignisprozedur im Formularmodul des Formulars *Kunden*.

Abb. 6.3

Die Ereignisprozedur des Formulars *Kunden*





Das *Load*-Ereignis

Direkt nach dem *Open*-Ereignis löst Access beim Öffnen eines Formulars ein zweites Ereignis mit dem Namen *Load* aus. Auf dieses Ereignis können Sie mithilfe der *BeimLaden*-Ereigniseigenschaft reagieren.

Im Prinzip dient das *Load*-Ereignis dem gleichen Zweck wie das *Open*-Ereignis. Zum Zeitpunkt des Auftretens des *Load*-Ereignisses ist das Formular allerdings bereits mit dem ersten Datensatz gefüllt. Ein weiterer Unterschied besteht darin, dass das *Load*-Ereignis nicht abgebrochen werden kann.

Wenn Sie mithilfe des *New*-Schlüsselworts mehrere Instanzen eines Formulars öffnen, löst Access zunächst nur das *Open*-Ereignis aus. Erst wenn Sie das Formular sichtbar machen, wird das *Load*-Ereignis ausgelöst.



Hinweis

Schließen von Formularen

Zum Schließen eines Formulars stellt Access eine Methode mit dem Namen *Close* zur Verfügung. Die Methode hat folgende Syntax:

```
DoCmd.Close [<ObjectType>, <ObjectName>] [, <Save>]
```

Da die *Close*-Methode nicht nur auf Formulare spezialisiert ist, sondern sich z.B. auch Berichte oder Tabellen damit schließen lassen, muss unter *<ObjectType>* zunächst einmal der Typ des Objekts angegeben werden. Die Angabe erfolgt über eine der folgenden Konstanten:



Objekttypen

Konstanten des Arguments *<ObjectType>*



Konstante	Beschreibung
<i>acDefault</i> (Standardwert)	Aktives Fenster
<i>acTable</i>	Tabelle
<i>acQuery</i>	Abfrage
<i>acForm</i>	Formular
<i>acReport</i>	Bericht
<i>acMacro</i>	Makro
<i>acModule</i>	Modul
<i>acDataAccessPage</i>	Datenzugriffsseite



Konstanten des Arguments <ObjectType> (Forts.)

Konstante	Beschreibung
acDiagram	Diagramm
acServerView	Serveransicht
acStoredProcedure	Gespeicherte Prozedur

Tabelle 6.3

Als zweiter Parameter muss schließlich der Name des zu schließenden Objekts als Zeichenkette angegeben werden. Die folgende Anweisung schließt beispielsweise das Formular mit dem Namen *Kunden*:

```
DoCmd.Close ObjectType:=acForm, ObjectName:="Kunden"
```

Der dritte Parameter <Save> spielt in der Praxis nur bei der Programmierung von Assistenten eine Rolle. Hiermit bestimmen Sie, ob die letzten Änderungen am Objekt gespeichert werden sollen oder nicht.

Wenn Sie die Angabe für den Objekttyp und den Objektnamen auslassen, verwendet Access die Konstante *acDefault* und schließt automatisch das momentan aktive Formular.

Ein beliebter Einsatzort der *Close*-Methode ist eine Schaltfläche mit der Aufschrift *Schließen*, die sich auf einem Formular befindet. Durch einen Klick auf diese Schaltfläche lässt sich das Formular komfortabel schließen. Eine solche Schaltfläche finden Sie im Formular *Kunden*. Die *Click*-Ereignisprozedur hat folgenden Aufbau:

```
Private Sub btn_Schließen_Click()

    DoCmd.Close ObjectType:=acForm, ObjectName:=Me.Name

End Sub
```

Eine Besonderheit dieser Prozedur ist der Ausdruck *Me.Name*. Über diesen Ausdruck wird auf die *Name*-Eigenschaft des aktuellen Formulars zugegriffen. Da der Ausdruck im Modul des Formulars *Kunden* eingesetzt wird, hat er die Zeichenkette *Kunden* zum Ergebnis. Der Ausdruck ist damit gleichwertig mit dem folgenden Ausdruck:

```
DoCmd.Close ObjectType:=acForm, ObjectName:="Kunden"
```





Kontrollieren des Schließens von Formularen

Genau wie beim Öffnen eines Formulars die beiden Ereignisse *Open* und *Load* ausgelöst werden, treten beim Schließen eines Formulars die beiden Ereignisse *Unload* und *Close* auf. Hierbei ist es unerheblich, ob das Formular durch einen Klick auf das X-Symbol in der rechten oberen Ecke des Fensters, den Menüpunkt *Datei > Schließen* oder die Methode *Close* geschlossen wurde.

Die beiden Ereignisprozeduren dieser Ereignisse haben den folgenden Aufbau:

```
Private Sub Form_Unload (Cancel As Integer)
```

```
End Sub
```

```
Private Sub Form_Close ()
```

```
End Sub
```

Die Ereignisprozedur des *Unload*-Ereignisses besitzt wieder den *Cancel*-Parameter, mit dessen Hilfe Sie das *Unload*-Ereignis abbrechen und damit das Schließen des Formulars verhindern können.

Im Zusammenhang mit dem Ereignis *Unload* gibt es einen Trick, wie Sie das Beenden der aktuellen Benutzerdatenbank verhindern können. Sobald Sie das Beenden einleiten (z.B. über den Menüpunkt *Datei > Beenden*) schließt Access selbstständig alle geöffneten Formulare. Trifft Access hierbei auf ein Formular, das sich nicht schließen lässt, bricht Access den Vorgang ab. Die *Unload*-Ereignisprozedur des Formulars könnte den folgenden Aufbau besitzen:

```
Private Sub Form_Unload (Cancel As Integer)
```

```
    Cancel = True
```

```
End Sub
```

Der Einsatz dieser Ereignisprozedur ist beispielsweise im Zusammenhang mit einem *Hauptmenü*-Formular sinnvoll, das Sie direkt beim Start der Datenbank mithilfe des *AUTOEXEC*-Makros öffnen.

Wenn das Formular nun einmal geöffnet ist, ergibt sich allerdings das Problem, dass das Hauptmenü überhaupt nicht mehr geschlossen werden kann. Dementsprechend kann auch die Benutzerdatenbank nicht mehr beendet werden. Dies lässt sich z.B. mithilfe einer globalen (bzw. öffentlichen) Variablen lösen, die den Namen *p_SchließenErlaubt* trägt. Die Variable deklarieren Sie im Deklarationsteil eines Standardmoduls mithilfe der folgenden Anweisung:

```
Public p_SchließenErlaubt As Boolean
```

Unload- und *Close*-
Ereignisprozeduren

Schließen der
aktuellen Daten-
bank verhindern





Wenn das Schließen des Formulars erlaubt sein soll, weisen Sie der globalen Variablen einfach den Wert *True* zu. Soll das Schließen nicht zulässig sein, weisen Sie ihr den Wert *False* zu.

Innerhalb der Ereignisprozedur muss nun nur noch das Abbrechen des Schließvorgangs vom Inhalt der Variablen abhängig gemacht werden:

```
Private Sub Form_Unload (Cancel As Integer)

    If p_SchließenErlaubt = False Then Cancel = True

End Sub
```

Suchen nach Datensätzen

Im Zusammenhang mit der *OpenForm*-Methode haben Sie bereits erfahren, wie Sie mithilfe einer Bedingung die Datensätze eines Formulars filtern können. Die Angabe dieser Bedingung eignet sich auch dazu, innerhalb eines Formulars eine Suche nach einem bestimmten Datensatz durchzuführen.

Die folgende Prozedur zeigt mithilfe der *InputBox*-Funktion ein Eingabefenster an, in das der Anwender einen Namen oder auch den Teil eines Namens eingeben kann. Anschließend wird das Formular *Kunden* mit einer passenden Bedingung geöffnet. In dem Formular werden also nur die Datensätze angezeigt, die der Bedingung entsprechen.

Formular mit Bedingung öffnen

```
Private Sub btnKundeSuchen_Click()

    Dim Suchtext As Variant

    Suchtext = InputBox("Bitte geben Sie den _  
Suchtext ein!")

    If Suchtext <> "" Then

        "Der Vorgang wurde nicht abgebrochen"
        DoCmd.OpenForm FormName:="Kunden", _
            WhereCondition:="Name LIKE "*" _  
            & Suchtext & "*"

    End If

End Sub
```



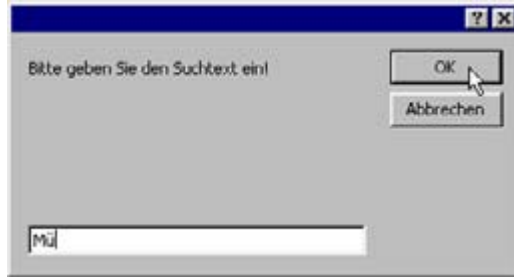


Abb. 6.4

Das Fenster der
InputBox-Funktion

Zunächst zeigt die *InputBox*-Funktion das Eingabefenster auf dem Bildschirm an. Das vom Anwender eingegebene Ergebnis wird in einer Variablen mit dem Namen *Suchtext* gespeichert. Sollte der Anwender im Eingabefenster auf *Abbrechen* geklickt haben, gibt die *InputBox*-Funktion eine leere Zeichenkette zurück. Diese wird mithilfe der *If*-Anweisung abgefangen. Nur dann, wenn der Anwender einen Suchtext eingibt und auf *OK* klickt, führt Access die Anweisung zum Öffnen des Formulars aus.

Mithilfe des Suchtextes wird schließlich über den Ausdruck

```
"Name LIKE "*" & Suchtext & "*"
```

eine Bedingung erstellt. Gibt der Anwender beispielsweise *Müller* ein, hat der obige Ausdruck folgende Zeichenkette zum Ergebnis:

```
"Name LIKE "*Müller*"
```

Es werden also alle Datensätze angezeigt, deren Namen die Buchstabenfolge *Müller* enthalten – demnach auch z. B. Rohmüller, Dr. Müller oder Müller-Bender.

Sie finden die Prozedur in dem Modul des Formulars *Hauptmenü*. Die Prozedur ist an das *Click*-Ereignis der Schaltfläche *Kunde suchen* gebunden.

Eine Besonderheit tritt dann auf, wenn Sie auf die Schaltfläche *Kunde suchen* klicken, obwohl das Formular *Kunden* bereits geöffnet ist. Wenn Sie nun einen neuen Suchtext eingeben und mit *OK* bestätigen, führt Access die *OpenForm*-Methode aus, obwohl das Formular bereits geöffnet ist. In diesem Fall kommt es nicht zu einem Fehler und Access arbeitet die Methode problemlos ab.

Der große Vorteil besteht darin, dass Access trotzdem die neue Bedingung berücksichtigt und die alte Bedingung dementsprechend ersetzt. Im Kundenformular werden nun also nur die zur neuen Bedingung passenden Kunden angezeigt.



Positionieren auf Datensätze

Eine Einschränkung bei der Suche nach Datensätzen mithilfe der *OpenForm*-Methode besteht darin, dass Access die Datenquelle des Formulars filtert. Wenn Sie z. B. als Suchtext *Müller* eingeben und Access nur einen passenden Datensatz findet, können Sie über die Navigationsschaltflächen des Formulars nicht mehr zu anderen Datensätzen wechseln, ohne die Bedingung vorher aufzuheben.

FindRecord-
Methode

In der Praxis ist es oft erwünscht, nur auf einen bestimmten Datensatz zu positionieren, ohne hierbei die Datenquelle – also die Datensätze des Formulars *Kunden* – zu filtern. Zu diesem Zweck stellt Access die Methode *FindRecord* zur Verfügung, die folgende Syntax hat:

```
DoCmd.FindRecord <FindWhat> [, <Match>] [, <MatchCase>]  
[, <Search>] [, <SearchAsFormatted>]  
[, <OnlyCurrentField>] [, <FindFirst>]
```

Die Aufgabe dieser Methode ist, innerhalb eines Steuerelements (vorzugsweise innerhalb eines Textfeldes) nach einem bestimmten Inhalt (<FindWhat>) zu suchen. Wenn die Methode fündig wird, positioniert sie auf den passenden Datensatz. Es folgt ein Überblick über die einzelnen Parameter der Methode:

Die Parameter der Methode *FindRecord*

Parameter	Beschreibung
<FindWhat>	Der Text, nach dem gesucht werden soll.
<Match>	Über dieses Argument bestimmen Sie, ob Access bei der Suche nur einen Teil des Feldinhalts oder den kompletten Feldinhalt vergleicht. Es können die folgenden drei Konstanten angegeben werden: <i>acAnywhere</i> (Teil des Feldinhalts), <i>acEntire</i> (gesamter Feldinhalt), <i>acStart</i> (Anfang des Feldinhalts). Wenn Sie dieses Argument auslassen, verwendet Access die Standardvorgabe <i>acEntire</i> .
<MatchCase>	Geben Sie hier <i>True</i> an, damit Access die Groß- und Kleinschreibung beim Vergleich berücksichtigt. Wenn Sie <i>False</i> angeben oder das Argument auslassen, beachtet Access die Schreibweise nicht.
<Search>	Legt fest, in welche Richtung die Suche vom aktuellen Datensatz aus erfolgen soll. Die Konstante <i>acDown</i> bewirkt eine Suche in Richtung Tabellenende, die Konstante <i>acUp</i> dagegen in Richtung Tabellenanfang. Wenn Access alle Datensätze durchsuchen soll, geben Sie <i>acSearchAll</i> an.



Die Parameter der Methode *FindRecord* (Forts.)

Parameter	Beschreibung
<code><SearchAsFormatted></code>	Mithilfe der <i>Format</i> -Eigenschaft können die Inhalte von Steuerelementen formatiert dargestellt werden. Die Darstellung stimmt damit nicht mehr mit den ursprünglichen Daten der zugrunde liegenden Datenquelle überein. Wenn Sie für dieses Argument <i>True</i> angeben, durchsucht Access den formatierten Inhalt. Wenn Sie das Argument dagegen auslassen oder <i>False</i> angeben, berücksichtigt Access den originalen Inhalt der Datenquelle.
<code><OnlyCurrentField></code>	Dieses Argument bestimmt, ob die Methode alle Steuerelemente des Formulars (Konstante <i>acAll</i>) oder lediglich das aktuelle Steuerelement durchsucht, das den Fokus besitzt (Einstellung <i>acCurrent</i>). Die Standardeinstellung ist <i>acCurrent</i> .
<code><FindFirst></code>	Geben Sie hier <i>True</i> an, damit Access die Suche mit dem ersten Datensatz beginnt. Wenn Sie hier <i>False</i> angeben, durchsucht Access die Datensätze vom aktuellen Datensatz aus. Die Suchrichtung bestimmt hierbei das Argument <code><Search></code> . Die Standardeinstellung ist <i>True</i> .

Tabelle 6.4

Wenn unter `<OnlyCurrentField>` die Konstante *acCurrent* angegeben wird, durchsucht die Methode immer das momentan aktive Steuerelement. Dies ist beispielsweise bei einer gezielten Suche nach einem Namen innerhalb des Kundenformulars sinnvoll. Damit die Suche in dem richtigen Steuerelement erfolgt, müssen Sie es vorher aktivieren. Dies erledigen Sie über die *SetFocus*-Methode des gewünschten Steuerelements.

Eine weitere Voraussetzung ist, dass das Formular, in dem die Suche erfolgen soll, ebenfalls aktiviert ist. Das folgende Beispiel aktiviert das Formular *Kunden* und anschließend das Steuerelement *Name*. Die *FindRecord*-Methode durchsucht dieses Feld schließlich nach dem Text *Müller*:

```
Forms!Kunden.SetFocus
Forms!Kunden!Name.SetFocus
```

```
DoCmd.FindRecord FindWhat:="Müller", _
    Match:=acAnywhere, _
    MatchCase:=False, Search:=acDown, _
    SearchAsFormatted:=False, _
    OnlyCurrentField:=acCurrent, FindFirst:=True
```

Steuerelement für
die Suche aktivieren





Sollte Access fündig werden, wird der erste passende Datensatz zum aktuellen Datensatz kenntlich gemacht. Sollte dagegen kein passender Datensatz vorhanden sein, bricht Access die Suche ohne Fehlermeldung ab. Der aktuelle Datensatz bleibt hierbei erhalten.

Suchen nach weiteren Datensätzen

In dem obigen Beispiel erfolgt die Suche immer vom ersten Datensatz des Formulars aus in Richtung Tabellenende. Bewirkt wird dies durch die entsprechenden Einstellungen der Argumente *<Search>* und *<FindFirst>*. Sollten in der Datenquelle mehrere Kunden mit dem Namen *Müller* enthalten sein, führt die Suche mittels der *FindRecord*-Anweisungen immer zum gleichen Ergebnis.

Suche fortsetzen

Um dieses Problem zu lösen, könnten Sie die Parameter der *FindRecord*-Methode etwas modifizieren, indem Sie bei einer erneuten Suche vom aktuellen Datensatz aus suchen. Das letzte Argument *<FindFirst>* besitzt in diesem Fall den Wert *False*:

```
DoCmd.FindRecord FindWhat:="Müller", _
    Match:=acAnywhere, _
    MatchCase:=False, Search:=acDown, _
    SearchAsFormatted:=False, _
    OnlyCurrentField:=acCurrent, FindFirst:=False
```

Access stellt jedoch in Form der *FindNext*-Methode noch eine etwas komfortablere Möglichkeit zur Verfügung. Die *FindNext*-Methode wiederholt die letzte Suche automatisch vom aktuellen Datensatz aus. Hierbei werden alle bei der letzten Suche über die *FindRecord*-Methode festgelegten Einstellungen übernommen. Dies ist auch der Grund dafür, dass die *FindNext*-Methode keine Parameter besitzt.

Das Duo *FindRecord* und *FindNext* lässt sich hervorragend innerhalb eines Formulars einsetzen, um – durch den Anwender gesteuert – nach einem bestimmten Feldinhalt zu suchen und nacheinander alle passenden Datensätze auszuwählen. In der Beispieldatenbank zu diesem Kapitel finden Sie ein entsprechendes Kundenformular mit dem Namen *Kunden1*, das zwei Schaltflächen mit den Beschriftungen *Suche starten* und *Weitersuchen* enthält.

Suchtext eingeben

Das Formular besitzt eine weitere Verbesserung: Der Suchtext wird nicht fest im Programmcode vorgegeben, sondern kann vom Anwender eingegeben werden. Zur Eingabe dieses Suchtextes würde sich beispielsweise wieder die *InputBox*-Funktion eignen. In diesem Fall befindet sich auf dem Formular jedoch ein spezielles Textfeld, das den Namen *txtSuchtext* trägt. In dieses Textfeld kann der Anwender den Suchtext eingeben. Da das Steuerelement an kein Datenfeld der zugrunde liegenden Datenquelle gebunden ist, werden solche Steuerelemente auch als *ungebundene Steuerelemente* bezeichnet.



Abb. 6.5

Das Formular *Kunden*

Die *BeimKlicken*-Ereignisprozedur der Schaltfläche *Suche starten* hat den folgenden Aufbau:

Suche starten

```
Private Sub btnSucheStarten_Click()

Dim Suchtext As Variant

Suchtext = Me!txtSuchtext

If IsNull(Suchtext) = True Then

    Beep
    MsgBox "Es ist kein Suchtext vorhanden!"

Else

    Me!Name.SetFocus

    DoCmd.FindRecord FindWhat:=Suchtext, _
        Match:=acAnywhere, _
        MatchCase:=False, Search:=acUp, _
        SearchAsFormatted:=False, _
        OnlyCurrentField:=acCurrent, _
        FindFirst:=True

End If

End Sub
```





Innerhalb der Ereignisprozedur wird zunächst einmal eine Variable mit dem Namen *Suchtext* deklariert, der in der zweiten Zeile der vom Benutzer eingegebene Suchtext zugewiesen wird. Der Zugriff auf das Textfeld *txtSuchtext* erfolgt hierbei wieder mithilfe der *Me*-Eigenschaft. Der vollständige Verweis wäre allerdings ebenfalls möglich

```
Forms!Kunden1!txtSuchtext
```

Nachdem der Inhalt aus dem Steuerelement in die Variable *Suchtext* eingelesen wurde, erfolgt mithilfe einer *If*-Abfrage eine Überprüfung, ob überhaupt ein Suchtext vorhanden ist. Wenn das Textfeld leer ist, hat es den Inhalt *NULL* und die *IsNull*-Funktion liefert das Ergebnis *True*. Sollte dies der Fall sein, wird der Anwender über ein Meldungsfenster auf den fehlenden Text aufmerksam gemacht.

Wenn dagegen ein Suchtext eingegeben wurde, erhält das Steuerelement *Name* den Fokus und die Suche wird gestartet. Sie erfolgt vom ersten Datensatz aus in Richtung Tabellenende.

Hinweis



Dem Formular *Kunden1* muss in diesem Beispiel nicht extra der Fokus zugewiesen werden, da sich die Schaltfläche *Suche starten* auf dem Formular *Kunden1* befindet. Klickt der Anwender die Schaltfläche an, erhält das Formular automatisch den Fokus. Dies geschieht, noch bevor Access die Ereignisprozedur der Schaltfläche startet.

Weitersuchen



Die Ereignisprozedur der zweiten Schaltfläche *Weitersuchen* besitzt folgenden Aufbau:

```
Private Sub btnWeitersuchen_Click()  
  
    Me!Name.SetFocus  
    DoCmd.FindNext  
  
End Sub
```

Damit die erneute Suche im richtigen Steuerelement erfolgt, muss vor der Ausführung der *FindNext*-Methode ebenfalls wieder der Fokus auf das Steuerelement *Name* platziert werden.

Test des Formulars



Das Formular können Sie nun einem Test unterziehen. Dazu öffnen Sie es in der Formularansicht und geben in dem Textfeld *txtSuchtext* einen Namen oder auch nur den Teil eines Namens ein (z.B. *Meier*). Mit einem Klick auf *Suche starten* wird der erste passende Datensatz angezeigt. Auf den nächsten Kunden mit dem Namen *Meier* positionieren Sie nun mittels der Schaltfläche *Weitersuchen*.

Hinweis



Beim Verweis auf das Steuerelement *txtSuchtext* wurde immer die *Me*-Eigenschaft verwendet:

```
Suchtext = Me!txtSuchtext
```

Wenn Sie innerhalb eines Formularmoduls auf ein Steuerelement zugreifen möchten, das sich auf dem dazugehörigen Formular befindet, kann die *Me*-Eigenschaft



auch ausgelassen werden. Die Zuweisung des Feldinhalts an die Variable *Suchtext* hätte also auch folgendermaßen erfolgen können:

```
Suchtext = txtSuchtext
```

Diese Kurzschreibweise ist allerdings etwas unübersichtlich, da auf den ersten Blick nicht klar ist, ob es sich bei *txtSuchtext* um ein Steuerelement oder etwa eine Variable handelt. Aus diesem Grund werden Steuerelementverweise in diesem Kapitel immer inklusive *Me* angegeben.

Die Prozedur *FormularDatensatzAuswählen*

Die Suche nach einem bestimmten Datensatz mithilfe der *FindRecord*-Methode hat einige Nachteile. Zum einen können Sie nicht in Erfahrung bringen, ob überhaupt ein Datensatz gefunden wurde. Das größere Problem ist allerdings, dass Sie nur Felder durchsuchen können, für die Sie ein Steuerelement angelegt haben.

Dieses Problem lässt sich mithilfe der Prozedur *FormularDatensatzAuswählen* lösen, deren Aufbau und Funktionsweise Sie bereits im *Kapitel 5 Datenzugriff mit VBA* kennen gelernt haben. Sie finden diese Prozedur unter anderem auch in der Beispieldatenbank zu diesem Kapitel. Die Funktion wird mithilfe der folgenden Syntax aufgerufen:

```
<Erfolg> = FormularDatensatzAuswählen (<Form-Objekt>, _  
<Bedingung>)
```

Der erste Parameter ist ein Verweis auf das Formular, in dem Sie den Datensatz suchen möchten. Die *<Bedingung>* bestimmt den Datensatz, auf den Sie positionieren möchten. Zu diesem Zweck geben Sie hier den *WHERE*-Teil einer SQL-Abfrage an. Das Ergebnis der Funktion ist *True*, falls ein passender Datensatz ausfindig gemacht werden konnte. Andernfalls gibt die Funktion *False* zurück.

Die folgende Prozedur positioniert auf den Kunden mit der *KundenNr* 8:

```
Private Sub KundeAuswählen ()  
  
Dim ok As Boolean  
  
    ok = FormularDatensatzAuswählen (Forms!Kunden, _  
        "KundenNr = 8")  
  
    If ok = False Then  
  
        Beep  
        MsgBox "Der Kunde konnte nicht gefunden werden!"  
  
    End If  
  
End Sub
```



Auswahl über Kombinationsfelder

Ein Nachteil der Prozedur *FormularDatensatzAuswählen* besteht darin, dass sich immer nur auf den ersten passenden Datensatz positionieren lässt. Der Einsatzbereich liegt daher in der Suche nach eindeutigen Schlüsseln wie beispielsweise der *KundenNr*.

Ein praktisches Beispiel finden Sie in dem Formular *Kunden2*. Im Kopf des Formulars befindet sich ein Kombinationsfeld, das den Steuerelementnamen *comSuchKundenNr* trägt. Mittels einer SQL-Abfrage werden in der Liste des Kombinationsfeldes alle Kundennamen mit den entsprechenden Kundennummern aufgelistet. Die gebundene Spalte ist die zweite Spalte, so dass ein Zugriff auf das Kombinationsfeld die *KundenNr* des hier ausgewählten Kunden zum Ergebnis hat.

Abb. 6.6

Das Kombinationsfeld
des Formulars *Kunden2*



Sobald nun ein Kunde aus diesem Kombinationsfeld ausgewählt wird, soll Access auf den passenden Datensatz positionieren. Eine Neupositionierung muss also immer nach der Auswahl eines neuen Kunden erfolgen. Wie fast alle Steuerelemente stellen Kombinationsfelder zu diesem Zweck die *BeimKlicken*-Eigenschaft zur Verfügung. Bei Kombinationsfeldern löst Access das *Click*-Ereignis immer dann aus, wenn ein neuer Listeneintrag ausgewählt wurde. Um nun auf dieses Ereignis mit der Positionierung auf den betreffenden Datensatz zu reagieren, ist die folgende Ereignisprozedur geeignet:

Auf einen ausge-
wählten Datensatz
positionieren

```
Private Sub comSuchKundenNr_Click()
```

```
Dim ok As Boolean
```





```
ok = FormularDatensatzAuswählen(Me, _
    "KundenNr = " & Me!comSuchKundenNr)
```

End Sub

Die Bedingung setzt sich aus der Zeichenkette *KundenNr =* zusammen, an die die Kundennummer angehängt wird. Wenn Sie beispielsweise aus dem Kombinationsfeld den Kunden *Kaputt OHG* auswählen, führt der Verweis *Me!comSuchKundenNr* zu der Kundennummer 8. Die Bedingung besitzt dementsprechend den Aufbau

KundenNr = 8

Mit dieser Information kann die Prozedur *FormularDatensatzAuswählen* schließlich die Positionierung erledigen.

Ob die Positionierung erfolgreich war oder nicht, teilt die *Function*-Prozedur über ihren Rückgabewert mit (*True* oder *False*). In dem obigen Beispiel ist diese Information allerdings nicht wichtig. Sie wird einfach der Variablen *OK* zugewiesen, deren Inhalt schließlich nicht ausgewertet wird. In solchen Fällen bietet es sich an, die *Function*-Prozedur im Stil einer *Sub*-Prozedur aufzurufen. Hierbei wird das Ergebnis einfach ignoriert. Weiterhin dürfen die Parameter nicht in Klammern angegeben werden. Die Ereignisprozedur lässt sich damit folgendermaßen vereinfachen:

```
Private Sub comSuchKundenNr_Click()

    FormularDatensatzAuswählen Me, "KundenNr = " & _
        Me!comSuchKundenNr
```

End Sub

Navigieren in Formularen

Zum Positionieren auf einen bestimmten Datensatz stellt Access dem Anwender in der unteren linken Ecke von Formularen die so genannten *Navigationsschaltflächen* zur Verfügung. Diese Schaltflächen erlauben den Wechsel zum ersten, letzten, vorhergehenden oder nachfolgenden Datensatz.

Die Navigation mittels der Navigationsschaltflächen ist allerdings etwas umständlich, da sie in Größe und Position nicht optimal sind. Aus diesem Grund können die Navigationsschaltflächen mithilfe der Formulareigenschaft *Navigationsschaltflächen* deaktiviert und dann schließlich durch eigene Schaltflächen ersetzt werden. Diese können Sie im Formular beliebig anordnen und beschriften.

Die Funktion der einzelnen Schaltflächen wird mithilfe der Methode *GoToRecord* implementiert. Die Methode hat folgende Syntax:

```
DoCmd GoToRecord [<ObjectType>, <ObjectName>]
[, <Record>] [, <Offset>]
```



**Navigations-
schaltflächen**



**GoToRecord-
Methode**



Objekttypen

Die ersten beiden Parameter *<ObjectType>* und *<ObjectName>* legen fest, innerhalb welchen Objekts Sie den Datensatz wechseln möchten. Unter *<ObjectType>* können Sie zu diesem Zweck folgende Angaben machen:

Mögliche Angaben für den Parameter *<ObjectType>*

Konstante	Beschreibung
<i>acActiveDataObject</i>	Wechselt den Datensatz innerhalb des aktiven Objekts. Dies ist die Standardeinstellung.
<i>acDataTable</i>	Ermöglicht den Datensatzwechsel innerhalb einer Tabelle, die in der Datenblattansicht geöffnet ist.
<i>acDataQuery</i>	Ermöglicht den Datensatzwechsel innerhalb einer Abfrage, deren Ergebnis in der Datenblattansicht angezeigt wird.
<i>acDataForm</i>	Wechselt den Datensatz innerhalb eines Formulars.

Tabelle 6.5

Datensatz

Der Name des betreffenden Objekts muss schließlich unter *<ObjectName>* als Zeichenkette angegeben werden. Über den dritten Parameter *<Record>* bestimmen Sie, zu welchem Datensatz gewechselt werden soll. Auch hier ist wieder die Angabe einer Konstanten erforderlich:

Angaben für den Parameter *<Record>*

Konstante	Beschreibung
<i>acPrevious</i>	Wechselt zum vorherigen Datensatz.
<i>acNext</i> (Standardwert)	Wechselt zum nächsten Datensatz.
<i>acFirst</i>	Wechselt zum ersten Datensatz.
<i>acLast</i>	Wechselt zum letzten Datensatz.
<i>acNewRec</i>	Wechselt zum neuen, leeren Datensatz. Der Anwender kann hier direkt mit der Eingabe neuer Daten beginnen.
<i>acGoto</i>	Überspringt eine bestimmte Anzahl von Datensätzen. Die Anzahl wird über das Argument <i><Offset></i> festgelegt. Positive Werte bewirken einen Sprung in Richtung Tabellenende; negative Werte in Richtung Tabellenanfang.

Tabelle 6.6



Den Parameter *<Offset>* können Sie nicht nur bei der Verwendung der Konstanten *acGoto* verwenden, sondern auch zusammen mit *acNext* und *acPrevious*. In diesen Fällen gibt der Parameter *<Offset>* die Anzahl der Datensätze an, die vorwärts bzw. rückwärts gesprungen werden sollen.



Hinweis

Um in dem Formular *Kunden* beispielsweise zum nächsten Datensatz zu wechseln, geben Sie einfach die folgende Anweisung an:

```
DoCmd.GoToRecord acForm, "Kunden", acNext
```

Natürlich ist die Verwendung benannter Parameter ebenfalls zulässig:

```
DoCmd.GoToRecord ObjectType:=acForm, _  
ObjectName:="Kunden", Record:=acNext
```

Ein Problem tritt auf, wenn in dem obigen Beispiel bereits der letzte Datensatz aktiv ist. Durch die Ausführung der *GoToRecord*-Methode wird in diesem Fall schließlich der neue, leere Datensatz aktiv. Wird die Methode ein weiteres Mal ausgeführt, meldet Access den Laufzeitfehler "Sie können nicht zu dem angegebenen Datensatz springen". Wenn Sie die Methode also zur Navigation in einem Formular einsetzen, müssen Sie diesen Fehler mithilfe der *On Error*-Anweisung abfangen bzw. ignorieren. Dazu geben Sie einfach vor der *GoToRecord*-Methode die Anweisung *On Error Resume Next* an:

```
On Error Resume Next  
DoCmd.GoToRecord acForm, "Kunden", acNext
```



Fehlerbehandlung

Abb. 6.7

Die Navigationsschaltflächen des Formulars *Kunden*





In dem Formular *Kunden3* der Beispieldatenbank finden Sie fünf Schaltflächen, über die Sie den aktuellen Datensatz wechseln können. Das folgende Beispiel zeigt die Click-Ereignisprozedur der Schaltfläche *btnNächster*:

```
Private Sub btnNächster_Click()  
  
    On Error Resume Next  
    DoCmd.GoToRecord acForm, Me.Name, acNext  
  
End Sub
```

Kontrolle der Datensatznavigation

Wenn Sie in Ihren Formularen nicht auf die Standard-Navigationssymbole von Access verzichten, aber trotzdem über einen Datensatzwechsel informiert werden möchten, können Sie dies mithilfe der *BeimAnzeigen*-Eigenschaft erledigen. Diese Eigenschaft korrespondiert mit dem *Current*-Ereignis (Deutsch: *Anzeigen-Ereignis*), das Access bei jedem Datensatzwechsel auslöst.

Hinweis



Da Access beim Öffnen eines Formulars zum ersten Datensatz wechselt, wird auch hierbei das *Current*-Ereignis ausgelöst. Das Ereignis tritt allerdings erst dann auf, wenn die beiden Ereignisse *Open* und *Load* bereits abgearbeitet wurden.

Die *Current*-Ereignisprozedur von Formularen hat den folgenden Aufbau:

```
Private Sub Form_Current ()  
  
End Sub
```

Formulare synchronisieren

Ein wichtiger Einsatzbereich des *Current*-Ereignisses ist die Synchronisation zweier Formulare. Dies können beispielsweise die beiden Formulare *Kunden* und *Rechnungen* sein. Beiden Formularen liegen zwei Tabellen zugrunde, zwischen denen eine 1:n-Beziehung existiert. Jeder Rechnung ist damit ein Kunde zugeordnet. Sie finden die beiden Formulare und die entsprechenden Tabellen in der Beispieldatenbank zu diesem Kapitel.

Current- Ereignis
des Rechnungs-
formulars



Wenn im Formular *Rechnungen* der Datensatz gewechselt wird, soll Access im Formular *Kunden* immer den zur Rechnung passenden Kundendatensatz anzeigen. Die komplette Aufgabe lässt sich mithilfe einer einzigen Ereignisprozedur lösen, die mit dem *Current*-Ereignis des Formulars *Rechnungen* verknüpft ist. Die Prozedur hat den folgenden Aufbau:

```
Private Sub Form_Current2()  
  
    If IsNull(Me!KundenNr) Then Exit Sub  
  
    If IstGeöffnet("Kunden") = True Then
```



```
Forms!Kunden.SetFocus
Forms!Kunden!KundenNr.SetFocus

DoCmd.FindRecord Me!KundenNr, acEntire, False, _
    acDown, False, acCurrent, True

End If

End Sub
```

Die erste *If*-Abfrage untersucht zunächst einmal, ob in dem Feld *KundenNr* des Formulars *Rechnungen* überhaupt ein Kunde ausgewählt wurde. Wenn dies nicht der Fall ist und das Feld damit den Inhalt *NULL* aufweist, wird die Prozedur mithilfe der *Exit Sub*-Anweisung verlassen.

Die nächste *If*-Abfrage überprüft anschließend mithilfe der Funktion *IstGeöffnet*, ob das Formular *Kunden* geöffnet ist. Bei der Funktion *IstGeöffnet* handelt es sich um eine nicht fest in Access eingebaute Funktion, deren Aufbau und Funktionsweise Sie bereits im *Kapitel 4 Programmieren mit Objekten* kennen gelernt haben. Die Funktion befindet sich im Modul *Tools* der Beispieldatenbank. Wenn das angegebene Formular geöffnet ist, gibt die Funktion den Wert *True* zurück. Andernfalls ist das Ergebnis *False*.

Sollte das Kundenformular geöffnet sein, erfolgt innerhalb der *If*-Abfrage die Positionierung auf den richtigen Kunden. Dazu wird zunächst der Fokus auf das Formular *Kunden* verschoben. Nachdem anschließend das Feld *KundenNr* aktiviert wurde, erfolgt die Suche mithilfe der *FindRecord*-Methode.

Sollte das Formular *Kunden* dagegen nicht geöffnet sein, erfolgt keine Synchronisation.

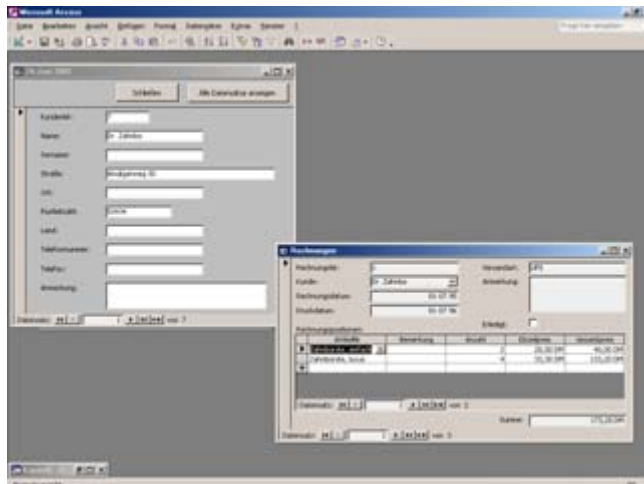


Abb. 6.8

Die beiden synchronisierten Formulare *Rechnungen* und *Kunden*





Anstelle mit der Methode *FindRecord* auf den passenden Datensatz zu positionieren, könnten Sie auch die bereits vorgestellte Prozedur *FormularDatensatzAuswählen* einsetzen. In diesem Fall hat die Prozedur folgenden Aufbau:

```
Private Sub Form_Current()  
  
    If IsNull(Me!KundenNr) Then Exit Sub  
  
    If IstGeöffnet("Kunden") = True Then  
  
        FormularDatensatzAuswählen Forms!Kunden, _  
            "KundenNr = " & Me!KundenNr  
  
    End If  
  
End Sub
```

Neueingabe von Datensätzen

Zur Eingabe eines neuen Datensatzes in ein Formular muss der Anwender lediglich zum letzten Datensatz wechseln. Hier erwartet ihn ein leerer Datensatz, der mit neuen Daten gefüllt werden kann. Wie Sie bereits im vorhergehenden Abschnitt *Navigieren in Formularen* erfahren haben, können Sie auf diesen Datensatz auch mithilfe einer Anweisung positionieren:

Zum neuen
Datensatz

```
DoCmd.GoToRecord acForm, "Kunden", acNewRec
```

Der Wechsel zu einem neuen, leeren Datensatz bedeutet aber noch nicht, dass der Benutzer den neuen Kundendatensatz auch wirklich anlegt. Der Anwender hat bei diesem Datensatz lediglich die *Möglichkeit* zur Anlage eines neuen Datensatzes.

Access legt erst dann einen neuen Datensatz in der zugrunde liegenden Tabelle an, wenn das erste Zeichen eingegeben wurde. Am Beispiel des Formulars *Kunden* können Sie dies leicht erkennen: Wechseln Sie einmal zum neuen Datensatz und begutachten Sie den Inhalt des Feldes *KundenNr*. In diesem Feld befindet sich der Eintrag (*AutoWert*). Dieser Eintrag kennzeichnet, dass Access bei der Anlage eines neuen Datensatzes in dieses Feld selbstständig einen eindeutigen Schlüsselwert einträgt.

Hinweis



Anhand eines *AutoWert*-Feldes lässt sich gut ermitteln, ob im Formular momentan ein "normaler" Datensatz oder ein neuer Datensatz angezeigt wird. Im letzteren Fall hat das *AutoWert*-Feld den Inhalt *NULL*. Da dieser Inhalt bei *AutoWert*-Feldern sonst nicht vorkommen kann, ist dies ein eindeutiges Indiz.

Physikalische
Anlage eines
Datensatzes

Klicken Sie nun das Feld *Name* an, um den Fokus auf dieses Steuerelement zu verschieben. Sobald Sie hier das erste Zeichen des Namens eintippen, erscheint in dem Feld *KundenNr* eine neue Kundennummer. Außerdem verwandelt sich das Symbol des Datensatzmarkierers in einen symbolischen Stift. Erst jetzt legt Access den neuen Datensatz an.



Für den Entwickler einer Datenbank ist es oft wichtig zu wissen, wann ein Benutzer einen neuen Datensatz anlegt. Zu diesem Zweck löst Access bei der Anlage eines neuen Datensatzes zwei Ereignisse aus: das *VorEingabe-* (*BeforeInsert*) und das *NachEingabe-*Ereignis (*AfterInsert*).

Das VorEingabe-Ereignis

Das *VorEingabe-*Ereignis tritt auf, wenn der Anwender gerade im Begriff ist, den neuen Datensatz anzulegen. Zu diesem Zeitpunkt hat Access den Datensatz also noch nicht physikalisch angelegt. Dieses Ereignis eignet sich gut dazu, das Anlegen eines neuen Datensatzes abubrechen. Wenn Sie mithilfe des Code-Generators eine Ereignisprozedur für dieses Ereignis anlegen, erstellt dieser im Formularmodul den folgenden Prozedurrumpf:

```
Private Sub Form_BeforeInsert (Cancel As Integer)
```

```
End Sub
```

Wie Sie bereits im *Kapitel 3 Die Programmiersprache VBA* erfahren haben, wird der Parameter per *Call by Reference* übergeben. In der Praxis bedeutet dies, dass Sie den Inhalt der Variablen während der Ausführung der Prozedur in *True* oder *False* verändern können. Sobald die Prozedur beendet ist, wertet Access den Inhalt der Variablen aus und reagiert entsprechend.

Über die *Cancel*-Variable haben Sie die Möglichkeit, das Anlegen eines neuen Datensatzes abubrechen. Standardmäßig hat die Variable den Inhalt *False*, wodurch die Neuanlage eines Datensatzes nicht abgebrochen wird. Erst wenn Sie ihr *True* zuweisen, bricht Access das Ereignis und damit den kompletten Vorgang ab.

○ *Cancel-Variable*

Die folgende Beispielprozedur finden Sie im Formular *Kunden3*. Sobald der Benutzer einen neuen Datensatz anlegen möchte, erscheint ein Meldungsfenster, in dem dies durch einen Klick auf die Schaltfläche *Ja* bestätigt werden muss. Die Prozedur stellt so sicher, dass nicht aus Versehen ein neuer Datensatz angelegt wird:

```
Private Sub Form_BeforeInsert(Cancel As Integer)
```

```
Dim Meldung As String
```

```
Meldung = "Möchten Sie wirklich einen neuen _  
Datensatz anlegen?"
```

```
If MsgBox(Meldung, vbYesNo + vbQuestion, _  
"Neuer Datensatz") = vbNo Then
```

```
Cancel = True
```

```
End If
```

```
End Sub
```





Sollte der Anwender die Frage zur Anlage des neuen Datensatzes mit *Nein* beantworten, gibt die *MsgBox*-Funktion den Wert der Konstanten *vbNo* (Wert 7) zurück. In diesem Fall wird die Variable *Cancel* auf den Wert *True* gesetzt, und der Vorgang wird abgebrochen. Sollte der Anwender *Ja* anklicken, gibt die *MsgBox*-Funktion den Wert der Konstanten *vbYes* (Wert 6) zurück und der Standardwert *False* der Variablen *Cancel* wird nicht verändert. Nachdem die Prozedur abgearbeitet ist, legt Access den Datensatz also wie gewohnt an.

Das *NachEingabe*-Ereignis

Während der Abarbeitung der *BeforeInsert*-Ereignisprozedur besitzen *AutoWert*-Felder immer noch den Inhalt *NULL*, da der Datensatz zu diesem Zeitpunkt noch nicht physikalisch angelegt ist.

Wenn Sie auf die Anlage eines neuen Datensatzes erst dann reagieren möchten, wenn der Datensatz bereits angelegt ist, verwenden Sie das Ereignis *AfterInsert*. Sie haben hier nicht nur die Möglichkeit, den Inhalt des Primärschlüsselfeldes zu ermitteln und damit zu arbeiten, sondern auch die Feldinhalte zu verändern. Sie können so beispielsweise Daten aus anderen Formularen oder Tabellen abfragen und direkt nach der Neuanlage des Datensatzes in einem Feld hinterlegen.

Im Gegensatz zum *BeforeInsert*-Ereignis besitzt die Ereignisprozedur *AfterInsert* den Parameter *Cancel* nicht:

```
Private Sub Form_AfterInsert ()
```

```
End Sub
```

Wenn Sie den Vorgang zu diesem (späten) Zeitpunkt noch abbrechen möchten, bleibt Ihnen nichts anderes übrig, als den bereits angelegten Datensatz wieder zu löschen.

Löschen von Datensätzen

Zum Löschen von Datensätzen gibt es eine ganze Reihe von Alternativen. Der direkteste Weg ist das Löschen mithilfe des entsprechenden Menüpunkts des Menüs *Bearbeiten*. Um einen Datensatz zu löschen, wählen Sie den Menüpunkt *Bearbeiten* > *Datensatz löschen*. Alternativ dazu markieren Sie zunächst durch einen Klick auf den Datensatzmarkierer den Datensatz oder wählen den Menüpunkt *Bearbeiten* > *Datensatz markieren*. Anschließend drücken Sie dann die Taste **Entf** oder wählen den Menüpunkt *Bearbeiten* > *Löschen* aus.





Löschen über Menübefehle

Genau die gleichen Schritte können Sie auch von VBA aus durchführen. Zur Auswahl der Menüpunkte stellt Access hierbei die Methode *RunCommand* zur Verfügung:

```
DoCmd.RunCommand <Command>
```

Jeder Menübefehl wird dabei durch eine Konstante repräsentiert. Für den Parameter *<Command>* geben Sie entsprechend dem gewünschten Menübefehl eine dieser Konstanten an.

Die *RunCommand*-Methode ersetzt die *DoMenuItem*-Methode aus früheren Access-Versionen. Bereits bestehende *DoMenuItem*-Methoden funktionieren jedoch weiterhin aus Gründen der Abwärtskompatibilität.



Hinweis

Entsprechend der *RunCommand*-Methode hat sich auch die Makroaktion *AusführenMenübefehl* in die Aktion *AusführenBefehl* geändert. Sie müssen nun nicht mehr die einzelnen Menüzeilenargumente eingeben, sondern wählen nur noch die passende Konstante aus. Diese stellt das Kombinationsfeld *Befehl* in deutscher Sprache zur Verfügung. Das lästige Abzählen von Menübefehlnummern für die *DoMenuItem*-Methode entfällt daher.



AusführenBefehl-Methode

In der folgenden Tabelle finden Sie die Konstanten der drei zuvor beschriebenen Menübefehle zum Löschen eines Datensatzes.

Beispiele für Befehlskonstanten der *RunCommand*-Methode



Englisch	Deutsch (in Makros)	entspricht Menübefehl
<i>acCmdSelectRecord</i>	<i>DatensatzMarkieren</i>	<i>Bearbeiten > Datensatz auswählen</i>
<i>acCmdDelete</i>	<i>Löschen</i>	<i>Bearbeiten > Löschen</i>
<i>acCmdDeleteRecord</i>	<i>DatensatzLöschen</i>	<i>Bearbeiten > Datensatz löschen</i>

Tabelle 6.7



Die Anweisung zum Löschen des aktuellen Datensatzes des aktuellen Formulars sieht damit folgendermaßen aus:

```
DoCmd.RunCommand acCmdDeleteRecord
```





Deaktivieren der Warnmeldung

Wenn Sie diese Anweisung einmal in eine *Click*-Ereignisprozedur einer Schaltfläche eingeben, die Sie auf dem Formular *Kunden* positionieren und schließlich ausführen, erscheint beim Löschen die folgende Warnmeldung:

Abb. 6.9

Warnmeldung beim
Löschen eines
Datensatzes



Diese Warnung ist wichtig, da so das unbeabsichtigte Löschen des Datensatzes verhindert werden kann. Der Text innerhalb der Meldung ist allerdings für den Endanwender etwas unverständlich. Um diese Meldung gegen ein selbstdefiniertes Meldungsfenster auszutauschen, muss die Standard-Warnmeldung deaktiviert werden. Dies lässt sich mithilfe der Methode *SetWarnings* bewerkstelligen:

```
DoCmd.SetWarnings [ <WarningsOn> ]
```

Wenn Sie diese Methode mit dem Parameter *False* angeben, unterdrückt Access in Zukunft alle Standard-Warnmeldungen. Die folgenden drei Anweisungen löschen den aktuellen Datensatz also ohne Bestätigung durch den Anwender:

```
DoCmd.SetWarnings False
DoCmd.RunCommand acCmdDeleteRecord
DoCmd.SetWarnings True
```



Hinweis

Nach dem Deaktivieren der Warnmeldungen ist es wichtig, sie wieder zu aktivieren, damit folgende Warnungen wie gewohnt angezeigt werden!

Was nun noch fehlt, ist eine selbstdefinierte Sicherheitsabfrage, die Sie leicht mithilfe der *MsgBox*-Funktion anlegen. Es folgt die *Click*-Ereignisprozedur der Schaltfläche *Datensatz löschen 1*, die Sie im Formular *Kunden4* finden:

```
Private Sub btn_Löschen1_Click()

    Dim Meldung As String

    Meldung = "Soll der Kunde wirklich gelöscht werden?"

    Beep

    If MsgBox(Meldung, vbYesNo + vbCritical, _
        "Kunde löschen") = vbYes Then
```



```
DoCmd.SetWarnings False
DoCmd.RunCommand acCmdDeleteRecord
DoCmd.SetWarnings True
```

```
End If
```

```
End Sub
```

Mithilfe der *RunCommand*-Methode kann auch ein momentan editierter Datensatz gespeichert werden. Dazu führen Sie einfach die folgende Anweisung aus:

```
DoCmd.RunCommand acCmdSaveRecord
```



Hinweis

Datensatz aus der Datenquelle löschen

Eine weitere Möglichkeit zum Löschen eines Datensatzes besteht darin, dass Sie diesen einfach aus der zugrunde liegenden Datenquelle entfernen. Im Fall des Formulars *Kunden* würde dies bedeuten, dass Sie den Datensatz direkt aus der Tabelle *Kunden* löschen. Um dies zu bewerkstelligen, können Sie z. B. auf die *DELETE*-Anweisung der Abfragesprache SQL zurückgreifen. Die folgende Prozedur löscht den Kunden mit der *KundenNr* 8:

```
Private Sub KundenLöschenSQL ()
```

```
Dim DB As Database
```

```
Set DB = CurrentDB
DB.Execute "DELETE * FROM Kunden WHERE KundenNr = 8"
```

```
End Sub
```

Die Aktionsabfrage zum Löschen des Datensatzes wird in diesem Beispiel mithilfe der *Execute*-Anweisung ausgeführt, die von Objekten des Typs *Database* zur Verfügung gestellt wird.



Löschen per SQL



Execute-Anweisung

Wenn die Löschabfrage mehrere Datensätze zum Ergebnis hat, werden alle Datensätze gelöscht. Da in der Bedingung der obigen Löschabfrage auf das (eindeutige) Primärschlüsselfeld *KundenNr* verwiesen wird, ist das Ergebnis entweder ein oder kein Datensatz. Das unbeabsichtigte Löschen mehrerer Datensätze muss also nicht befürchtet werden. Wenn die Bedingung jedoch mehreren Datensätzen entspricht, löscht die *Execute*-Anweisung alle Datensätze, die der Bedingung genügen. Um anschließend die Anzahl der gelöschten Datensätze herauszufinden, verwenden Sie die *RecordsAffected*-Eigenschaft.



Hinweis





Delete-Methode

- Eine weitere Möglichkeit stellt das Löschen per *Recordset*-Objekt und *Delete*-Methode dar:

```
Private Sub KundenLöschenDelete ()

Dim R As Recordset

    "Datensatzgruppe des Formulars klonen
    Set R = Me.RecordsetClone

    "Beide Datensatzgruppen synchronisieren
    R.Bookmark = Me.Bookmark

    "Datensatz löschen
    R.Delete

End Sub
```

In beiden Prozeduren wird immer nur der Kunde mit der Nummer 8 gelöscht. Wenn Sie den aktuellen Datensatz des Formulars *Kunden* löschen möchten, ermitteln Sie die *KundenNr* einfach aus dem gleichnamigen Feld des Formulars. Es folgt die Ereignisprozedur der Schaltfläche *Datensatz löschen 2*, die Sie ebenfalls im Formular *Kunden4* finden:

```
Private Sub btnLöschen2_Click()

Dim Meldung As String

    If Me.NewRecord = True Then Exit Sub

    Meldung = "Soll der Kunde wirklich gelöscht werden?"

    Beep

    If MsgBox(Meldung, vbYesNo + vbCritical, _
        "Kunde löschen") = vbYes Then

        DBEngine(0)(0).Execute _
            "DELETE * FROM Kunden WHERE KundenNr = " & _
            & Me!KundenNr

        Me.Requery

    End If

End Sub
```



Zum Löschen des aktuellen Datensatzes verwendet diese Prozedur die erste Variante unter Zuhilfenahme einer Löschabfrage. Vor dem Löschvorgang wird zuerst überprüft, ob das Formular überhaupt einen gültigen Datensatz besitzt – also nicht gerade ein neuer, leerer Datensatz der aktuelle Datensatz ist. Diese Überprüfung nimmt die erste *If*-Abfrage vor, indem die Eigenschaft *NewRecord* des Formulars auf den Inhalt *True* untersucht wird. Ist dies der Fall, wird die Prozedur sofort mithilfe der *Exit Sub*-Anweisung beendet.

Eine weitere Besonderheit der Prozedur hängt damit zusammen, wie Access Datensätze aus Formularen löscht. Ähnlich wie bei *Recordset*-Objekten des Typs *Dynaset* (siehe Kapitel 5 *Datenzugriff mit VBA*) werden gelöschte Datensätze lediglich als gelöscht markiert. In den einzelnen Datenfeldern des gelöschten Datensatzes zeigt Access in diesem Fall den Text *#Gelöscht* an.

Die *Requery*-Methode



Abb. 6.10

Ein gelöschter Datensatz

Aus diesem Grund ist es vorteilhaft, die Datensätze direkt nach dem Löschen zu aktualisieren. Dies erledigt die Methode *Requery*, die von allen Objekten des Typs *Form* zur Verfügung gestellt wird:

Datensätze aktualisieren

```
<Form-Objekt>.Requery
```

Die Aufgabe dieser Methode ist es, die Datenquelle der Formulars erneut von der bzw. den zugrunde liegenden Tabellen abzufragen.

In der oben abgedruckten Ereignisprozedur wird diese Methode auf das *Me*-Objekt angewendet, welches das aktuelle Formular repräsentiert. Nach der Ausführung der Methode sind alle Datensätze wieder auf dem neuesten Stand. Gleichzeitig positioniert Access im Formular auf den ersten Datensatz.



Hinweis



Wenn Sie die Reaktion der Prozedur einmal ohne *Requery*-Methode ausprobieren möchten, stellen Sie der Zeile *Me.Requery* einfach ein einfaches Anführungszeichen (") oder die *Rem*-Anweisung voran. Die Zeile wird von Access damit als Kommentar angesehen und nicht ausgeführt:

```
"Me.Requery  
Rem Me.Requery
```

Kontrolle des Löschvorgangs

Zum Löschen eines Datensatzes stellt Access dem Anwender eine Reihe von Möglichkeiten zur Verfügung. Eine Möglichkeit haben Sie bereits unter Zuhilfenahme des Menüpunkts *Bearbeiten > Datensatz löschen* kennen gelernt. Wenn der Anwender diesen Menüpunkt ausführt, haben Sie auf den ersten Blick keine Möglichkeit, in den Löschvorgang einzugreifen, um ihn beispielsweise zu verhindern.

Die Kontrolle wird nun dadurch ermöglicht, dass Access beim Löschen eines Datensatzes für das betreffende Formular ein Ereignis auslöst. Innerhalb einer entsprechenden Ereignisprozedur können Sie dann beispielsweise eine Sicherheitsabfrage unterbringen oder das Löschen komplett unterbinden.

Delete-Ereignis



Beim Löschen löst Access allerdings nicht nur ein, sondern gleich drei verschiedene Ereignisse aus. Diese unterscheiden sich in dem Zeitpunkt der Auslösung. Wenn Sie das Löschen eines Datensatzes einleiten (z. B. durch Markieren des Datensatzes und Drücken der Entf-Taste), löst Access als Erstes das *Delete*-Ereignis aus, auf das Sie mit der *BeimLöschen*-Eigenschaft reagieren können. Eine Ereignisprozedur, die Sie mithilfe des Code-Generators anlegen, hat folgenden Aufbau:

```
Private Sub Form_Delete (Cancel As Integer)
```

```
End Sub
```

Den Parameter *Cancel* haben Sie bereits im Zusammenhang mit dem Einfügen eines neuen Datensatzes kennen gelernt. Er dient dazu, den Löschvorgang bei Bedarf abubrechen. Zu diesem Zweck muss der Variablen *Cancel* während der Abarbeitung der Prozedur lediglich der logische Wert *True* zugewiesen werden. Die folgende Ereignisprozedur verhindert beispielsweise grundsätzlich das Löschen eines Datensatzes:

```
Private Sub Form_Delete (Cancel As Integer)
```

```
    Beep  
    MsgBox "Es dürfen keine Datensätze gelöscht werden!"  
    Cancel = True
```

```
End Sub
```



Wenn das Ereignis *Delete* nicht abgebrochen wurde, löst Access als Nächstes das *BeforeDelConfirm*-Ereignis aus. Die entsprechende Ereignisprozedur hat folgenden Aufbau:

```
Private Sub Form_BeforeDelConfirm (Cancel As Integer,
Response As Integer)

End Sub
```

Neben dem *Cancel*-Parameter, der wieder zum Abbrechen des Löschvorgangs dient, besitzt die Prozedur einen weiteren Parameter mit dem Namen *Response*. Über diesen Parameter können Sie festlegen, ob nach Abarbeitung der Prozedur die Standard-Warnmeldung angezeigt wird. Es sind folgende zwei Einstellungen möglich:

Einstellungen für den Parameter *Response*

Konstante	Beschreibung
<i>acDataErrContinue</i>	Die Standard-Warnmeldung wird nicht angezeigt.
<i>acDataErrDisplay</i>	Die Standard-Warnmeldung wird angezeigt. Dies ist die Standardeinstellung für den Parameter <i>Response</i> .

Tabelle 6.8

Dank dieses Parameters haben Sie nun die Gelegenheit, die Standard-Warnmeldung von Access durch Ihre eigene Sicherheitsabfrage zu ersetzen. Es folgt eine entsprechende Ereignisprozedur:

```
Private Sub Form_BeforeDelConfirm(Cancel As Integer,
Response As Integer)

Dim Meldung As String

Meldung = "Soll der Kunde wirklich gelöscht werden?"

Beep
If MsgBox(Meldung, vbCritical + vbYesNo, _
"Kunde löschen") <> vbYes Then

Cancel = True

End If

Response = acDataErrContinue

End Sub
```

BeforeDelConfirm-
Ereignis

Response-Parameter





Diese Ereignisprozedur finden Sie in dem Formular *Kundens* der Beispieldatenbank.

Wenn Sie nun auf irgendeine Weise einen Datensatz aus dem Formular *Kundens* löschen, zeigt Access stets die selbstdefinierte Meldung an. Wenn die Meldung positiv quittiert werden sollte, erfolgt schließlich der eigentliche Löschvorgang, bei dem der Datensatz von Access automatisch aus der zugrunde liegenden Tabelle entfernt wird. Andernfalls wird der Löschvorgang abgebrochen.

AfterDelConfirm-
Ereignis

Als Letztes löst Access das Ereignis *AfterDelConfirm* (*NachLöschbestätigung*) aus. Dies ist auch dann der Fall, wenn Sie vorher das Ereignis *BeforeDelConfirm* abgebrochen haben sollten. Die Ereignisprozedur hat folgenden Aufbau:

```
Private Sub Form_AfterDelConfirm (Status As Integer)

End Sub
```

Der Parameter *Status* dient in diesem Fall nicht dazu, das weitere Vorgehen von Access zu beeinflussen, sondern Ihnen mitzuteilen, ob der Datensatz von Access gelöscht wurde oder nicht. Der Parameter kann folgende Werte annehmen:

Werte des Parameters *Status*

Konstante	Beschreibung
<i>acDeleteOk</i>	Access hat den Datensatz erfolgreich gelöscht.
<i>acDeleteCancel</i>	Das Ereignis <i>BeforeDelConfirm</i> wurde durch Setzen der Variablen <i>Cancel</i> auf den Wert <i>True</i> abgebrochen. Der Datensatz wurde also nicht gelöscht.
<i>acDeleteUserCancel</i>	Der Löschvorgang wurde vom Benutzer über die Standard-Warnmeldung von Access abgebrochen.

Tabelle 6.9

Im Formular *Kundens* wird die folgende Ereignisprozedur verwendet, um dem Benutzer den Erfolg oder Misserfolg beim Löschen des Datensatzes mitzuteilen:

```
Private Sub Form_AfterDelConfirm(Status As Integer)

    Beep

    If Status = acDeleteOK Then

        MsgBox "Der Datensatz wurde erfolgreich _
               gelöscht!"

    Else
```



```
MsgBox "Der Löschvorgang wurde abgebrochen!"
```

```
End If
```

```
End Sub
```

Kontrolle der Dateneingabe

Ein wichtige Aufgabe des Entwicklers ist die Kontrolle der Dateneingabe in Formularen. Access stellt zu diesem Zweck bei Textfeldern die Eigenschaft *Gültigkeitsregel* zur Verfügung. Hier lässt sich eine einfache Regel eingeben, die Access beim Verlassen des betreffenden Steuerelements auswertet. In der *Gültigkeitsregel*-Eigenschaft des Feldes *Postleitzahl* wäre beispielsweise der Einsatz folgender Regel sinnvoll:

```
>=0 Und < 100000
```

Diese Regel stellt sicher, dass die Postleitzahl im Zahlenbereich von 0 bis 99.999 liegt. Wenn Sie eine Zahl außerhalb dieses Bereichs angeben, quittiert Access dies mit einer Fehlermeldung, deren Text Sie über die *Gültigkeitsmeldung*-Eigenschaft bestimmen können.

In vielen Fällen ist diese Art der Überprüfung von Feldinhalten allerdings nicht ausreichend. Dies ist beispielsweise dann der Fall, wenn bestimmte Feldinhalte sich gegenseitig ausschließen. Der Datensatz muss hierbei als Ganzes betrachtet und auf einen konsistenten Inhalt überprüft werden.

Genauso wichtig ist die Überprüfung bestimmter Felder auf das Vorhandensein eines Inhalts. So ist es beispielsweise bei einem neuen Kundendatensatz wichtig, dass in dem Feld *Name* ein Eintrag vorgenommen wurde.

VorAktualisierung- und NachAktualisierung-Ereignisse

Um solche Überprüfungen vorzunehmen, löst Access zwei Ereignisse mit den Namen *BeforeUpdate* (*VorAktualisierung*) und *AfterUpdate* (*NachAktualisierung*) aus, sobald ein editierter oder neu angelegter Datensatz in der zugrunde liegenden Tabelle gespeichert wird.

Beide Ereignisse unterscheiden sich wieder in dem Zeitpunkt der Auslösung. Das *BeforeUpdate*-Ereignis löst Access aus, noch bevor der Datensatz endgültig gespeichert wird. Das *AfterUpdate*-Ereignis tritt dagegen ein, wenn der Datensatz bereits physikalisch in der Tabelle hinterlegt wurde.

Die beiden Ereignisprozeduren haben folgenden Aufbau:

```
Private Sub Form_BeforeUpdate (Cancel As Integer)
```

```
End Sub
```

Gültigkeitsregeln





```
Private Sub Form_AfterUpdate ()
```

```
End Sub
```

Wie Sie an dem Parameter *Cancel* der *BeforeUpdate*-Ereignisprozedur erkennen, kann dieses Ereignis abgebrochen und das Speichern der Änderungen damit verhindert werden.

Das Vorhandensein
eines Feldinhalts
überprüfen

Die folgende Ereignisprozedur nutzt diese Möglichkeit, um das Feld *Name* des Formulars *Kunden* auf einen Inhalt zu überprüfen. Sollte es keinen Inhalt haben, wird der Benutzer mittels eines Meldungsfensters darauf hingewiesen. Nach dem Anzeigen des Meldungsfensters wird der Fokus auf das Feld *Name* verschoben. Dies ist nicht unbedingt nötig, es erleichtert dem Benutzer aber die Eingabe des fehlenden Namens.

```
Private Sub Form_BeforeUpdate (Cancel As Integer)
```

```
    If IsNull(Me!Name) = True Then
```

```
        Beep
```

```
        MsgBox "Es muss ein Kundenname angegeben  
werden!"
```

```
        Me!Name.SetFocus
```

```
        Cancel = True
```

```
    End If
```

```
End Sub
```

Filtern und Sortieren

Mit der *OpenForm*-Methode haben Sie bereits eine Methode kennen gelernt, mit deren Hilfe Sie die Datensätze eines Formulars filtern können. Der Nachteil dieser Methode ist allerdings, dass die Datensätze eines Formulars nur beim Öffnen gefiltert oder sortiert werden. Um jedoch auch nachträglich einen Filter oder eine Sortierung zu aktivieren, stellen Formulare die beiden Eigenschaften *Filter* und *OrderBy* (deutsch *Sortiert nach*) zur Verfügung.

Filter-Eigenschaft

Unter der *Filter*-Eigenschaft können Sie den *WHERE*-Teil einer *SQL-SELECT*-Abfrage angeben. Dies kann beispielsweise der Ausdruck *KundenNr = 5* oder *Name Like "M*"* sein. Wenn Sie der *Filter*-Eigenschaft eine solche Bedingung zugewiesen haben, wird der Filter jedoch nicht automatisch aktiv. Die Aktivierung erfolgt mittels der *FilterOn*-Eigenschaft, mit der sich der Filter an- oder abschalten lässt. Der Eigenschaft muss dazu einer der Werte *True* oder *False* zugewiesen werden. Das folgende Beispiel definiert für das Kundenformular einen Filter und aktiviert diesen anschließend:

```
Forms!Kunden.Filter = "Name Like "M*"
```

```
Forms!Kunden.FilterOn = True
```



Sobald der Filter aktiv ist, zeigt Access rechts neben den Navigationsschaltflächen den Vermerk (*Gefiltert*) an. Um den Filter wieder zu deaktivieren, setzen Sie die *FilterOn*-Eigenschaft auf *False*:

```
Forms!Kunden.FilterOn = False
```

Der Inhalt der *Filter*-Eigenschaft bleibt erhalten, bis Sie einen neuen Inhalt zuweisen oder das Formular schließen. Soll der Filter zusammen mit dem Formular gespeichert werden, müssen Sie die *Filter*-Eigenschaft in der Entwurfsansicht des Formulars verändern und die Änderungen anschließend speichern.

Der Zugriff auf die *Filter*-Eigenschaft ist auch beim Formularentwurf über das Eigenschaftsfenster des Formulars möglich. Hier können Sie bereits beim Entwurf einen vordefinierten Filter eintragen, der anschließend per *FilterOn*-Eigenschaft aktiviert werden kann.

In der Beispieldatenbank finden Sie ein Formular mit dem Namen *Kundenfiltern*. Das Formular enthält im Kopf ein Kombinationsfeld mit dem Namen *comFilter*, über das ein Anfangsbuchstabe ausgewählt werden kann. Sobald Sie hier einen Eintrag auswählen, filtert Access alle Datensätze, deren Feld *Name* mit dem ausgewählten Buchstaben beginnt.

Aktivieren und Deaktivieren des Filters



Hinweis

Abb. 6.11

Das Formular *Kundenfiltern*

Die einzelnen Buchstaben des Kombinationsfeldes werden über eine Werteliste definiert, die der *Datensatzherkunft*-Eigenschaft des Kombinationsfeldes beim Formularentwurf zugewiesen wurde. Damit dies funktioniert, muss die *Herkunftstyp*-Eigenschaft auf *Werteliste* eingestellt sein. Neben den Buchstaben enthält die Werteliste als ersten Eintrag den Text *Alle*, über den der Filter deaktiviert werden kann.





Sobald der Benutzer einen Eintrag auswählt, löst Access für das Kombinationsfeld das *Click*-Ereignis aus. Die entsprechende Ereignisprozedur der *BeimKlicken*-Eigenschaft hat folgenden Aufbau:

```
Private Sub comFilter_Click()  
  
    If Me!comFilter = "Alle" Then  
  
        "Filter deaktivieren  
        Me.FilterOn = False  
  
    Else  
  
        Me.Filter = "Name Like " & Me.comFilter & ""  
        Me.FilterOn = True  
  
    End If  
  
End Sub
```

Über eine *If*-Abfrage wird untersucht, ob der Eintrag *Alle* ausgewählt wurde. In diesem Fall wird der Filter deaktiviert, indem der *FilterOn*-Eigenschaft der Wert *False* zugewiesen wird. Sollte der Benutzer einen anderen Eintrag ausgewählt haben, wird eine Bedingung zusammengestellt und der *Filter*-Eigenschaft zugewiesen. Anschließend wird der Filter aktiviert. Ein Beispiel: Wenn der Anwender den Buchstaben *M* auswählt, erstellt Access folgende Bedingung:

```
Name Like "M*"
```

OrderBy-Eigenschaft

Neben dem Filter kann auch die Sortierreihenfolge nachträglich verändert werden. Wie bereits erwähnt, ist hierfür die *OrderBy*-Eigenschaft zuständig. Hier kann der *ORDER BY*-Teil einer *SQL-SELECT*-Abfrage angegeben werden. Wenn Sie beispielsweise lediglich den Namen eines Feldes angeben, erfolgt die Sortierung aufsteigend nach diesem Feld. Wenn Sie eine absteigende Sortierung wünschen, geben Sie hinter dem Feldnamen einfach ein Leerzeichen und den Text *DESC* an (z.B. *Name DESC*). Wenn nach mehreren Feldern sortiert werden soll, müssen Sie die einzelnen Feldnamen durch Kommas trennen (z. B. *Name, Vorname* oder *Name DESC, Vorname*).

Aktivieren der Sortierreihenfolge

Die Aktivierung der neuen Sortierreihenfolge erfolgt mittels der *OrderByOn*-Eigenschaft. Im Formular *Kundenfilter* finden Sie ein zweites Kombinationsfeld mit dem Namen *comSortierung*, über das eine Sortierung der Datensätze nach einem bestimmten Feld erfolgen kann. Dieses Kombinationsfeld hat eine Werteliste, in der alle wichtigen Felder des Kundenformulars aufgeführt sind. Der erste Eintrag *Keine Sortierung* deaktiviert die Sortierung. Die entsprechende *Click*-Ereignisprozedur hat den folgenden Aufbau:



```
Private Sub comSortierung_Click()

    If Me!comSortierung = "Keine Sortierung" Then

        "Sortierung deaktivieren"
        Me.OrderByOn = False

    Else

        Me.OrderBy = Me.comSortierung
        Me.OrderByOn = True

    End If

End Sub
```

Arbeiten mit formularbasierten Filtern

Über den Menüpunkt *Datensätze > Filter > Formularbasierter Filter* aktiviert Access einen Filterdialog für das aktuelle Formular, der vom Aussehen mit dem Formular identisch ist. Anstelle der Daten werden in die einzelnen Eingabefelder nun jedoch Kriterien eingegeben. Diese wandelt Access beim Schließen des Filterdialogs in eine Bedingung um, die schließlich die Datensätze des Formulars filtert.

Aktivieren und
Deaktivieren

Abb. 6.12

Der formularbasierte
Filter des
Kundenformulars



Wenn der Benutzer den formularbasierten Filter aktiviert, deaktiviert Access alle Steuerelemente, die nicht an Datenfelder gebunden sind. Dazu gehören beispielsweise Befehlsschaltflächen. In einigen Fällen ist es jedoch wünschenswert, dass noch weitere Steuerelemente deaktiviert oder sogar ausgeblendet werden, da hierfür keine Filterkriterien definiert werden sollen. Auf der anderen Seite kann es notwendig sein, bestimmte Steuerelemente nur dann einzublenden, wenn ein formularbasierter Filter aktiv ist.

Zur Kontrolle, wann der Benutzer einen formularbasierten Filter (oder auch jeden anderen Filter) bearbeitet oder anwendet, stellt Access die beiden Ereignisseigenschaften *BeiFilter* und *BeiAngewendetemFilter* zur Verfügung. Die dazu passenden Ereignisse tragen die Namen *Filter* und *ApplyFilter*.

Filter-Ereignis

Sobald der Benutzer eine Aktion ausführt, um einen Filter zu bearbeiten, löst Access das *Filter*-Ereignis aus. Hierbei ist es unerheblich, ob er einen formularbasierten Filter oder einen anderen Filter des Menüs *Datensätze > Filter* aktiviert. Die Ereignisprozedur zum *Filter*-Ereignis hat den folgenden Aufbau:

```
Private Sub Form_Filter(Cancel As Integer, FilterType
As Integer)
```

```
End Sub
```

Über den Parameter *Cancel* kann verhindert werden, dass der Anwender einen Filter definiert. In diesem Fall muss der Parameter *Cancel* während der Abarbeitung der Prozedur auf *True* gesetzt werden. Das zweite Argument *FilterType* gibt an, welchen Filter der Anwender definieren möchte. Access übergibt einen der folgenden Werte:

Werte des Parameters *FilterType*

Konstante	Wert	Beschreibung
<i>acFilterByForm</i>	0	Der Benutzer aktiviert einen formularbasierten Filter.
<i>acFilterAdvanced</i>	1	Der Benutzer aktiviert einen anderen Filter (z. B. einen <i>auswahlbasierten Filter</i> oder einen <i>Spezialfilter</i> . Die entsprechenden Menüpunkte dazu finden Sie im Menü <i>Datensätze > Filter</i>).

Tabelle 6.10



Das zweite Ereignis *ApplyFilter* wird ausgelöst, wenn der Benutzer einen definierten Filter anwenden möchte. Die Ereignisprozedur hat den folgenden Aufbau:

ApplyFilter-Ereignis

```
Private Sub Form_ApplyFilter(Cancel As Integer, _
    ApplyType As Integer)
```

```
End Sub
```

Auch hier haben Sie über den Parameter *Cancel* wieder die Möglichkeit, den Vorgang – also die Anwendung des Filters – abubrechen. Der Parameter *ApplyType* gibt genauere Auskunft darüber, welche Aktion zum Auslösen des Ereignisses geführt hat:

Werte des Parameters *ApplyType*

Konstante	Wert	Beschreibung
<i>acShowAllRecords</i>	0	Der Filter wird entfernt (z. B. über den Menüpunkt <i>Datensätze > Filter > Sortierung entfernen</i>).
<i>acApplyFilter</i>	1	Der zuletzt definierte Filter wird angewendet (z. B. über den Menüpunkt <i>Datensätze > Filter > Sortierung anwenden</i>).
<i>acCloseFilterWindow</i>	2	Der Filterdialog wird geschlossen. Dies betrifft sowohl den formularbasierten Filter als auch den Spezialfilter.

Tabelle 6.11

Mithilfe der beiden Ereignisse können Sie genau kontrollieren, wann der Benutzer einen Filter aktiviert und deaktiviert. In der Beispieldatenbank finden Sie das Formular *Kundefilter2*, das speziell an die Anwendung eines formularbasierten Filters angepasst wurde. Das Formular enthält eine Schaltfläche, über die der formularbasierte Filter aktiviert wird. Die *Click*-Ereignisprozedur hat den folgenden Aufbau:

```
Private Sub btnFilterAktivieren_Click()
```

```
    DoCmd.RunCommand acCmdFilterByForm
```

```
End Sub
```

Mithilfe der *RunCommand*-Methode wird der Menüpunkt *Datensätze > Filter > Formularbasierter Filter* ausgeführt. Sobald der Filter aktiviert wird, löst Access das *Filter*-Ereignis aus. In einer entsprechenden Ereignisprozedur werden nun die Steuerelemente *Telefonnummer*, *Telefax* und *Anmerkung* ausgeblendet, da diese zur Bildung eines Filters unwichtig sind. Es folgt die Ereignisprozedur:





```
Private Sub Form_Filter(Cancel As Integer, FilterType  
As Integer)
```

```
    If FilterType = acFilterByForm Then
```

```
        Me!Telefonnummer.Visible = False
```

```
        Me!Telefax.Visible = False
```

```
        Me!Anmerkung.Visible = False
```

```
    End If
```

```
End Sub
```

Die Prozedur überprüft zunächst anhand des *FilterType*-Ereignisses, ob der Anwender wirklich einen formularbasierten Filter und nicht etwa einen Spezialfilter aktiviert. Wenn dies der Fall ist, werden die drei Steuerelemente unsichtbar gemacht.

Nachdem der Filter geschlossen bzw. angewendet wurde, müssen die drei Steuerelemente wieder sichtbar gemacht werden. Diese Aufgabe erledigt die *ApplyFilter*-Ereignisprozedur:

```
Private Sub Form_ApplyFilter(Cancel As Integer,  
ApplyType As Integer)
```

```
    Me!Telefonnummer.Visible = True
```

```
    Me!Telefax.Visible = True
```

```
    Me!Anmerkung.Visible = True
```

```
End Sub
```

Filter speichern

Während Access nach einem neu definierten Filter das *ApplyFilter*-Ereignis auslöst, ist dieser bereits als SQL-Text in der *Filter*-Eigenschaft des Formulars abgelegt. Sie haben zu diesem Zeitpunkt also die Möglichkeit, den Filter zu speichern und damit für spätere Zecke abrufbereit zur Verfügung zu stellen. Ein entsprechendes Formular finden Sie in der Beispieldatenbank unter dem Namen *Kundenfilter3*.

Das Formular enthält zwei Schaltflächen, über die sich entweder ein formularbasierter oder ein Spezialfilter aktivieren lässt. Weiterhin enthält das Formular ein Kombinationsfeld mit dem Namen *comFilter*, über das ein gespeicherter Filter ausgewählt und angewendet werden kann. Das Kombinationsfeld basiert auf der Tabelle *Kundenfilter*, in der jeder Filter als SQL-Text zusammen mit einem Namen und einer *FilterNr* gespeichert wird. Das Kombinationsfeld ist an die Spalte *FilterNr* der Tabelle gebunden. Angezeigt werden allerdings nur die Namen der einzelnen Filter.



Die *ApplyFilter*-Ereignisprozedur erledigt das Speichern des Filters:

Speichern des Filters

```
Private Sub Form_ApplyFilter(Cancel As Integer,
ApplyType As Integer)

Dim Filtername As Variant
Dim RKundenfilter As Recordset
Dim FormFilter As String
Dim TabFilter As Variant

If ApplyType = acCloseFilterWindow _
Or ApplyType = acApplyFilter Then

    FormFilter = Me.Filter

    If Not IsNull(Me!comFilter) Then

        TabFilter = DLookup("SQL", "Kundenfilter", _
            "Filternr = " & Str(Me!comFilter))

        If Not IsNull(TabFilter) Then

            If FormFilter = TabFilter Then Exit Sub

        End If

    End If

    If MsgBox("Möchten Sie den Filter speichern?", _
        vbYesNo) = vbYes Then

        Filtername = InputBox("Bitte geben Sie den
            _ Filternamen ein.")

        If Len(Filtername) = 0 Then

            Exit Sub

        Else

            Set RKundenfilter = _
                DBEngine(0) (0).OpenRecordset("Kundenfilter")

            RKundenfilter.AddNew
            RKundenfilter.Filtername = Filtername
            RKundenfilter.SQL = Me.Filter
            RKundenfilter.Update

        End If

    End If

End Sub
```





```

        Me!comFilter.Requery

    End If

End If

End If

End Sub

```

Als Erstes untersucht die Funktion mithilfe einer *If*-Abfrage, ob der Filter geschlossen oder angewendet wird. Wenn eine der Bedingungen erfüllt ist, prüft die Funktion, ob der im Kombinationsfeld ausgewählte Filter dem aktuellen Filter entspricht. Ist dies nicht der Fall, zeigt Access ein Meldungsfenster an, über das der Benutzer das Speichern des Formulars bestätigen oder abbrechen kann. Wenn der Benutzer mit *Ja* bestätigt, wird per *InputBox*-Funktion ein Fenster zur Eingabe des Filternamens angezeigt. Eine weitere *If*-Abfrage überprüft, ob wirklich ein Name eingegeben wurde. In diesem Fall wird auf der Grundlage der Tabelle *Kundenfilter* eine Datensatzgruppe angelegt und der Filter gespeichert. Anschließend wird das Kombinationsfeld *comFilter* aktualisiert, so dass der neue Filter in der Liste erscheint.

Auswahl eines
gespeicherten
Filters

Wenn der Benutzer einen bereits gespeicherten Filter aus der Liste auswählt, löst Access für das Kombinationsfeld das *Click*-Ereignis aus, das mit folgender Ereignisprozedur verknüpft ist.

```

Private Sub comFilter_Click()

Dim SQL As Variant

    If IsNull(Me!comFilter) Then Exit Sub

    SQL = DLookup("SQL", "Kundenfilter", _
        "FilterNr = " & Str(Me!comFilter))

    If Not IsNull(SQL) Then

        Me.Filter = SQL
        Me.FilterOn = True

    End If

End Sub

```

Um sicherzustellen, dass wirklich ein gültiger Listeneintrag ausgewählt wurde, kontrolliert die Funktion das Kombinationsfeld auf den Inhalt *NULL* und bricht ggf. ab. Wenn ein Eintrag ausgewählt wurde, fragt die *DLookup*-Funktion im nächsten Schritt den SQL-Text des Filters aus der Tabelle *Kundenfilter* ab. Nachdem dieser SQL-Text der *Filter*-Eigenschaft zugewiesen wurde, aktiviert die Prozedur den Filter per *FilterOn*-Eigenschaft.



Dialogfenster

In dem vorhergehenden Abschnitt haben Sie bereits mehrere Arten von Dialogfenstern kennen gelernt. Ein typisches Beispiel für ein Dialogfenster ist beispielsweise das Meldungsfenster oder das *InputBox*-Dialogfenster zur Eingabe eines Textes. Beide Meldungsfenster lassen sich mithilfe von Parametern an Ihre Anforderungen anpassen. So können Sie beispielsweise bei Meldungsfenstern, die Sie mithilfe der *MsgBox*-Funktion anzeigen, den Meldungstext bestimmen.

In vielen Fällen reichen diese beiden Meldungsfenster allerdings nicht aus, da sie zu unflexibel sind. Access bietet daher die Möglichkeit, ein Formular als Meldungsfenster zu konfigurieren. Hier können Sie dann Steuerelemente Ihrer Wahl anordnen.

Das in der folgenden Abbildung abgedruckte Formular dient beispielsweise dazu, dem Benutzer die Auswahl eines Kunden zu ermöglichen. Zu diesem Zweck enthält es ein Kombinationsfeld mit einer Liste aller Kunden. Das Kombinationsfeld ist an die Spalte *KundenNr* gebunden, so dass ein Verweis auf das Steuerelement die *KundenNr* des ausgewählten Kunden zum Ergebnis hat.



Abb. 6.13

Ein halbfertiges Dialogfenster

Um aus diesem Formular nun ein "vernünftiges" Dialogfenster zu machen, sind einige spezielle Einstellungen der Formulareigenschaften erforderlich. Dazu gehört beispielsweise, dass Sie die Bildlaufleiste, die Navigationsschaltflächen und den Datensatzmarkierer mittels der drei gleichnamigen Eigenschaften deaktivieren. Außerdem sollten Sie die Eigenschaft *Automatisch zentrieren* auf *Ja* einstellen, damit das Formular beim Öffnen in der Mitte des Bildschirms erscheint. Eine weitere Einstellung betrifft die *Rahmenart*-Eigenschaft. Stellen Sie hier *Dialog* ein, um dem Formular den für Dialoge typischen Rahmen zu geben.

Formulareigenschaften für Dialoge

Als Letztes müssen auf dem Formular zwei Schaltflächen mit den Beschriftungen *OK* und *Abbrechen* angeordnet werden, über die der Benutzer die Auswahl des Kunden bestätigen oder abbrechen kann. Die beiden Schaltflächen besitzen die Namen *btnOk* und *btnAbbrechen*. Das fertige Dialogfenster könnte folgendermaßen aussehen:



Abb. 6.14

Das fertige Dialogfenster zur Auswahl eines Kunden



Sie finden das Formular in der Beispieldatenbank unter dem Namen *Kundenauswahl, Dialog*.

Öffnen von Dialogfenstern

Um das neu erstellte Dialogfenster nun zu öffnen, kann wieder die *OpenForm-Methode* verwendet werden, die Sie bereits am Anfang dieses Kapitels kennen gelernt haben. Zur Erinnerung folgt noch einmal die Syntax dieser Methode:

```
DoCmd.OpenForm <FormName> [, <View>] [, <FilterName>]
[, <WhereCondition>]
[, <DataMode>] [, <WindowMode>] [, <OpenArgs>]
```

Fenstermodus

Im Zusammenhang mit Dialogfenstern ist das Argument *<WindowMode>* von besonderem Interesse. Über dieses Argument bestimmen Sie, in welcher Art und Weise das Fenster des geöffneten Formulars erscheint:

Einstellungen des Arguments <WindowMode>

Konstante	Beschreibung
<i>acWindowNormal</i>	Öffnet das Fenster wie gewohnt. Dies ist die Standardeinstellung, falls Sie die Angabe dieses Arguments auslassen.
<i>acHidden</i>	Das Formular wird zwar geöffnet, es ist aber nicht sichtbar. In diesem Zustand sind Verweise auf das Formular uneingeschränkt möglich. Um das Formular wieder sichtbar zu machen, weisen Sie der <i>Visible</i> -Eigenschaft des Formulars den logischen Wert <i>True</i> zu.
<i>acIcon</i>	Das Fenster wird minimiert geöffnet und erscheint damit als Symbol am unteren Fensterrand von Access.
<i>acDialog</i>	Das Fenster wird als Dialog geöffnet.

Tabelle 6.12



Für den Einsatz mit Dialogfenstern ist besonders die letzte Einstellung *acDialog* von Interesse. Sie bewirkt, dass ein Fenster automatisch mit dem für Dialoge typischen Rahmen geöffnet wird. Wichtiger ist aber die Tatsache, dass Access die Ausführung der aktuellen Prozedur stoppt, bis das Dialogfenster wieder geschlossen wird. Die folgende Prozedur soll dies zeigen:

```
Private Sub Dialogtest1()

    MsgBox "Das Dialogfenster wird geöffnet!"

    DoCmd.OpenForm FormName:="Kundenauswahl", Dialog, _
        WindowMode:=acDialog

    MsgBox "Das Dialogfenster wurde geschlossen!"

End Sub
```

Bei der Ausführung dieser Prozedur zeigt Access zunächst die erste Meldung an und öffnet anschließend das Formular *Kundenauswahl, Dialog* als Dialogfenster. Die Programmausführung stoppt nun, bis das Dialogfenster wieder geschlossen wird. Erst jetzt zeigt Access die zweite Meldung an.

Eine weitere Besonderheit ist, dass von dem geöffneten Dialogfenster aus nicht zu einem anderen Access-Formular gewechselt werden kann. Das Dialogfenster muss also erst geschlossen werden, bevor mit Access weitergearbeitet werden kann. Dies eignet sich sehr gut dazu, den Anwender zu einer Eingabe zu zwingen.

Das Problem besteht nun darin, die ausgewählte *KundenNr* in Erfahrung zu bringen. Außerdem muss festgestellt werden, ob der Anwender den Vorgang mit *OK* bestätigt oder mit *Abbrechen* abgebrochen hat. Um dieses Problem zu lösen, kann folgende Strategie angewendet werden:

Das Formular wird zunächst wie beschrieben als Dialogfenster geöffnet. Wenn der Anwender einen Abbruch vornehmen möchte, klickt er auf die Schaltfläche *Abbrechen*, wodurch das Formular geschlossen wird. Bestätigt der Anwender dagegen mithilfe der *OK*-Schaltfläche, wird das Formular lediglich ausgeblendet. Auf die *OpenForm-Methode* hat dies die gleiche Wirkung wie das Schließen des Dialogfensters. Die Programmausführung wird also ebenfalls fortgesetzt.

Es folgen die beiden Ereignisprozeduren der Schaltflächen *OK* und *Abbrechen*, die das Ausblenden bzw. Schließen des Dialogfensters erledigen:

```
Private Sub btnOk_Click ()

    If IsNull(Me!KundenNr) Then

        Beep
        MsgBox "Es ist kein Kunde ausgewählt!"

    Else
```

OK- und Abbrechen-Schaltfläche





```
Me.Visible = False

End If

End Sub

Private Sub btnAbbrechen_Click()

    DoCmd.Close acForm, Me.Name

End Sub
```

Die Ereignisprozedur der *OK*-Schaltfläche blendet das Dialogfenster aus, indem der *Visible*-Eigenschaft der Wert *False* zugewiesen wird. Vorher wird allerdings noch überprüft, ob wirklich ein Kunde ausgewählt wurde. Sollte dies nicht der Fall sein, gibt die Prozedur eine Meldung aus und lässt das Dialogfenster geöffnet.

Die zweite Ereignisprozedur der *Abbrechen*-Schaltfläche schließt das Formular dagegen auf die bekannte Art und Weise mithilfe der *Close*-Methode.

Abfragen der Daten aus dem Dialogfenster

Sollte der Anwender das Dialogfenster mit *OK* bestätigt haben, ist dieses lediglich unsichtbar. Auf das Kombinationsfeld und damit die *KundenNr* kann aber nach wie vor mithilfe des folgenden Ausdrucks zugegriffen werden:

```
Forms![Kundenauswahl, Dialog].KundenNr
```

Die Daten des Dialogfensters – in diesem Fall also der ausgewählte Kunde – werden nun von der Prozedur ausgelesen, die das Dialogfenster geöffnet hat. Nach dem Öffnen muss die Prozedur hierzu lediglich überprüfen, ob das Formular *Kundenauswahl, Dialog* noch geöffnet ist. Ist dies der Fall, hat der Anwender *nicht* auf *Abbrechen* geklickt und die gewünschte Kundennummer kann aus dem Formular abgerufen werden. Das folgende Beispiel zeigt, wie die Prozedur aufgebaut sein könnte:

```
Sub Dialogtest2()

    Dim KundenNr As Variant

    DoCmd.OpenForm FormName:="Kundenauswahl, Dialog", _
        WindowMode:=acDialog

    If IstGeöffnet("Kundenauswahl, Dialog") = True Then

        KundenNr = Forms![Kundenauswahl, _
            Dialog].KundenNr

        DoCmd.Close acForm, "Kundenauswahl, Dialog"
```





```
MsgBox "Ausgewählte KundenNr: " & KundenNr
```

```
End If
```

```
End Sub
```

Ob das Formular *Kundenauswahl, Dialog* noch geöffnet ist, wird mithilfe der bereits bekannten Funktion *IstGeöffnet* überprüft. Wenn die Funktion *True* zum Ergebnis hat und das Formular damit noch geöffnet ist, wird der Wert des Kombinationsfeldes *KundenNr* aus dem Dialogfenster in die Variable *KundenNr* eingelesen. Anschließend wird das unsichtbare Dialogfenster geschlossen und die *KundenNr* in einem Meldungsfenster angezeigt.

Der Vorgang zur Auswahl einer *KundenNr* kann mithilfe der folgenden Funktion verallgemeinert werden:

Kundennummer
auswählen

```
Function KundenNrAbfragen () As Variant
```

```
DoCmd.OpenForm FormName:="Kundenauswahl, Dialog", _  
WindowMode:=acDialog
```

```
If IstGeöffnet("Kundenauswahl, Dialog") = True Then
```

```
KundenNrAbfragen = Forms![Kundenauswahl, _  
Dialog].KundenNr
```

```
DoCmd.Close acForm, "Kundenauswahl, Dialog"
```

```
Else
```

```
KundenNrAbfragen = 0
```

```
End If
```

```
End Function
```

In diesem Beispiel wird die ausgewählte *KundenNr* als Ergebnis der Funktion *KundenNrAbfragen* zurückgegeben. Sollte der Anwender den Vorgang abbrechen, gibt die Funktion den Wert 0 zurück.

Wenn Sie diese Funktion in einem globalen Modul platzieren, können Sie sie von jedem Ort Ihrer Datenbank aus aufrufen. Dazu ist lediglich ein Aufruf der folgenden Art erforderlich:

```
KundenNr = KundenNrAbfragen()
```





Informationen an Dialogfenster übergeben

Bis jetzt haben Sie nur eine Möglichkeit kennen gelernt, eine oder mehrere Informationen von einem Dialogfenster abzufragen. Mit einem kleinen Trick ist es aber auch möglich, Informationen an ein Dialogfenster zu übergeben. So wäre es beispielsweise wünschenswert, wenn Sie der Funktion *KundenNrAbfragen* einen Text übergeben könnten, der innerhalb des Dialogfensters erscheint:

```
KundenNr = KundenNrAbfragen("Bitte wählen Sie den _
                             Kunden aus, den Sie löschen möchten!")
```

Kundennummer
abfragen

Um eine einzelne Information, wie z.B. einen Text, an ein Formular zu übergeben, stellt die *OpenForm-Methode* ein Argument mit dem Namen *<Öffnungsargumente>* zur Verfügung. Hierbei handelt es sich um einen Parameter des Datentyps *Variant*, für den Sie neben Texten z. B. auch Zahlen- und Datumswerte oder Variant-Arrays angeben können. Die modifizierte Funktion *KundenNrAbfragen* könnte nun den folgenden Aufbau haben:

```
Function KundenNrAbfragen (Text As String) As Variant

    DoCmd.OpenForm FormName:="Kundenauswahl, Dialog", _
        WindowMode:=acDialog, _
        OpenArgs:=Text

    If IstGeöffnet("Kundenauswahl, Dialog") = True Then

        KundenNrAbfragen = Forms![Kundenauswahl, _
                                Dialog].KundenNr

        DoCmd.Close acForm, "Kundenauswahl, Dialog"

    Else

        KundenNrAbfragen = 0

    End If

End Function
```

Der einzige Unterschied zur vorhergehenden Funktion besteht in dem Parameter *Text*, der beim Öffnen des Formulars übergeben wird. Durch die Übergabe des Textes erscheint dieser allerdings noch nicht als Meldungstext innerhalb des Dialogfensters. Diese Aufgabe muss vom Formular selbst bzw. der *Open*-Ereignisprozedur dieses Formulars erledigt werden.





Innerhalb des geöffneten Dialogfensters können die übergebenden Öffnungsargumente mithilfe der *OpenArgs*-Eigenschaft abgefragt werden. Wenn Sie dem Formular beispielsweise den Text "Bitte wählen Sie einen Kunden aus" übergeben, hat der Ausdruck

```
Me.OpenArgs
```

genau diese Zeichenkette zum Ergebnis.

Damit der Text direkt beim Öffnen des Dialogfensters erscheint, muss der Inhalt der *OpenArgs*-Eigenschaft beim Auftreten des *Open*-Ereignisses ausgelesen und als Überschrift in das Dialogfenster geschrieben werden.

Das einzige Problem ist nun noch, *wie* der Text in das Dialogfenster "geschrieben" wird. Zu diesem Zweck eignet sich beispielsweise ein Bezeichnungsfeld sehr gut, dem Sie einfach mithilfe der *Caption*-Eigenschaft eine neue Aufschrift zuweisen. Im Formular *Kundenauswahl*, *Dialog* befindet sich bereits ein solches Bezeichnungsfeld mit dem Namen *lblMeldung*. Dieses wird von der folgenden *Open*-Ereignisprozedur des Dialogfensters zum Anzeigen der Meldung verwendet:

```
Private Sub Form_Open (Cancel As Integer)
```

```
Dim Text As Variant
```

```
    Text = Me.OpenArgs
```

```
    Me.lblMeldung.Caption = Text
```

```
End Sub
```

In der Prozedur wird der übergebene Text mithilfe der *OpenArgs*-Eigenschaft des Formulars abgefragt und in der Variablen *Text* abgelegt. Im nächsten Schritt wird die Beschriftung des Bezeichnungsfeldes *lblMeldung* durch den übergebenen Text ersetzt.

Übergabe mehrerer Informationen

Obwohl das Argument *<Öffnungsargumente>* der *OpenForm-Methode* im Plural angegeben ist, lässt sich damit immer nur eine einzelne Information – also eine Zahl oder ein Text – übergeben. Wenn die Übergabe mehrerer Informationen erforderlich ist, versagt das Öffnungsargument. Sie können natürlich mit dem *Variant*-Parameter ein Variant-Array bilden und in diesem mehrere Parameter unterbringen, wodurch aber die Übersichtlichkeit meist stark leidet.

In diesem Fall bleibt Ihnen nichts anderes übrig, als die Informationen mittels öffentlicher Variablen zu übergeben. Da öffentliche Variablen von allen Datenbankobjekten aus sichtbar sind, können Sie sowohl von einer Startprozedur aus als auch innerhalb eines Dialogfensters auf den Inhalt zugreifen.



Die *OpenArgs*-
Eigenschaft



Öffentliche
Variablen





Titeltext übergeben

Das folgende Beispiel ist eine Erweiterung der Funktion *KundenNrAbfragen*. Neben einem Text wird hier als zweite Information ein weiterer Text übergeben, der im Titel des Dialogfensters erscheinen soll:

```
Function KundenNrAbfragen(Text As String, Titel As  
String) As Variant  
  
    p_Meldung = Text  
    p_Titel = Titel  
  
    DoCmd.OpenForm FormName:="Kundenauswahl, Dialog", _  
        WindowMode:=acDialog  
  
    If IstGeöffnet("Kundenauswahl, Dialog") = True Then  
  
        KundenNrAbfragen = Forms![Kundenauswahl, _  
            Dialog].KundenNr  
  
        DoCmd.Close acForm, "Kundenauswahl, Dialog"  
  
    Else  
  
        KundenNrAbfragen = 0  
  
    End If  
  
End Function
```

Innerhalb der Prozedur werden die Inhalte der beiden Parameter den globalen Variablen *p_Meldung* und *p_Titel* zugewiesen, die vorher im Deklarationsteil eines Standardmoduls auf die folgende Weise deklariert wurden:

```
Public p_Meldung As String  
Public p_Titel As String
```

Open-Ereignisprozedur des Dialogfensters

In der *Open*-Ereignisprozedur des Dialogfensters werden nun anstelle der Öffnungsargumente die Inhalte der beiden globalen Variablen ausgewertet:

```
Private Sub Form_Open(Cancel As Integer)  
  
    Me!lblMeldung.Caption = p_Meldung  
    Me.Caption = p_Titel  
  
End Sub
```



Die fertige Funktion könnte nun beispielsweise zum Löschen eines Kunden eingesetzt werden:

Dialog zum Löschen von Kunden

```
Private Sub KundeLöschen()

Dim KundenNr As Variant
Dim DB As DATABASE

    KundenNr = KundenNrAbfragen( _
        "Bitte geben Sie den Kunden an, " & _
        " den Sie löschen möchten!", "Löschen")

    If KundenNr = 0 Then

        Beep
        MsgBox "Der Kunde wurde nicht gelöscht!"

    Else

        DB.Execute "DELETE * FROM Kunden WHERE _
            KundenNr = " & KundenNr

        Beep
        MsgBox "Der Kunde wurde gelöscht!"

    End If

End Sub
```

Positionierung, Größenänderung und Aktivierung von Formularen

Im vorhergehenden Abschnitt haben Sie im Zusammenhang mit dem Dialogfenster *Kundenauswahl*, *Dialog* bereits erfahren, wie Sie ein Formular beim Öffnen automatisch in der Bildschirmmitte platzieren. Dazu muss lediglich die Eigenschaft *Automatisch zentrieren* auf *Ja* eingestellt sein. Hat diese Eigenschaft dagegen den Inhalt *Nein*, öffnet Access das Formular automatisch an der Position, die das Formular beim letzten Speichern besaß.

Mithilfe der *MoveSize*-Methode lässt sich ein Formular auch nach dem Öffnen von VBA aus verschieben. Eine weitere Aufgabe dieser Methode ist die Größenänderung eines Formulars. Die Methode hat folgende Syntax:

Die *MoveSize*-Methode

```
DoCmd.MoveSize [<Right>] [, <Down>] [, <Width>] [, <Height>]
```



Die einzelnen Parameter haben folgende Bedeutung:

Parameter der *MoveSize*-Methode

Parameter	Beschreibung
<Right>, <Down>	Legen die Position der linken oberen Ecke des Fensters fest.
<Width>	Bestimmt die Breite des Formulars.
<Height>	Bestimmt die Höhe des Formulars.

Tabelle 6.13

Hinweis



Alle Angaben erfolgen in der Maßeinheit *Twips*. Ein Zentimeter entspricht 567 *Twips*.

Die Angabe der einzelnen Parameter ist optional. Wenn Sie beispielsweise nur die Parameter <Right> und <Down> angeben, verschiebt Access das Formular unter Beibehaltung der Breite und der Höhe:

```
DoCmd.MoveSize 500, 500
```

Wenn Sie dagegen z. B. die Höhe auf 0 verändern möchten, geben Sie den folgenden Ausdruck an:

```
DoCmd.MoveSize Height:=0
```

Formular verschieben



Da sich die *MoveSize*-Methode immer auf das momentan aktive Formular bezieht, muss vorher sichergestellt werden, dass das passende Formular auch wirklich aktiviert ist. Das folgende Beispiel öffnet das Formular *Kunden* und verschiebt es in die linke obere Fensterecke:

```
Private Sub KundenformularVerschieben ()

    DoCmd.OpenForm "Kunden"
    DoCmd.MoveSize 0, 0

End Sub
```



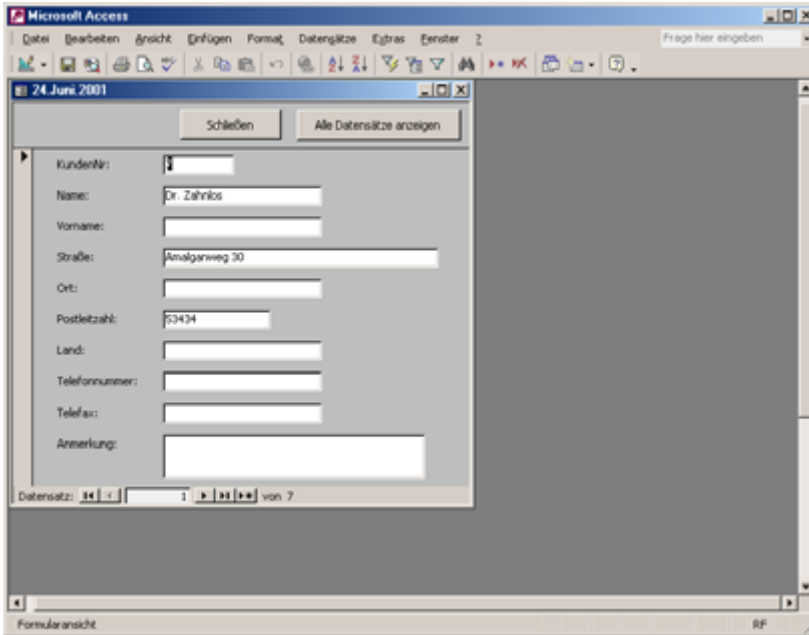


Abb. 6.15

Das in die linke obere Ecke verschobene Kundenformular

Kontrolle der Größenänderung

Neben der Möglichkeit, die Größe eines Formulars mithilfe der *MoveSize*-Methode von VBA aus zu verändern, kann natürlich auch der Anwender mithilfe der Maus eine Größenänderung eines Formulars vornehmen. Voraussetzung ist lediglich, dass die *Rahmenart*-Eigenschaft des Formulars auf *Veränderbar* eingestellt ist. Bei den übrigen Einstellungen *Keine*, *Dünn* und *Dialog* ist keine Größenänderung möglich.

In einigen Fällen ist es sinnvoll, den Inhalt eines Formulars an die aktuelle Größe des Formulars anzupassen. Um dies zu ermöglichen, löst Access bei einer Größenänderung durch den Anwender das *Resize*-Ereignis aus, auf das Sie mithilfe der *Bei-Größenänderung*-Ereigniseigenschaft reagieren können. Die entsprechende Ereignisprozedur enthält keine Parameter und besitzt damit den folgenden Aufbau:

```
Private Sub Form_Resize ()
```

```
End Sub
```

Ein praktisches Beispiel für die Reaktion auf eine Größenänderung finden Sie in der Beispieldatenbank. Das Formular *Navigator* enthält eine Liste mit dem Namen *IstKundenliste*, in der alle Kunden aufgelistet sind. Ein Doppelklick auf einen Kundennamen führt dazu, dass Access das Kundenformular mit dem passenden Kunden öffnet.

Kunden-Navigator-fenster



Abb. 6.16

Das Formular Navigator



Wenn Sie das Formular des Navigators in der Größe verändern, werden Sie feststellen, dass auch die Kundenliste an die neue Größe angepasst wird. So wird die Formularfläche des Navigators optimal ausgenutzt.

Formulargröße
ermitteln

Um die Größenanpassung der Liste zu ermöglichen, müssen erst einmal Informationen über die aktuellen Formularausmaße ermittelt werden. Zu diesem Zweck stellen Formulare die beiden Eigenschaften *InsideHeight* und *InsideWidth* zur Verfügung. In diesen Eigenschaften ist die Höhe und Breite des inneren Formularbereichs ohne Ränder und Formulartitel enthalten. Die Maßeinheit ist – wie bei allen Maßen, die Sie innerhalb von VBA angeben – wieder Twips.

Listenbreite
anpassen

Um nun die Breite der Liste an die Formularbreite anzupassen, kann innerhalb der *Resize*-Ereignisprozedur der folgende Ausdruck verwendet werden:

```
Me!lstKundenliste.Width = Me.InsideWidth
```

Links und rechts von der Liste befindet sich allerdings noch ein kleiner Rand, so dass die Liste etwas schmaler als das Formular sein muss:

```
Me!lstKundenliste.Width = Me.InsideWidth - 200
```

Laufzeitfehler
verhindern

Im Zusammenhang mit der Subtraktion tritt ein weiteres Problem auf: Wenn die Breite des Formulars beispielsweise 150 Twips beträgt, berechnet sich die Listenbreite zu dem ungültigen Wert -50. Access löst in diesem Fall einen Laufzeitfehler aus. Diesen können Sie entweder abfangen oder von vornherein verhindern, indem Sie sicherstellen, dass die Breite nicht unter 0 sinken kann. Am einfachsten bewerkstelligen Sie dies mit einer kleinen Unterprozedur, die den Namen *Max* trägt:

```
Function Max (a, b)

    Max = IIf(a > b, a, b)

End Function
```





Diese Prozedur erwartet beim Aufruf zwei Werte. Das Ergebnis der Prozedur ist immer der größere dieser beiden Werte. Wenn Sie die Funktion beispielsweise mit den Parametern 0 und -100 aufrufen, ist das Ergebnis 0:

```
Max(0, -100)
```

Die Funktion kann nun auf folgende Weise eingesetzt werden:

```
Me!lstKundenliste.Width = Max(0, Me.InsideWidth - 200)
```

Sobald also der Ausdruck *Me.WindowWidth - 200* negativ wird, gibt die Funktion den Wert 0 zurück.

Die komplette *Resize*-Ereignisprozedur muss natürlich neben der Formularbreite auch die Formularhöhe berücksichtigen:

```
Private Sub Form_Resize()  
  
    Me!lstKundenliste.Width = Max(0, _  
        Me.InsideWidth - 200)  
    Me!lstKundenliste.Height = Max(0, _  
        Me.InsideHeight - 500)
```

```
End Sub
```

Sie finden diese Ereignisprozedur im Modul des Formulars *Navigator*. Die Funktion *Max* ist in dem Modul *Beispielprozeduren* enthalten.

Aktivierung und Deaktivierung von Formularen

Oft ist es erforderlich, darüber informiert zu werden, wann ein bestimmtes Formular aktiviert oder deaktiviert wird. Aus diesem Grund löst Access bei jeder Aktivierung und Deaktivierung eines Formulars die beiden Ereignisse *Activate* und *Deactivate* aus. Auf diese beiden Ereignisse können Sie mithilfe der Ereignisseigenschaften *BeiAktivierung* und *BeiDeaktivierung* reagieren. Die entsprechenden Ereignisprozeduren besitzen folgenden Aufbau:

```
Private Sub Form_Activate ()  
  
End Sub  
  
Private Sub Form_Deactivate ()  
  
End Sub
```



BeiAktivierung- und
BeiDeaktivierung-
Ereigniseigen-
schaften





Das *Deactivate*-Ereignis

- Ein beliebiger Einsatzbereich des *Deactivate*-Ereignisses ist das Speichern des aktuellen Datensatzes. Wenn der Benutzer beispielsweise einen Datensatz editiert, spiegeln sich diese Änderungen erst dann in der zugrunde liegenden Tabelle wider, wenn eine der folgenden Aktionen erfolgt:
 - Der Anwender wechselt den aktuellen Datensatz.
 - Der Anwender klickt auf den Datensatzmarkierer.
 - Der Anwender wählt den Menüpunkt *Datensätze > Datensatz speichern* aus.
 - Das Formular wird geschlossen.

Wenn der Anwender also während des Editierens zu einem zweiten Formular wechselt, ohne vorher eine der obigen Aktionen ausgeführt zu haben, enthält die Tabelle noch die veralteten Daten. Erfolgt nun innerhalb des zweiten Formulars ein Zugriff auf diese Daten, sind diese nicht auf dem aktuellen Stand.

Um den Anwender nicht mit dem ständigen Speichern von Datensätzen zu belasten, sollte dies immer dann automatisch erfolgen, wenn das betreffende Formular deaktiviert wird. Dies bewerkstelligen Sie mithilfe der folgenden *Deactivate*-Ereignisprozedur:

```
Private Sub Form_Deactivate ()  
  
    DoCmd.RunCommand acCmdSaveRecord  
  
End Sub
```

Die *RunCommand*-Methode

Die *RunCommand*-Methode, die Sie bereits im Abschnitt *Löschen von Datensätzen* kennen gelernt haben, führt einfach den Menübefehl *Datensätze > Datensatz speichern* aus. Der aktuelle Datensatz wird damit also immer beim Wechsel zu einem anderen Formular gespeichert.

Hinweis



Neben den Ereignissen *Activate* und *Deactivate* stellen Formulare auch die Ereignisse *GotFocus* und *LostFocus* zur Verfügung. Diese Ereignisse werden von Access nur dann ausgelöst, wenn ein Formular aktiviert bzw. deaktiviert wird, das keine Steuerelemente enthält. In der Praxis spielen diese Ereignisse daher kaum eine Rolle

RollUp-Formulare

Eine weitere, vielleicht nicht ganz so ernste Anwendung der Ereignisse *Activate* und *Deactivate* ist die Möglichkeit, einem Formular beim Deaktivieren mithilfe der *MoveSize*-Methode die Höhe 0 zuzuweisen:

```
Private Sub Form_Deactivate()  
  
    DoCmd.MoveSize Height:=0  
  
End Sub
```



Wenn das Formular erneut aktiviert wird, erhält es wieder die ursprüngliche Höhe:

```
Private Sub Form_Activate()  
  
    DoCmd.MoveSize Height:=6225  
  
End Sub
```

Die Wirkung können Sie anhand des Formulars *Kunden6* der Beispieldatenbank begutachten. Sobald Sie das Kundenformular deaktivieren, wird es platzsparend verkleinert, so dass nur noch der Formulartitel sichtbar ist. Wenn Sie diesen anklicken und das Formular damit aktivieren, erscheint das Formular wieder in voller Größe.



Abb. 6.17

Das platzsparend verkleinerte Kundenformular

Zeitgeberprogrammierung

Mithilfe von so genannten *Timern* ist es möglich, eine Prozedur in regelmäßigen Zeitabständen ausführen zu lassen. Timer werden ausschließlich von Formularen zur Verfügung gestellt, wobei ein Formular immer genau einen Timer enthält.

Zur Steuerung des Timers besitzen Formulare die *Zeitgeberintervall*-Eigenschaft, mit der Sie das Intervall des Timers bestimmen. Die Länge des Intervalls wird in Millisekunden angegeben. Der Wert 1000 bedeutet beispielsweise, dass Access jede Sekunde einmal das *Timer*-Ereignis (*BeiZeitgeber*-Ereigniseigenschaft) auslöst. Maximal kann ein Wert von 2.147.483.647 Millisekunden (entspricht ca. 25 Tagen) angegeben werden. Der Wert 0 bedeutet, dass der Timer deaktiviert ist.

Der Umgang mit Timern ist sehr einfach. Öffnen Sie einfach ein Formular, und tragen Sie unter der *Zeitgeberintervall*-Eigenschaft den Wert 1000 ein. Anschließend legen Sie über die *Bei Zeitgeber*-Ereigniseigenschaft die folgende Ereignisprozedur an:

```
Private Sub Form_Timer()  
  
    Beep  
  
End Sub
```

Wenn Sie nun in die Formularansicht umschalten, wird der Timer gestartet und es ertönt jede Sekunde ein Signalton.

Zeitgeberintervall-Eigenschaft

Bei Zeitgeber-Ereigniseigenschaft





Mit einer etwas abgewandelten Version dieser Ereignisprozedur ist es beispielsweise möglich, immer die aktuelle Uhrzeit im Titel eines Formulars anzuzeigen:

```
Private Sub Form_Timer()  
  
    Me.Caption = Format(Now, "hh:nn:ss")  
  
End Sub
```

Ein entsprechendes Formular finden Sie in der Beispieldatenbank unter dem Namen *Kunden7*.

Hinweis



Wenn Sie unter der *Zeitgeberintervall*-Eigenschaft sehr kurze Intervalle eingeben, kann Access unter Umständen die *Timer*-Ereignisprozedur nicht komplett abarbeiten, bevor das nächste *Timer*-Ereignis auftritt. In diesem Fall gehen die betreffenden *Timer*-Ereignisse verloren.

Tipp



Sollten Sie einmal ausschließlich einen Timer und kein Formular benötigen, öffnen Sie das Formular einfach unsichtbar. Dies bewerkstelligen Sie beispielsweise mithilfe der folgenden Anweisung:

```
DoCmd.OpenForm FormName:="Timer-Formular",  
WindowMode:=acHidden
```

Das Formular ist so für den Anwender nicht sichtbar, der Timer funktioniert aber trotzdem auf die gewohnte Art und Weise.

Abfangen von Formularfehlern

Im *Kapitel 3 Die Programmiersprache VBA* haben Sie bereits Möglichkeiten kennen gelernt, um einen Laufzeitfehler, der innerhalb einer Prozedur auftritt, abzufangen. Aber nicht nur in VBA-Programmcode, sondern auch innerhalb von Formularen können Fehler auftreten. Diese Fehler werden allerdings nicht vom Programmcode, sondern von der JET-Datenbank-Engine gemeldet.

Doppelte Indexwerte

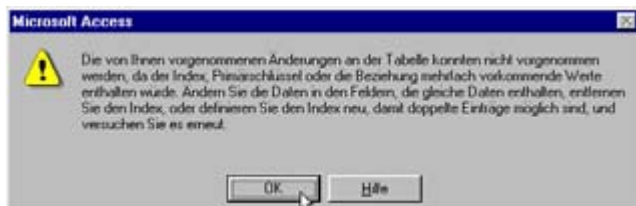


Ein typischer Fehler, der von der Datenbank-Engine gemeldet wird, ist die doppelte Eingabe eines Feldinhalts, dessen Feld mit einem eindeutigen Index versehen ist. In diesem Fall meldet Access beim Speichern des Datensatzes folgenden Fehler:

Abb. 6.18



Fehlermeldung bei doppeltem Indexeintrag



Diese Meldung mag für einen Entwickler noch verständlich sein; für den Anwender ist sie allerdings nicht sehr informativ. Beim Auftreten solcher Fehler löst Access daher noch vor dem Anzeigen der Fehlermeldung das *Error*-Ereignis aus, auf das Sie mithilfe der *BeiFehler*-Eigenschaft reagieren können. Die entsprechende Ereigniseigenschaft hat den folgenden Aufbau:

```
Private Sub Form_Error (DataErr As Integer, _  
    Response As Integer)  
  
End Sub
```

Der Parameter *DataErr* gibt den Code des Fehlers an, der aufgetreten ist. Dieser Fehlercode ist identisch mit dem Fehlercode der *Number*-Eigenschaft des *Err*-Objekts, das Sie im *Kapitel 3 Die Programmiersprache VBA* kennen gelernt haben. Jeder Fehlercode repräsentiert hierbei einen bestimmten Fehler. Wenn Sie den entsprechenden Fehlertext zu einem Fehler ermitteln möchten, wenden Sie einfach die *Error*-Funktion an:

```
MsgBox Error(DataErr)
```

Der zweite Parameter *Response* dient dazu, das Anzeigen der Standardfehlermeldung von Access zu unterdrücken. Dem Parameter kann zu diesem Zweck einer der beiden folgenden Werte zugewiesen werden:

Konstanten des Parameters *Response*

Konstante	Beschreibung
<i>acDataErrContinue</i>	Die Standardfehlermeldung wird nicht angezeigt.
<i>acDataErrDisplay</i>	Die Standardfehlermeldung wird angezeigt. Diese Einstellung ist die Standardeinstellung.

Tabelle 6.14

Mithilfe dieses Parameters können Sie nun die Standardfehlermeldung durch Ihre eigene ersetzen. Dazu zeigen Sie einfach eine Meldung mit Ihrem Fehlertext an und unterdrücken anschließend das Anzeigen der Standardfehlermeldung. Das folgende Beispiel zeigt, wie eine Ereignisprozedur zum Abfangen des doppelten Indexinhalts aufgebaut sein kann:

```
Private Sub Form_Error (DataErr As Integer, Response  
As Integer)  
  
    If DataErr = 3022 Then
```

Fehlermeldung anzeigen

Eigene Fehlermeldungen anzeigen





```
Beep
MsgBox "Sie haben einen Feldinhalt eingegeben, "
      & "der bereits vorhanden ist!"

Response = acDataErrContinue

End If

End Sub
```

Die Beispielprozedur finden Sie im Formular *Artikel* der Beispieldatenbank. Das Datenfeld *Artikelbezeichnung* ist hier mit einem eindeutigen Index versehen. Wenn Sie also einen neuen Datensatz anlegen und eine bereits vorhandene Artikelbezeichnung eingeben, führt Access die obige Prozedur aus.

Hinweis



Der Fehlercode 3022 des Indexfehlers wurde vor dem Anlegen der Prozedur ermittelt, indem der Fehler provoziert und der Inhalt des *DataErr*-Parameters im Testfenster ausgegeben wurde. Sollte ein anderer Fehler auftreten, zeigt Access die Standardmeldung an.

Wie geht es weiter?

Nachdem Sie nun in diesem Kapitel den Umgang mit Formularen und deren Ereignissen kennen gelernt haben, geht es im nächsten Kapitel um die einzelnen Steuerelemente, die ebenfalls eine Vielzahl von Ereignissen auslösen können.





Steuerelemente programmieren



Textfelder	372
Optionsfelder, Kontrollkästchen und Umschaltflächen	381
Befehlsschaltflächen	388
Listen- und Kombinationsfelder	389
Seitenumbrüche	406
Register	408
Hyperlinks	411





Access stellt eine Palette von insgesamt 18 Steuerelementen zur Verfügung, mit denen sich Formulare individuell gestalten lassen. Steuerelemente besitzen die Aufgabe, dem Anwender den Zugriff auf Daten zu ermöglichen und die Ausführung von Aktionen zu vereinfachen.

Genau wie Formulare können auch fast alle 18 Steuerelementtypen Ereignisse auslösen, die speziell das jeweilige Steuerelement betreffen. Eine Ausnahme bilden lediglich die Steuerelementtypen *Rechteck* und *Linie*, die ausschließlich der optischen Verschönerung von Formularen dienen.

In diesem Kapitel geht es darum, wie Sie VBA effektiv zur Steuerung und Kontrolle von Steuerelementen einsetzen. Hierzu werden die einzelnen Steuerelementtypen anhand praktischer Beispiele erläutert. Als Datenzugriffsmodell wird DAO verwendet.

Objektfelder, in denen Sie OLE-Objekte ablegen können, zählen zwar ebenfalls zu den Steuerelementen; aufgrund ihrer Komplexität werden diese allerdings im *Kapitel 13 OLE und DDE* erläutert.

Alle Beispiele zu diesem Kapitel finden Sie in der Datenbank *Kapitel7.mdb*.

Textfelder

Steuerelement- inhalt-Eigenschaft

Die wichtigsten Steuerelemente bei der Entwicklung einer Datenbankanwendung sind Textfelder. Sie dienen dazu, einen Text oder einen Zahlenwert der Datenquelle zu repräsentieren, die dem Formular zugrunde liegt. Zu diesem Zweck werden Textfelder mithilfe der Eigenschaft *Steuerelementinhalt* an ein bestimmtes Datenfeld gebunden. Durch diese Verknüpfung stellt Access in dem Textfeld automatisch immer den Feldinhalt des aktuellen Datensatzes dar.

Ungebundene Textfelder

Neben diesen an ein Datenfeld gebundenen Textfeldern können in Access auch so genannte *ungebundene Textfelder* eingesetzt werden, bei denen die Eigenschaft *Steuerelementinhalt* keinen Inhalt besitzt. Ein solches Steuerelement haben Sie bereits im vorhergehenden Kapitel im Zusammenhang mit dem Suchen nach einem Kunden kennen gelernt. Da ein ungebundenes Textfeld mit keinem Tabellenfeld verknüpft ist, wird bei einem Wechsel des Datensatzes der Inhalt weder gelöscht noch verändert. Eine weitere Besonderheit ungebundener Textfelder besteht darin, dass die hier eingegebenen Daten beim Schließen des Formulars verloren gehen.

Im folgenden Abschnitt geht es darum, wie Sie die Eingabe in ein Textfeld kontrollieren. Bei gebundenen Steuerelementen ist diese Kontrolle besonders wichtig: Sie können so bereits während der Eingabe überprüfen, ob der Anwender zulässige Daten eingibt.



Überwachen der Dateneingabe

Im vorhergehenden Kapitel haben Sie bereits eine Möglichkeit kennen gelernt, die Benutzereingaben eines Formulars zu überwachen. Sobald ein Datensatz gespeichert wird, löst das Formular die Ereignisse *BeforeUpdate* und *AfterUpdate* aus, mit deren Hilfe Sie eine Überprüfung der einzelnen Feldinhalte vornehmen können.

BeforeUpdate-
und *AfterUpdate-*
Ereignisse

Textfelder und fast alle anderen Steuerelementtypen stellen ebenfalls die Ereignisse *BeforeUpdate* und *AfterUpdate* zur Verfügung. Diese Ereignisse beziehen sich hierbei allerdings nicht auf das Formular, sondern auf das betreffende Steuerelement.

Anders als bei Formularen werden die Ereignisse *BeforeUpdate* und *AfterUpdate* nicht erst ausgelöst, wenn der Feldinhalt in der zugrunde liegenden Datenquelle gespeichert wird, sondern bereits dann, wenn der Inhalt eines Steuerelements verändert wurde und es den Fokus verliert.

Wichtig ist hierbei, dass Access die Ereignisse nur dann auslöst, wenn an dem Steuerelementinhalt wirklich eine Änderung vorgenommen wurde. Wenn Sie also beispielsweise im Kundenformular das Feld *Name* aktivieren, aber keine Änderung vornehmen, führt der Wechsel zu einem anderen Steuerelement, nicht zum Auslösen der beiden Ereignisse. In der Praxis lässt sich dies sehr gut verwenden, um den Inhalt eines Textfeldes bereits direkt nach der Eingabe isoliert zu betrachten und zu überprüfen.

Genau wie bei Formularen unterscheiden sich die beiden Ereignisse *BeforeUpdate* und *AfterUpdate* in dem Zeitpunkt, zu dem sie ausgelöst werden. Das *BeforeUpdate*-Ereignis tritt in den folgenden beiden Fällen auf:

- Der Inhalt des Steuerelements wurde verändert und der Fokus wird auf ein anderes Steuerelement des gleichen Formulars verschoben.
- Der Inhalt des Steuerelements wurde verändert und der Datensatz wird (z. B. über den Menübefehl *Datensätze > Datensatz speichern* oder durch das Schließen des Formulars) gespeichert.

Ursachen für das
BeforeUpdate-
Ereignis

In beiden Fällen kann der Vorgang noch abgebrochen werden. Die Ereignisprozedur besitzt dementsprechend einen Parameter mit dem Namen *Cancel*:

```
Sub <Steuerelementname>_BeforeUpdate (Cancel As Integer)
```

```
End Sub
```

Wird nun während der Abarbeitung der Ereignisprozedur der Variablen *Cancel* der Wert *True* zugewiesen, bricht Access den Vorgang ab. In der Praxis bedeutet dies, dass das betreffende Steuerelement weiterhin den Fokus besitzt und der Datensatz ggf. nicht gespeichert wurde.





Eingabe der
Vorwahlnummer
erzwingen

Um beispielsweise sicherzustellen, dass im Kundenformular die Telefonnummer stets inklusive Vorwahlnummer eingegeben wird, kann die folgende Ereignisprozedur eingesetzt werden. Sie finden die Prozedur im Formular *Kunden* der Beispieldatenbank:

```
Private Sub Telefonnummer_BeforeUpdate(Cancel As Integer)

    If Left(Me!Telefonnummer, 1) <> "0" Then

        Beep
        MsgBox "Die Telefonnummer muss inklusive " & _
            "Vorwahlnummer angegeben werden!"
        Cancel = True

    End If

End Sub
```

Diese Prozedur überprüft, ob die in das Steuerelement *Telefonnummer* eingegebene Nummer mit der Ziffer 0 beginnt. Sollte dies nicht der Fall sein, fehlt die Vorwahlnummer. Anschließend wird eine entsprechende Fehlermeldung ausgegeben und der Vorgang abgebrochen.

Hinweis



Beachten Sie, dass die obige Ereignisprozedur nur dann ausgeführt wird, wenn der Inhalt des Feldes *Telefonnummer* wirklich verändert wurde. Befindet sich in dem Tabellenfeld bereits eine ungültige Eingabe, die auf einem anderen Weg in die Tabelle *Kunden* gelangte, findet die Überprüfung erst dann statt, wenn die Telefonnummer vom Formular aus verändert wird.

Das zweite Ereignis *AfterUpdate* wird von Access ausgelöst, wenn der Fokus bereits auf ein anderes Steuerelement verschoben wurde. In der Programmierpraxis hat dieses Ereignis im Zusammenhang mit der Überprüfung des Feldinhalts wenig Bedeutung.

Kontrolle der Eingabe

Das *BeforeUpdate*-Ereignis wird von Access immer *nach* dem vollständigen Abschluss einer Eingabe ausgelöst. Wenn Sie aber bereits während der Eingabe eine Überprüfung vornehmen möchten, ist dieses Ereignis nicht geeignet.

Change-Ereignis

Als Ersatz stellt Access bei Textfeldern das *Change*-Ereignis zur Verfügung. Dieses Ereignis wird bei *jeder* Änderung am Inhalt eines Textfeldes ausgelöst. Wenn Sie beispielsweise in das Formular *Kunden* einen Namen eingeben, der aus fünf Zeichen besteht, löst Access das Ereignis fünfmal aus. Auch das Entfernen von Zeichen mithilfe der Tasten **Entf** oder **↵** ist eine Änderung, die zur Auslösung des *Change*-Ereignisses führt.



Um Ihnen die Möglichkeit zur Reaktion auf das *Change*-Ereignis zu geben, stellt Access bei Textfeldern die *BeiÄnderung*-Ereigniseigenschaft zur Verfügung. Die entsprechende Ereignisprozedur hat keine Parameter.

Das folgende Beispiel verwendet das *Change*-Ereignis, um die Anzahl der Zeichen, die in dem Textfeld *Name* des Kundenformulars eingegeben werden, zu beschränken. Zu diesem Zweck wird die aktuelle Länge mithilfe der *Len*-Funktion ermittelt und mit einem Grenzwert verglichen. Sie finden die Prozedur im Formular *Kunden* der Beispieldatenbank:

```
Private Sub Name_Change()  
  
Const MaxLänge = 20  
  
If IsNull(Me!Name.Text) Then Exit Sub  
  
If Len(Me!Name.Text) > MaxLänge Then  
  
    Beep  
    MsgBox "Der Name darf aus maximal" & _  
        Str(MaxLänge) & " Zeichen bestehen!"  
  
End If  
  
End Sub
```

In diesem Beispiel wird mithilfe der *Text*-Eigenschaft des Steuerelements auf den Inhalt des Textfeldes zugegriffen. Dies ist erforderlich, da der normale Zugriff über den Ausdruck *Me!Name* noch den alten Text vor der Änderung zum Ergebnis hätte. Erst wenn das Steuerelement den Fokus verliert und Access damit das *NachAktualisierung*-Ereignis auslöst, stimmt das Ergebnis des Ausdrucks *Me!Name* wieder mit dem aktuellen Inhalt überein. Die *Text*-Eigenschaft enthält dagegen stets den aktuellen Inhalt.



Hinweis

Tastaturereignisse

Eine noch bessere Kontrolle einer Eingabe erlaubt das Ereignis *KeyPress*, das von Access bei jeder Betätigung einer Taste ausgelöst wird. Die entsprechende Ereignisprozedur der *BeiTaste*-Ereigniseigenschaft hat folgenden Aufbau:

```
Sub <Steuerelementname>_KeyPress (KeyAscii As Integer)  
  
End Sub
```

Über den Parameter *KeyAscii* erfahren Sie, welche Taste vom Anwender gedrückt wurde. Hierbei wird der Zeichencode der gedrückten Taste übergeben, der in dem für die Tastatur relevanten Bereich mit dem ANSI-Code übereinstimmt. Eine Tabelle dieser Codes finden Sie in der auf der CD zum Buch enthaltenen Referenz.



KeyPress-Ereignis





Um die Wirkung des *KeyPress*-Ereignisses auszuprobieren, legen Sie für das Steuerelement *Ort* des Formulars *Kunden* eine neue Ereignisprozedur an, die Sie dann mit einer Anweisung zur Ausgabe des Zeichencodes füllen. Die Ereignisprozedur könnte folgendermaßen aussehen:

```
Sub Ort_KeyPress (KeyAscii As Integer)

    MsgBox Str(KeyAscii)

End Sub
```

Wenn Sie nun in die Formularansicht umschalten und einen Ort eingeben, erscheint bei jeder Tastenbetätigung ein Meldungsfenster mit dem entsprechenden Zeichencode.

Wie Sie eventuell beim Ausprobieren des Beispiels bemerkt haben, erscheint beim Drücken der Taste zunächst das Meldungsfenster. Erst wenn Sie dieses mit *OK* bestätigen, erscheint das entsprechende Zeichen in dem Steuerelement *Ort*. Das *KeyPress*-Ereignis wird also ausgelöst, noch bevor Access die Betätigung der Taste verarbeitet.

Tastatureingaben
ignorieren

In der Praxis kann dies dazu verwendet werden, bestimmte gedrückte Tasten zu ignorieren. In diesem Fall weisen Sie dem Parameter *KeyAscii* während der Ausführung der Ereignisprozedur einfach den Wert 0 zu. Die folgende Ereignisprozedur verhindert beispielsweise die Eingabe der deutschen Umlaute sowie des Zeichens "ß":

```
Private Sub Name_KeyPress(KeyAscii As Integer)

    Dim Zeichen As String

    Zeichen = Chr(KeyAscii)

    If Zeichen = "ö" Or Zeichen = "ä" Or _
       Zeichen = "ü" Or Zeichen = "ß" Then

        Beep
        KeyAscii = 0

    End If

End Sub
```

Die Prozedur verwendet eine Funktion mit dem Namen *Chr*, die Sie bereits aus dem *Kapitel 2 Schnellstart VBA* kennen. Die Funktion gibt zu einem bestimmten Zeichencode das jeweilige Zeichen als Zeichenkette zurück. Den umgekehrten Vorgang, nämlich die Umwandlung eines Zeichens in den dazugehörigen Zeichencode, erledigt die *ASC*-Funktion.



Eine weitere Anwendung des *KeyPress*-Ereignisses besteht darin, eine Tastaturbetätigung in ein anderes Zeichen umzuwandeln. Dem *KeyAscii*-Parameter muss zu diesem Zweck einfach der neue Zeichencode zugewiesen werden.

Tastatureingaben
umwandeln

Die folgende Ereignisprozedur ersetzt bereits während der Eingabe das Leerzeichen gegen den Unterstrich (_). Sie finden die Prozedur in dem Formular *Kunden* der Beispieldatenbank. Sie ist an die *BeiTaste*-Eigenschaft des Feldes *Land* gebunden:

```
Private Sub Land_KeyPress(KeyAscii As Integer)

    If Chr(KeyAscii) = " " Then

        KeyAscii = Asc("_")

    End If

End Sub
```

Mithilfe des *KeyPress*-Ereignisses lässt sich ein einzelner Tastendruck immer nur in ein einzelnes neues Zeichen umwandeln. Es wäre aber vorteilhaft, wenn beispielsweise bei Betätigung der Taste **B** im Textfeld automatisch die Zeichen *ss* erscheinen würden.

Dieses Problem kann mithilfe eines kleinen Tricks gelöst werden: Bei Betätigung der Taste **B** wird diese zunächst einmal ignoriert, indem der Parameter *KeyAscii* auf den Wert 0 gesetzt wird. Anschließend werden mithilfe der Funktion *SendKeys* die zwei Tastenanschläge **S****S** simuliert. Die *SendKeys*-Funktion hat hierbei die gleiche Wirkung, als wenn die als Parameter übergebene Zeichenkette direkt über die Tastatur eingegeben worden wäre. Die Ereignisprozedur zur Umwandlung des Zeichens *ß* in *ss* könnte folgendermaßen aussehen:

```
Sub Straße_KeyPress (KeyAscii As Integer)

    If Chr(KeyAscii) = "ß" Then

        KeyAscii = 0
        SendKeys "ss"

    End If

End Sub
```

Die folgende Erweiterung der Prozedur ist an das Feld *Straße* des Kundenformulars geknüpft. Sie wandelt alle deutschen Sonderzeichen entsprechend um:

```
Private Sub Straße_KeyPress(KeyAscii As Integer)

    Select Case Chr(KeyAscii)
```



```

Case "ä": SendKeys "ae": KeyAscii = 0
Case "ü": SendKeys "ue": KeyAscii = 0
Case "ö": SendKeys "oe": KeyAscii = 0
Case "ß": SendKeys "ss": KeyAscii = 0

```

```
End Select
```

```
End Sub
```

Funktions- und Steuertasten

KeyDown- und KeyUp-Ereignisse

Eine Schwäche des *KeyPress*-Ereignisses ist, dass sich die Betätigung von Funktions-tasten wie **[Alt]** oder **[Strg]** nicht ermitteln lässt. Um dies zu ermöglichen, muss auf die Ereignisse *KeyDown* und *KeyUp* zurückgegriffen werden. Diese Ereignisse werden immer dann ausgelöst, wenn eine Taste niedergedrückt bzw. gelöst wird. Im Zusammenhang mit dem *KeyPress*-Ereignis ergibt sich bei der Betätigung einer Taste folgende Reihenfolge, in der Access die Ereignisse beim Drücken einer Taste auslöst:

```
KeyDown => KeyPress => KeyUp
```

Die Ereignisprozeduren der beiden Ereignisse *KeyDown* und *KeyUp* besitzen identische Parameterlisten. Als Beispiel folgt der Aufbau der *KeyDown*-Ereignisprozedur:

```

Sub <Steuerelementname>_KeyDown (KeyCode As Integer, _
Shift As Integer)

```

```
End Sub
```

Der KeyCode- Parameter

Der Parameter *KeyCode* ist vergleichbar mit dem Parameter *KeyAscii* des *KeyPress*-Ereignisses. Der Unterschied besteht darin, dass ein Code immer genau auf eine Taste bezogen ist. Bei der Betätigung der Taste **[A]** wird beispielsweise immer der *KeyCode* 65 zurückgegeben. Dies ist unabhängig davon, ob Sie ein kleines a oder (in Verbindung mit der **[⇧]**-Taste) ein großes A eingeben. Der Parameter *KeyAscii* des *KeyPress*-Ereignisses nimmt dagegen eine entsprechende Unterscheidung vor und gibt zwei verschiedene Zeichencodes aus.

Eine weitere Besonderheit des *KeyCode*-Parameters ist, dass sich damit auch die Betätigung der Funktionstasten **[F1]** bis **[F12]** und einiger weiterer Sondertasten ermitteln lässt.

[F1]-Taste abfragen

Um beispielsweise zu ermitteln, ob die Taste **[F1]** gedrückt wurde, vergleichen Sie den Inhalt des Parameters *KeyCode* einfach mit der Konstanten *vbKeyF1*:

```
If KeyCode = vbKeyF1 Then
```

```
    MsgBox "Es wurde die Taste F1 betätigt!"
```

```
End If
```



Der zweite Parameter mit dem Namen *Shift* erlaubt es festzustellen, ob zusammen mit der gedrückten Taste eine oder mehrere der Tasten ,  oder  gedrückt wurden. Die einzelnen Tasten werden hierbei durch folgende Konstanten repräsentiert:

Der -Parameter

Konstanten des Parameters *Shift*

Konstante	Entspricht Taste
<i>acShiftMask</i>	
<i>acCtrlMask</i>	
<i>acAltMask</i>	

Tabelle 7.1

Um herauszufinden, welche Tastenkombination gedrückt wurde, müssen die Konstanten als Bitmasken verwendet werden. Der folgende Ausdruck hat den Wert *True* zum Ergebnis, wenn die -Taste gedrückt wurde:

```
(Shift And acCtrlMask) > 0
```

Der folgende Ausdruck testet, ob die Tasten  und  gleichzeitig gedrückt wurden:

```
(Shift And acCtrlMask) > 0 And (Shift And acAltMask) > 0
```

Mithilfe des *KeyDown*-Ereignisses könnten Sie beispielsweise die Funktionstasten mit Textbausteinen belegen, die von Access automatisch in ein Textfeld eingetragen werden. Die folgende Prozedur ist an das Feld *Ort* des Formulars *Kunden* gebunden. Wenn das Steuerelement *Ort* den Fokus hat und Sie die Tastenkombination ++ drücken, trägt Access in das Feld den Ortsnamen *Düsseldorf* ein. Wird dagegen ++ gedrückt, trägt Access den Namen *Köln* ein:

```
Private Sub Ort_KeyDown(KeyCode As Integer, Shift As Integer)

    If (Shift And acShiftMask) > 0 And _
        (Shift And acCtrlMask) > 0 Then

        Select Case Chr(KeyCode)

            Case "D": Me!Ort = "Düsseldorf": KeyCode = 0
            Case "K": Me!Ort = "Köln": KeyCode = 0
```



```
End Select
```

```
End If
```

```
End Sub
```

Nachdem überprüft wurde, ob die beiden Tasten  und  niedergedrückt sind, wird mithilfe des Parameters *KeyCode* untersucht, ob zusätzlich eine der Tasten  oder  betätigt wurden. Wenn dies der Fall ist, wird der betreffende Ortsnamen in das Textfeld geschrieben. Anschließend wird der *KeyCode* auf 0 gesetzt, damit die Zeichen D bzw. K nicht zusätzlich noch in dem Textfeld erscheinen.

Eine Möglichkeit zur Erweiterung dieser Prozedur wäre eine Tabelle, in der alle Textbausteine zusammen mit den jeweiligen Testcodes abgelegt sind. Die Prozedur könnte dann aus dieser Tabelle den jeweiligen Textbaustein abrufen.

Hinweis



Formulare stellen ebenfalls die drei Ereignisseigenschaften *BeiTaste*, *BeiTasteAb* und *BeiTasteAuf* zur Verfügung. Die Ereignisse werden jedoch nur bei Formularen ausgelöst, die keine Steuerelemente beinhalten. Wenn Sie allerdings die *Tastenvorschau*-Eigenschaft eines Formulars auf *Ja* einstellen, werden alle Tastaturereignisse, die für die Steuerelemente des Formulars ausgelöst werden, zunächst an das Formular weitergeleitet. Sie haben so die Möglichkeit, auf bestimmte Tasten global für das komplette Formular zu reagieren.

Kontrolle des Fokus

Das *LostFocus*-Ereignis



Innerhalb von Access hat immer nur ein bestimmtes Steuerelement den Fokus. "Fokus" bedeutet in diesem Zusammenhang, dass das betreffende Steuerelement auf Ihre Tastatureingaben reagiert. Zur Kontrolle des Fokus löst Access beim Fokusverlust eines Steuerelements das *LostFocus*-Ereignis aus, auf das Sie mithilfe der Eigenschaft *BeiFokusverlust* reagieren können.

Das *GotFocus*-Ereignis



Neben dem *LostFocus*-Ereignis stellt Access das *GotFocus*-Ereignis zur Verfügung, das immer dann ausgelöst wird, wenn das betreffende Steuerelement den Fokus erhält. Die Ereignisseigenschaft zu diesem Ereignis trägt den Namen *BeiFokuserhalt*.

Enter- und Exit-Ereignisse



Zwei weitere Ereignisse mit dem Namen *Enter* und *Exit* sind eng mit den Ereignissen *GotFocus* und *LostFocus* verwandt. Sie werden ebenfalls ausgelöst, wenn ein Steuerelement den Fokus erhält bzw. verliert.

Der Unterschied besteht darin, dass die Ereignisse *Enter* und *Exit* nur bei einem Fokuswechsel *innerhalb* eines Formulars ausgelöst werden. Wenn Sie beispielsweise dem Textfeld *Name* des Formulars *Kunden* den Fokus zuweisen und anschließend den Fokus auf das Feld *Vorname* verschieben, löst Access für das Steuerelement *Name* des Formulars *Kunden* sowohl das *Exit*- als auch das *LostFocus*-Ereignis aus. Wenn Sie dagegen ein Steuerelement auf einem anderen Formular (z. B. dem Formular *Rechnungen*) aktivieren, löst Access ausschließlich das *LostFocus*-Ereignis aus.



Ein weiterer Unterschied betrifft das Ereignis *Exit*. Die entsprechende Ereignisprozedur zu diesem Ereignis besitzt den bereits bekannten Parameter *Cancel*. Über diesen können Sie das Verlassen des betreffenden Steuerelements verhindern. Im Zusammenhang mit diesem Ereignis können Sie ebenfalls eine Kontrolle der Eingabe vornehmen. Wenn die Eingabe nicht korrekt war, geben Sie eine Meldung aus und setzen den Parameter *Cancel* auf *True*. Das Steuerelement kann nun nicht mehr verlassen werden, bis eine korrekte Eingabe erfolgt.

In dem Beispielformular *Kunden1* werden die Ereignisse *Enter* und *Exit* dazu genutzt, das fokussierte Steuerelement farbig zu hinterlegen. Dazu wird die *BackColor*-Eigenschaft (Deutsch: *Hintergrundfarbe*-Eigenschaft) beim Eintritt (*Enter*-Ereignis) in das Steuerelement verändert. Beim Verlassen wird die weiße Hintergrundfarbe wiederhergestellt. Pro Steuerelement müssen also zwei Ereignisprozeduren angelegt werden. Nachfolgend sind exemplarisch die beiden Ereignisprozeduren des Steuerelements *Name* abgedruckt:

```
Private Sub Name_Enter()  
  
    Me!Name.BackColor = 16776960  
  
End Sub  
  
Private Sub Name_Exit(Cancel As Integer)  
  
    Me!Name.BackColor = 16777215  
  
End Sub
```

Die Farbcodes für die gewünschte Hintergrundfarbe können ermittelt werden, indem einem Steuerelement in der Entwurfsansicht testweise die gewünschte Farbe zugewiesen wird. Der Wert kann dann aus der *Hintergrundfarbe*-Eigenschaft des Eigenschaftsfensters ausgelesen werden. Eine andere Möglichkeit stellen die beiden Funktionen *RGB* und *QBColor* zur Verfügung.

Aktivierte Steuerelemente farbig hinterlegen



Hinweis

Optionsfelder, Kontrollkästchen und Umschaltflächen

Kontrollkästchen haben die gleiche Aufgabe wie Textfelder – nämlich die Präsentation von Daten. Im Gegensatz zu einem Textfeld können in Kontrollkästchen allerdings keine Texte eingegeben werden. Stattdessen ist es möglich, ein Kontrollkästchen – z. B. durch einen Mausklick – anzukreuzen.

Kontrollkästchen

Der Zustand eines Kontrollkästchens wird durch die beiden logischen Werte *True* und *False* widerspiegelt. Zusätzlich zu diesen beiden Werten können Kontrollkästchen allerdings noch einen dritten Zustand annehmen, der dem Wert *NULL* entspricht. Der Zustand des Kontrollkästchens ist in diesem Fall unbestimmt, was durch eine graue Hinterlegung des Steuerelements gekennzeichnet wird.



Die Wirkung dieser drei Zustände können Sie anhand des Formulars *Kontrollkästchentest1* begutachten, das Sie in der Beispieldatenbank zu diesem Kapitel finden. Auf dem Formular befindet sich ein ungebundenes Kontrollkästchen mit dem Namen *chkTest1*. Außerdem besitzt das Formular drei Schaltflächen mit den Beschriftungen *True*, *False* und *NULL*, mit denen Sie das Kontrollkästchen in den betreffenden Zustand versetzen können.

Abb. 7.1

Das Formular *Kontrollkästchentest1*



Die Zuweisung der drei Zustände erfolgt mittels dreier Ereignisprozeduren, die an die drei Schaltflächen *True*, *False* und *NULL* gebunden sind. Die *Click*-Ereignisprozedur der Schaltfläche *True* hat z. B. den folgenden Aufbau:

```
Sub btnTrue_Click ()

    Me!chkTest1 = True

End Sub
```

Die eigentliche Zuweisung wird über den Ausdruck *Me!chkTest1 = True* erledigt. Stattdessen hätte auch der vollständige Ausdruck unter Zuhilfenahme der *Value*-Eigenschaft angegeben werden können:

```
Me!chkTest1.Value = True
```

Wie Sie aber bereits im *Kapitel 4 Programmieren mit Objekten* erfahren haben, sind die beiden Ausdrücke in Bezug auf den Verweis auf Steuerelementinhalte vollkommen gleichwertig.

Dreifacher Status

Obwohl das Steuerelement drei Zustände – nämlich *True*, *False* und *NULL* optisch unterschiedlich darstellt, können Sie normalerweise durch direktes Anklicken nur zwischen den beiden Zuständen *True* und *False* wechseln. Wenn Sie dagegen – wie in dem Beispielformular – die Eigenschaft *Dreifacher Status* des Kontrollkästchens auf *Ja* einstellen, aktivieren Sie durch wiederholtes Anklicken nacheinander die Zustände *True*, *False* und *NULL*.



Ereignisse von Kontrollkästchen

Die Ereignisseigenschaften von Kontrollkästchen sind fast vollständig identisch mit denen von Textfeldern. Sie haben also auch bei Kontrollkästchen die Möglichkeit zur Kontrolle des Fokus oder des aktuellen Inhalts bzw. Zustands. Eine einzige Ausnahme stellt die *BeiÄnderung*-Ereigniseigenschaft dar, die bei Kontrollkästchen nicht verfügbar ist.

Um auf eine Änderung des Zustands eines Kontrollkästchens zu reagieren, die durch den Benutzer verursacht wurde, löst Access das Ereignis *Click* aus. Wichtig ist hierbei, dass Access das Ereignis nicht nur beim Anklicken auslöst, sondern auch dann, wenn der Benutzer den Zustand des Kontrollkästchens über die Tastatur ändert. Dies kann beispielsweise durch Drücken der ☐-Taste erfolgen.

Eine Ausnahme stellen Änderungen dar, die Sie vom Programmcode aus vornehmen. In dem vorher vorgestellten Formular *Kontrollkästchentest1* war dies beispielsweise der Fall. Durch einen Klick auf die Schaltfläche *True* wird die Anweisung *Melchk_Test1 = True* ausgeführt und das Kontrollkästchen dadurch angekreuzt. Access löst bei diesem Vorgang *nicht* das *Click*-Ereignis aus. Dies ist nicht nur für Kontrollkästchen, sondern für alle Access-Steuerelemente typisch.

Um auf das *Click*-Ereignis zu reagieren, verwenden Sie die *BeimKlicken*-Ereigniseigenschaft des Kontrollkästchens. In dem Beispielformular *Kontrollkästchentest2* finden Sie eine entsprechende Ereignisprozedur, die bei einer Zustandsänderung des Kontrollkästchens *chk_Test1* den Text *True*, *False* oder *NULL* in einem Textfeld mit dem Namen *txt_Zustand* ausgibt:

```
Private Sub chkTest1_Click()

    Select Case Me!chkTest1

        Case True: Me!txtZustand = "True"
        Case False: Me!txtZustand = "False"
        Case Else: Me!txtZustand = "Null"

    End Select

End Sub
```

Click-Ereignis

BeimKlicken-
Ereigniseigenschaft



Abb. 7.2

Das Formular *Kontrollkästchentest2*



Die Click-Ereignis-
prozedur der
Schaltfläche

Hinweis



Wenn Sie das Formular öffnen, besitzt das Kontrollkästchen zunächst den Inhalt *NULL*. Sobald Sie es nun mit der Maus anklicken, wird der neue Zustand als Text in das Feld *txtZustand* geschrieben.

Kontrollkästchen, die an kein Datenfeld gebunden sind, besitzen standardmäßig den Zustand *NULL*. Mithilfe der *Standardwert*-Eigenschaft können Sie allerdings einen Zustand festlegen, der direkt beim Öffnen des Formulars aktiv ist. Dazu tragen Sie beim Formularentwurf einfach den Text *Wahr* (entspricht *True*) oder *Falsch* (entspricht *False*) ein.

Umschaltflächen

Umschaltflächen haben exakt die gleiche Aufgabe wie Kontrollkästchen. Auch sie können die beiden Zustände *True* und *False* repräsentieren. Eine Umschaltfläche verhält sich wie eine Schaltfläche, die niedergedrückt werden kann und diesen Zustand beibehält. Hierbei besitzt die Umschaltfläche den Wert *True*. Ist die Schaltfläche dagegen nicht niedergedrückt, besitzt Sie den Zustand *False*.

Wenn die Eigenschaft *Dreifacher Status* der Umschaltfläche aktiviert ist, kann eine Umschaltfläche drei Zustände – nämlich *True*, *False* und *NULL* – repräsentieren. Der Zustand *NULL* wird durch eine graue Einfärbung der Schaltfläche angedeutet.

Das Beispielformular *Umschaltflächentest1* eignet sich gut dazu, mit den drei Zuständen zu experimentieren. Das Formular ist eine Kombination aus den beiden vorhergehenden Beispielformularen. Anstelle eines Kontrollkästchens wird allerdings eine Umschaltfläche mit dem Namen *tbtTest1* verwendet.

Abb. 7.3

Das Formular
Umschaltflächentest1



Optionsfelder

Optionsfelder unterscheidet zunächst nichts von Kontrollkästchen oder Umschaltflächen. Auch sie können die drei Zustände *True*, *False* und *NULL* besitzen.

Optionsgruppen

Dies ändert sich allerdings, sobald Optionsfelder innerhalb von Steuerelementen des Typs *Optionsgruppe* platziert werden. In diesem Fall fehlt nämlich die Eigenschaft *BeimKlicken*, so dass Sie auf eine Änderung des Zustands nicht mehr reagieren können. Auch der Versuch, einem Optionsfeld innerhalb einer Optionsgruppe einen der Werte *True*, *False* oder *NULL* zuzuweisen, führt zu einem Laufzeitfehler.



Der Grund dafür liegt darin, dass die Optionsfelder einer Optionsgruppe nicht mehr wie eigenständige Steuerelemente auftreten, sondern nun eine Komponente der Optionsgruppe darstellen. Innerhalb einer Optionsgruppe können mehrere Optionsfelder angeordnet werden, wobei immer nur eine bestimmte Option markiert sein darf.

Jedes Optionsfeld hat einen eindeutigen Zahlenwert, der mithilfe der *Optionswert*-Eigenschaft festgelegt wird. Wenn Sie beispielsweise in einer Optionsgruppe vier Optionsfelder mit den Bezeichnungen *Rot*, *Grün*, *Gelb* und *Blau* anordnen, können Sie den einzelnen Optionsfeldern die Werte 1, 2, 3 und 4 zuordnen. Der Wert 1 steht in Zukunft also für das erste Optionsfeld und damit die Farbe Rot. Sie finden eine entsprechende Optionsgruppe mit dem Namen *ogrFarben* im Beispielformular *Optionsfeldtest1*:

Die *Optionswert*-Eigenschaft

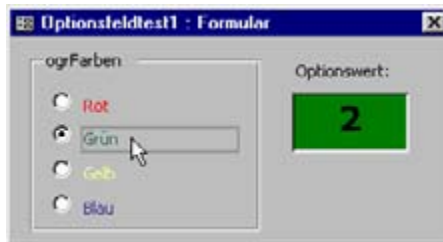


Abb. 7.4

Das Beispielformular *Optionsfeldtest1*

Wenn ein Optionsfeld der Gruppe durch den Anwender markiert wird, sorgt Access automatisch dafür, dass die Markierung des vorherigen Optionsfeldes gelöscht wird. So ist sichergestellt, dass immer nur ein Optionsfeld der Gruppe zu einer bestimmten Zeit markiert ist.

Um nun herauszufinden, welches Optionsfeld markiert ist, muss nicht auf die einzelnen Optionsfelder, sondern die Optionsgruppe verwiesen werden. Der Zugriff auf die Optionsgruppe hat immer den *Optionswert* des markierten Optionsfeldes zum Ergebnis

Direkt nach dem Öffnen des Beispielformulars *Optionsfeldtest1* ist zunächst einmal keines der vier Optionsfelder markiert. Der Verweis auf die Optionsgruppe hat daher den Wert *NULL* zum Ergebnis. Wenn in dem Beispielformular die Farbe *Grün* markiert wird, führt der Verweis

Me!ogrFarben

zu dem Ergebnis 2.

Um dem Entwickler die Gelegenheit zu geben, auf eine Änderung der Optionsgruppe zu reagieren, löst Access das Ereignis *Click* aus. Genau wie bei Kontrollkästchen oder Umschaltflächen ist es hierbei ebenfalls unerheblich, ob der Zustand der Optionsgruppe vom Anwender durch einen Mausklick oder über die Tastatur verändert wird.





In dem Beispielformular *Optionsfeldtest1* wird die folgende *Click*-Ereignisprozedur verwendet, um auf die Auswahl einer Farbe zu reagieren. Die Prozedur trägt den Optionswert der ausgewählten Farbe in das Textfeld *txtOptionswert* ein und ändert zusätzlich die Hintergrundfarbe des Textfeldes:

```
Private Sub ogrFarben_Click()  
  
    Me!txtOptionswert = Me!ogrFarben  
  
    Select Case Me!ogrFarben  
  
        Case 1: Me!txtOptionswert.BackColor = QBColor(12)  
        Case 2: Me!txtOptionswert.BackColor = QBColor(2)  
        Case 3: Me!txtOptionswert.BackColor = QBColor(14)  
        Case 4: Me!txtOptionswert.BackColor = QBColor(9)  
  
    End Select  
  
End Sub
```

Wird vom Anwender beispielsweise die Farbe *Rot* ausgewählt, hat die Optionsgruppe den Wert 1. Diesen Wert liest die Prozedur aus der Optionsgruppe aus und schreibt ihn in das Textfeld. Anschließend wird dem Textfeld mittels einer *Select-Case*-Verzweigung die passende Hintergrundfarbe zugewiesen.

Die *QBColor*-
Funktion

In der Prozedur wird eine Funktion mit dem Namen *QBColor* verwendet. Diese Funktion ermittelt zu einer bestimmten Farbe den Farbcode, der dann beispielsweise in die *Hintergrundfarbe*-Eigenschaft eingetragen werden kann:

```
QBColor (<Farbe>)
```

Die gewünschte Farbe muss hierbei gemäß folgender Tabelle als Parameter *<Farbe>* angegeben werden:

Werte für den Parameter <i>Farbe</i>			
Einstellung	Farbe	Einstellung	Farbe
0	Schwarz	8	Grau
1	Blau	9	Hellblau
2	Grün	10	Hellgrün
3	Cyan	11	Hellcyan
4	Rot	12	Hellrot

Werte für den Parameter *Farbe* (Forts.)

Einstellung	Farbe	Einstellung	Farbe
5	Magenta	13	Hellmagenta
6	Gelb	14	Hellgelb
7	Weiß	15	Leuchtend Weiß

Tabelle 7.2

Der Funktionsaufruf

`QBColor(9)`

ermittelt beispielsweise den Farbcode *16711680*, den Access als Hellblau interpretiert.

Optionsgruppen, die an kein Datenfeld gebunden sind, haben standardmäßig den Inhalt *NULL*. Dies bedeutet, dass kein Optionsfeld der Gruppe markiert ist. Wenn Sie allerdings von vornherein immer ein bestimmtes Optionsfeld markieren möchten, weisen Sie den entsprechenden Optionswert einfach der *Standardwert*-Eigenschaft der Optionsgruppe zu.



Hinweis

Zustandsänderungen von Optionsgruppen

Auf die gleiche Weise, wie Sie den aktuellen Optionswert einer Optionsgruppe auslesen, können Sie diesen natürlich auch von VBA aus verändern. Wenn Sie beispielsweise innerhalb der Optionsgruppe *ogrFarben* die Farbe *Grün* auswählen möchten, weisen Sie der Optionsgruppe einfach den Wert 2 zu:

```
Me!ogrFarben = 2
```

Das Markieren des passenden Optionsfeldes wird automatisch von Access erledigt.

Kontrollkästchen und Umschaltflächen in Optionsgruppen

Neben Optionsfeldern können auch Kontrollkästchen innerhalb von Optionsgruppen eingesetzt werden. In diesem Fall verhalten sich Kontrollkästchen genauso wie Optionsfelder. Da diese Anordnung für Windows-Anwendungen untypisch ist und viele Anwender verwirren dürfte, sollten Sie darauf möglichst verzichten.

In der Praxis werden aber häufig Umschaltflächen innerhalb von Optionsgruppen eingesetzt, die sich in dieser Anordnung ebenfalls wie Optionsfelder verhalten. Der Vorteil von Umschaltflächen gegenüber Optionsfeldern besteht darin, dass sie eine Beschriftung oder sogar ein Symbol besitzen können. Bei Optionsfeldern muss dagegen zur Kommentierung auf Bezeichnungsfelder zurückgegriffen werden.



In der Beispieldatenbank finden Sie ein Formular mit dem Namen *Umschaltflächen-test2*, das die Änderung der Schriftart eines Textfeldes ermöglicht.

Abb. 75

Das Formular *Umschaltflächen-test2*



Das Ändern der Schriftart wird mithilfe der folgenden Ereignisprozedur erledigt, die mit dem *Click*-Ereignis der Optionsgruppe verknüpft ist:

```
Private Sub ogrSchriftart_Click()

    Select Case ogrSchriftart

        Case 1: txtText.FontName = "Arial"
        Case 2: txtText.FontName = "Times New Roman"
        Case 3: txtText.FontName = "Courier"

    End Select

End Sub
```

Hinweis



In dem Beispielformular *Umschaltflächentest2* wurde die *Rahmenart*-Eigenschaft der Optionsgruppe auf *Transparent* eingestellt, so dass nur die Umschaltflächen für den Anwender sichtbar sind.

Befehlsschaltflächen

Den Umgang mit Befehlsschaltern haben Sie bereits in den ersten beiden Kapiteln kennen gelernt, so dass hier nur kurz darauf eingegangen wird.

Schaltflächen besitzen fast identische Ereignisseigenschaften wie Textfelder. So können Sie auch bei Befehlsschaltflächen den Fokus überwachen oder Tastenbetätigungen registrieren. Was fehlt, sind lediglich die Ereignisseigenschaften, die mit dem Speichern von Daten zusammenhängen. Der Grund dafür liegt darin, dass Befehlsschaltflächen nicht an ein Datenfeld gebunden werden können.

Das *Click*-Ereignis

Das wichtigste Ereignis ist das *Click*-Ereignis, das Access bei einem Klick auf die Schaltfläche auslöst. Als Reaktion auf dieses Ereignis führt Access die *Click*-Ereignisprozedur aus, die Sie mit den gewünschten Anweisungen füllen können.



Neben dem *Click*-Ereignis stellen Schaltflächen zusätzlich das *DblClick*-Ereignis zur Verfügung, das Access bei einem Doppelklick auslöst. Bei der Anwendung dieses Ereignisses sollten Sie beachten, dass Access bei einem Doppelklick auf eine Schaltfläche zusätzlich zum *DblClick*-Ereignis immer auch das *Click*-Ereignis auslöst.

Eine Besonderheit von Schaltflächen ist die Möglichkeit, diese auszublenden, ohne ihre Funktion hierdurch zu beeinträchtigen. Dies erledigen Sie, indem Sie die *Transparent*-Eigenschaft der Schaltfläche auf *Ja* einstellen. Sie könnten nun beispielsweise mehrere dieser unsichtbaren Schaltflächen auf einer Grafik anordnen, um dem Anwender so die Möglichkeit zu geben, bestimmte Bildbereiche anzuklicken.

Listen- und Kombinationsfelder

Listenfelder gehören mit zu den kompliziertesten Steuerelementen, die Access zur Verfügung stellt. Dies liegt daran, dass Listenfelder nicht nur einen einzelnen Feldinhalt repräsentieren, sondern mehrere Datensätze einer Tabelle oder Abfrage enthalten. Nicht weniger kompliziert sind die so genannten *Kombinationsfelder*, die eine Kombination aus Listen- und Textfeld darstellen. Im Folgenden wird zunächst der Umgang mit Listenfeldern erläutert.

Listenfelder haben die Aufgabe, mehrere Einträge in Form einer Liste anzuzeigen. In der Regel stammen diese Einträge direkt aus einer Tabelle oder Abfrage. Zu diesem Zweck kann unter der *Datensatzherkunft*-Eigenschaft der Name einer Tabelle oder einer Abfrage angegeben werden. Alternativ ist es auch möglich, direkt einen SQL-Text einzugeben. Sobald nun das Formular, auf dem sich das Listenfeld befindet, in die Formularansicht umgeschaltet wird, füllt Access das Listenfeld mit Daten.

○ —————
Listenfelder

Ein Listenfeld kann nicht nur eine, sondern bei Bedarf auch mehrere Spalten haben. Die Anzahl dieser Spalten wird hierbei mithilfe der *Spaltenanzahl*-Eigenschaft festgelegt. Sie sollten darauf achten, dass die hier angegebene Spaltenanzahl mit der Spaltenanzahl der Datenquelle übereinstimmt.

○ —————
Mehrspaltige
Listenfelder

Eine weitere wichtige Eigenschaft trägt den Namen *Gebundene Spalte*. Über diese Eigenschaft legen Sie fest, welchen Spalteninhalt ein Verweis auf das Listenfeld zum Ergebnis hat. Sollte das Listenfeld an ein Datenfeld gebunden sein, bestimmt die Einstellung der Eigenschaft *Gebundene Spalte* außerdem, welchen Feldinhalt Access in das Datenfeld einträgt.

Angenommen, Sie verfügen über ein Listenfeld mit dem Namen *IstKundenliste*, in dem alle Kunden aufgelistet sind. Jeder Kunde wird hier mit seinem Namen (Spalte 1) und der Kundennummer (Spalte 2) aufgeführt. Durch einen Klick auf den ersten Listeneintrag wählen Sie den Kunden mit dem Namen *Müller* und der *KundenNr* 3 aus. Wenn die Eigenschaft *Gebundene Spalte* den Wert 1 hat, führt der Verweis auf das Feld zum Ergebnis *Müller*. Ist das Listenfeld dagegen an die Spalte 2 gebunden, hat es den Inhalt 3.





Sie haben ein solches Listenfeld bereits im vorhergehenden Kapitel im Zusammenhang mit dem Formular *Navigator* kennen gelernt, das Sie auch in der Beispieldatenbank zu diesem Kapitel finden.

Abb. 7.6

Das Formular *Navigator*



Das Formular enthält eine Liste aller Kunden, wobei diese mithilfe der folgenden SQL-Abfrage aus der Tabelle *Kunden* abgerufen wird:

```
SELECT Name, KundenNr FROM Kunden ORDER BY Name;
```

Da die Abfrage zwei Datenfelder zum Ergebnis hat, ist die *Spaltenanzahl*-Eigenschaft dementsprechend auf den Wert 2 eingestellt. Die Eigenschaft *Gebundene Spalte* hat ebenfalls den Wert 2, so dass der Verweis

```
Forms!Kunden!lstKunden
```

immer die Nummer des momentan ausgewählten Kunden zum Ergebnis hat. Eine Besonderheit ist hierbei, dass die Spaltenbreite dieser zweiten Spalte die Breite 0 hat. Die Spalte ist damit für den Anwender nicht sichtbar.

Hinweis



Die Breite der einzelnen Ausgabespalten einer Liste wird mithilfe der *Spaltenbreiten*-Eigenschaft festgelegt.

Auswahl von Listeneinträgen

Sobald das Formular *Navigator* in der Formularansicht geöffnet wird, führt Access die SQL-Abfrage aus und füllt die Liste mit den einzelnen Einträgen. Da es sich in dem Beispiel um ein ungebundenes Listenfeld handelt, das an kein Datenfeld gebunden ist, ist zu diesem Zeitpunkt noch kein Listeneintrag ausgewählt. Ein Verweis auf das Listenfeld führt damit zu dem Ergebnis *NULL*.

Sobald der Anwender nun einen Listeneintrag auswählt, löst Access das *Click*-Ereignis aus. Hierbei ist es unerheblich, ob die Auswahl durch einen Klick auf den Eintrag oder durch Verschieben der Markierung mithilfe der Pfeiltasten erfolgt. Zusätzlich



werden außerdem die Ereignisse *BeforeUpdate* und *AfterUpdate* ausgelöst, die Sie bereits von Textfeldern kennen. Im Prinzip ist es unerheblich, über welches dieser drei Ereignisse Sie sich von der Auswahl eines Listeneintrags informieren lassen. In der Praxis erweist sich allerdings das *Click*-Ereignis als am unproblematischsten.

Neben dem *Click*-Ereignis stellen Listenfelder das *DblClick*-Ereignis zur Verfügung, das Access bei einem Doppelklick auf einen Eintrag auslöst. Hierauf können Sie mithilfe der *BeimDoppelklicken*-Ereigniseigenschaft reagieren. Im Beispielformular *Navigator* wird dieses Ereignis dazu genutzt, das Kundenformular mit dem passenden Kunden zu öffnen. Die Ereignisprozedur hat folgenden Aufbau:

BeimDoppelklicken-
Ereigniseigenschaft

```
Private Sub lst_Kundenliste_DblClick(Cancel As Integer)

Dim KundenNr As Variant

    KundenNr = Me!lst_Kundenliste

    DoCmd.OpenForm "Kunden"

    FormularDatensatzAuswählen Forms!Kunden, _
        "KundenNr = " & KundenNr

End Sub
```

Sobald ein Doppelklick auf einen Listeneintrag erfolgt, ermittelt die Prozedur die Kundennummer des momentan ausgewählten Kunden und legt diese in der Variablen *KundenNr* ab. Anschließend wird das Kundenformular geöffnet und schließlich auf den passenden Kunden positioniert.

Eine weitere Verbesserung des Navigators ergibt sich, wenn nicht nur bei einem Doppelklick, sondern bei jeder Auswahl eines Listeneintrags auf den passenden Kunden positioniert wird. Die Positionierung soll in diesem Fall aber nur dann erfolgen, wenn das Kundenformular bereits geöffnet ist. Genau diese Aufgabe erledigt die folgende Ereignisprozedur, die an das *Click*-Ereignis der Liste gebunden ist:

Die Click-Ereignis-
prozedur

```
Private Sub lst_Kundenliste_Click()

Dim KundenNr As Variant

    KundenNr = Me!lstKundenliste

    If IstGeöffnet("Kunden") Then

        FormularDatensatzAuswählen Forms!Kunden, _
            "KundenNr = " & KundenNr

    End If

End Sub
```

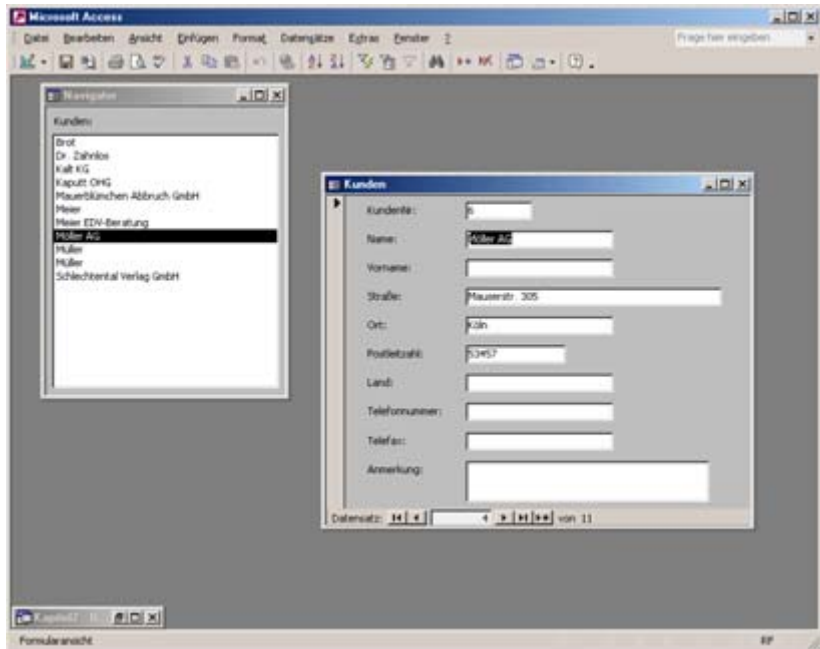


Navigator testen

Um einen Test des *Navigators* vorzunehmen, öffnen Sie das Formular in der Formularansicht. Wenn Sie doppelt auf einen Kunden klicken, öffnet Access das Kundenformular und zeigt den Kunden an. Von nun an genügt ein einfacher Klick auf einen Kunden, um diesen im Kundenformular anzeigen zu lassen.

Abb. 7.7

Der Navigator und das Formular Kunden



Hinweis



In Bezug auf die Ereignisseigenschaften besitzen Listenfelder genau die gleichen Ereignisseigenschaften wie Textfelder. Eine Ausnahme bildet lediglich die *BeiÄnderung*-Ereigniseigenschaft, die bei Textfeldern zur Steuerung der Texteingabe dient. Da bei Listenfeldern allerdings keine Eingabe von Texten möglich ist, fehlt diese Ereignisseigenschaft.

Zugriff auf Spalten

Column-Eigenschaft

Oft ist es erforderlich, nicht auf eine einzelne Spalte eines Listenfeldes, sondern auf mehrere oder sogar alle Spalten zuzugreifen. Da sich mithilfe der Eigenschaft *Gebundene Spalte* allerdings immer nur eine Spalte an das Listenfeld binden lässt, ist dies nicht so ohne weiteres möglich. Aus diesem Grund stellt Access bei Listenfeldern die Eigenschaft *Column* zur Verfügung, mit der sich gezielt auf den Inhalt einer Spalte zugreifen lässt. Der Zugriff erfolgt hierbei über die Spaltennummer:

```
<Listenfeld-Objekt>.Column(<Spaltennummer>)
```



Im Gegensatz zur Eigenschaft *Gebundene Spalte* hat die erste Spalte allerdings nicht die Position 1, sondern die Position 0. Außerdem müssen Sie darauf achten, dass Sie unter *<Spaltenposition>* immer einen Wert angeben, der unter dem Wert der *Spaltenanzahl*-Eigenschaft liegt. Ein Verweis auf eine Spalte oberhalb des einge-tragenen Wertes führt immer zum Ergebnis *NULL*.

Das Ergebnis der *Column*-Eigenschaft hat den Datentyp *Variant*. Sollte in dem Lis-tenfeld noch kein Eintrag markiert sein, gibt die *Column*-Eigenschaft das Ergebnis *NULL* zurück!



Hinweis

In der Beispieldatenbank finden Sie ein Formular mit dem Namen *Kundenauswahl*. Dieses Formular ermöglicht die Auswahl eines einzelnen Kunden wobei der Ort und die Postleitzahl in einem Textfeld unterhalb der Liste angezeigt werden. Das Textfeld trägt den Namen *txtWohnort*.



Abb. 7.8

Das Formular
Kundenauswahl

Die Datenquelle der Liste wird wieder mithilfe einer SQL-Abfrage definiert, die fol-genden Aufbau hat:

Datenquelle

```
SELECT Name,Postleitzahl,Ort FROM Kunden ORDER BY Name;
```

Die *Spaltenanzahl*-Eigenschaft hat dementsprechend den Wert 3. Damit innerhalb der Liste nur die erste Spalte – also der Inhalt des Feldes *Name* – sichtbar ist, wird den Spalten *Postleitzahl* und *Ort* mithilfe der *Spaltenbreiten*-Eigenschaft die Breite 0 zugewiesen:

```
7 cm;0 cm;0 cm
```

Immer dann, wenn ein neuer Listeneintrag ausgewählt wird, müssen in das Text-feld *txtWohnort* die aktuellen Daten eingetragen werden. Dies erledigt die folgende Ereignisprozedur, die mit dem *Click*-Ereignis der Liste verknüpft ist:

Die *Click*-Ereignis-
prozedur





```
Private Sub lstKundenliste_Click()

Dim Wohnort As Variant

    Wohnort = Me!lstKundenliste.Column(1) & ", " & _
        Me!lstKundenliste.Column(2)

    Me!txtWohnort = Wohnort

End Sub
```

Mithilfe der *Column*-Eigenschaft wird die Postleitzahl sowie der Wohnort des momentan markierten Eintrags abgerufen und zu einer Zeichenkette zusammengestellt. Diese wird anschließend im Textfeld *txtWohnort* abgelegt.

Hinweis



Wenn Sie die Daten eines Listenfeldes aktualisieren möchten, wenden Sie genau wie bei Formularen die *Requery*-Methode an:

```
<Listenfeld-Objekt>.Requery
```

Dies ist beispielsweise erforderlich, wenn sich die Datenquelle, die dem Listenfeld zugrunde liegt, nach dem Öffnen des Formulars oder der letzten Aktualisierung geändert hat.

Listenfelder mit Wertelisten füllen

Für den Programmierer tritt oft das Problem auf, dass ein Listenfeld mit einigen Werten gefüllt werden muss, die allerdings nicht in einer Tabelle, sondern beispielsweise in einem Datenfeld (Array) vorliegen. Um solche Daten in einem Listenfeld zu präsentieren, könnte man eine Tabelle anlegen und diese dann als Datenquelle des Listenfeldes definieren. Dieser Vorgang ist allerdings relativ aufwändig und langsam. Wenn es sich nur um wenige Listeneinträge handelt, führt ein kleiner Trick zu einem schnelleren Ergebnis.

Herkunftstyp



Alle Listenfelder besitzen eine Eigenschaft mit dem Namen *Herkunftstyp*, über die sich bestimmen lässt, ob die Daten aus einer Tabelle bzw. Abfrage oder einer *Werteliste* stammen.

Eine Werteliste ist nichts anderes als eine Zeichenkette, in der mehrere Textelemente durch Semikolons voneinander getrennt angegeben werden. Die folgende Werteliste hat beispielsweise drei Elemente:

```
Rot;Gelb;Grün
```

Wenn Sie ein neues Listenfeld anlegen, unter *Herkunftstyp* den Eintrag *Werteliste* auswählen und schließlich als *Datensatzherkunft* die obige Zeichenkette eintragen, zeigt Access beim Umschalten in die Formularansicht die drei Listeneinträge *Rot*, *Gelb* und *Grün* an.



Auch mehrspaltige Listenfelder lassen sich mithilfe von Wertelisten füllen. Wenn Ihre Liste beispielsweise zwei Spalten haben soll, geben Sie als Werteliste abwechselnd ein Element der ersten und ein Element der zweiten Spalte an. Zusätzlich müssen Sie die *Spaltenanzahl*-Eigenschaft auf den Wert 2 einstellen.

Mehrspaltige
Listenfelder

Die folgende Werteliste füllt das Listenfeld mit drei Einträgen. Jedem Eintrag ist hierbei eine Zahl zugeordnet, die in der zweiten Spalte erscheint:

```
Rot;1;Gelb;2;Grün;3
```

Listeneinträge, die Sonderzeichen oder das Semikolon enthalten, müssen in einfachen Anführungszeichen (") angegeben werden!



Hinweis

Um nun eine Liste von VBA aus zu füllen, muss lediglich eine Werteliste zusammengestellt werden, die anschließend in die *Datensatzherkunft*-Eigenschaft (Englisch: *RowSource*-Eigenschaft) des Listenfeldes eingetragen wird. Damit Access von dieser Änderung erfährt, muss außerdem die *Requery*-Methode auf das Listenfeld angewendet werden.

Ein Beispiel ist die Anzeige aller Dateien des aktuellen Verzeichnisses innerhalb einer Liste. Die Abfrage des aktuellen Verzeichnisses erfolgt mithilfe der Funktion *Dir*, die die folgende Syntax besitzt:

Dateiverzeichnis

```
Dir [(<Suchmaske>)]
```

Unter *<Suchmaske>* kann das gleiche Suchkriterium angegeben werden, wie bei dem DOS-Befehl *Dir*. Um beispielsweise nach allen Dateien zu suchen, die die Dateierendung *.mdb* besitzen, geben Sie einfach **.mdb* an. Die Suchmaske *** ermittelt dagegen alle Dateien. Das Ergebnis der Funktion ist eine Zeichenkette, die den gefundenen Dateinamen enthält.

Ein Verzeichnis kann mehrere Dateien enthalten, die der angegebenen *<Suchmaske>* entsprechen. Beim Aufruf der *Dir*-Funktion gibt diese aber immer nur den ersten gefundenen Dateinamen zurück. Um trotzdem eine vollständige Liste zu erhalten, wird die Funktion zunächst ein erstes Mal zusammen mit einer Suchmaske angegeben. Die Funktion wird damit initialisiert und gibt den ersten gefundenen Dateinamen aus. Im Folgenden muss die Funktion nun wiederholt aufgerufen werden, um jeweils weitere Dateinamen zu ermitteln. Bei diesen Wiederholungen darf allerdings keine Suchmaske angegeben werden. Erst dann, wenn kein passender Dateiname mehr ausfindig gemacht werden konnte, gibt die Funktion eine leere Zeichenkette zurück.

Die folgende Prozedur nutzt die *Dir*-Funktion, um eine Werteliste aller Dateien des aktuellen Verzeichnisses anzulegen. Sie finden die Prozedur im Modul des Formulars *Dateiliste1*. Der Start der Prozedur erfolgt in der *Open*-Ereignisprozedur des Formulars. So ist sichergestellt, dass die Liste bereits beim Öffnen mit Einträgen gefüllt wird.

Dateiliste erstellen





```
Private Sub ListeFüllen()

Dim Werteliste As Variant
Dim Dateiname As String

    "Initialisieren der Dir-Funktion
    "mittels einer Suchmaske

    Dateiname = Dir("*. *")

    Do While Dateiname <> ""

        "Die Werteliste wird um den Dateinamen ergänzt

        Werteliste = Werteliste & "" & Dateiname & ""

        "Wiederholen der Suche nach einem weiteren Datenamen

        Dateiname = Dir

        "Sollte noch eine Datei folgen,
        "wird der Werteliste ein
        "Semikolon als Trennzeichen angehängt

        If Dateiname <> "" Then Werteliste = Werteliste _
            & ";"

    Loop

    "Die Werteliste wird in die
    "Datensatzherkunft-Eigenschaft
    "der Liste geschrieben. Anschließend erfolgt eine
    "Aktualisierung der Liste

    Me!lstDateiliste.RowSource = Werteliste
    Me!lstDateiliste.Requery

End Sub
```





Abb. 7.9

Das Formular *Dateiliste1*

Mithilfe der *Dir*-Funktion können auch Verzeichnisse ausfindig gemacht werden. Die Funktion hat dazu einen zweiten Parameter, über den Sie die entsprechenden Einstellungen vornehmen können. Die beiden Anweisungen *CurDir* und *ChDrive* erlauben es, das aktuelle Verzeichnis bzw. das aktuelle Laufwerk zu wechseln.



Hinweis

Listenfelder mit einer Callback-Funktion füllen

Zum Füllen umfangreicher Listen ist die Verwendung einer Werteliste nicht zu empfehlen, da das Zusammenstellen der einzelnen Listeneinträge zeitaufwändig ist. Eine weitere Beschränkung ist, dass Wertelisten eine Zeichenanzahl von 2.048 Zeichen nicht überschreiten dürfen.

Aus diesen Gründen haben die Entwickler von Access die Möglichkeit vorgesehen, eine Liste mithilfe einer VBA-Funktion zu füllen. Zu diesem Zweck muss der Name einer Funktion, die Sie selbst geschrieben haben, unter der *Herkunftstyp*-Eigenschaft des Listenfeldes eingetragen werden. Es darf nur der Name der Funktion eingetragen werden. Die Angabe eines Gleichheitszeichens, einer Klammer oder einer Parameterliste führt zu einem Fehler!



**Funktion als
Herkunftstyp**

Der Kopf dieser Funktion muss folgenden Aufbau haben:

```
Function <Prozedurname> (<St> As Control, <ID>,
<Zeile>, <Spalte>, <Code>) As Variant
```



Die angegebene Funktion wird von Access mehrmals hintereinander aufgerufen. Bei jedem Aufruf muss die Funktion eine bestimmte Information zurückliefern. Die Art der Information bestimmt das Argument *Code*:

Werte für den Parameter *Code*

<Code>	Erwarteter Rückgabewert
0	Initialisierung der Funktion. Die Funktion muss den Wert <i>True</i> zurückgeben, wenn die Initialisierung erfolgreich war, und <i>False</i> , wenn die Aufgabe nicht erfüllt werden kann.
1	Bei diesem Wert muss die Funktion einen eindeutigen Identifikationswert zurückgeben. Diese ID-Nummer wird in Zukunft als Parameter <ID> bei jedem Aufruf an die Funktion übergeben. Werden mehrere Steuerelemente über diese Prozedur gefüllt, kann so zwischen den Aufrufen verschiedener Steuerelemente unterschieden werden.
3	Die Anzahl der Zeilen, die in dem Listefeld angezeigt werden sollen.
4	Die Anzahl der Spalten, die in dem Listefeld angezeigt werden sollen.
5	Die Breite der Spalten in der Einheit Twips (1 cm = 567 Twips). Der Rückgabewert -1 bedeutet, dass Access die Vorgabe der <i>Spaltenbreiten</i> -Eigenschaft verwendet.
6	Ein Listeneintrag, der in dem Listefeld erscheinen soll. Welchen Eintrag Access erwartet, wird über die numerischen Argumente <Zeile> und <Spalte> mitgeteilt.
7	Eine Formatzeichenkette, die das Anzeigeformat jeweils für den Listeneintrag <Zeile> > <Spalte> bestimmt. Bei dem Rückgabewert <i>NULL</i> verwendet Access die Standardeinstellung.
9	Kein Rückgabewert. Dieser Aufruf folgt immer als Letztes, um beispielsweise offene Dateien zu schließen.

Tabelle 7.3

Code-Reihenfolge

Als Erstes ruft Access die Funktion mit den <Code>-Werten 0, 1, 3 und 4 auf und erwartet die entsprechenden Rückgabewerte. Danach wird die Funktion ein erstes Mal mit dem <Code> 5 aufgerufen, um die Breite des Listefeldes zu ermitteln, und ein zweites Mal, um die Breite der einzelnen Spalten zu bestimmen. Die Funktion wird jetzt mehrmals hintereinander mit den Codes 6 und 7 aufgerufen, um das Listefeld zu füllen. Jedes Mal übergibt Access die <Spalte> und die <Zeile>, deren Inhalt erwartet wird. Zum Abschluss wird die Funktion ein letztes Mal mit dem <Code> 9 aufgerufen.



Ein Beispielformular zur Anwendung dieser Funktion finden Sie in der Beispieldatenbank zu diesem Kapitel unter dem Namen *Dateiliste2*. In diesem Beispiel wird die Liste *IstKundenliste* wieder mit allen Dateien des aktuellen Verzeichnisses gefüllt.

Dateiliste über eine Funktion erstellen

Die Liste wird hierzu mithilfe der *Dir*-Funktion in ein eindimensionales, dynamisches Array eingelesen, das auf folgende Weise im Modul des Formulars *Dateiliste2* deklariert wurde:

```
Private Dateiliste() As String
```

Die Anzahl der Dateinamen innerhalb eines Arrays wird in der Variablen *DateilisteAnzahl* festgehalten, die ebenfalls auf Modulebene deklariert ist:

```
Private DateilisteAnzahl As Integer
```

Das Füllen dieses Arrays erfolgt mithilfe der folgenden Prozedur:

```
Private Sub DateiArrayFüllen()

Dim Dateiname As String
Dim i As Integer

    "Initialisieren der Dir-Funktion
    "mittels einer Suchmaske

    Dateiname = Dir("*. *")

    Do While Dateiname <> ""

        "Grenzen des Arrays erweitern
        ReDim Preserve Dateiliste(0 To i)

        "Der Dateiname wird dem Array hinzugefügt
        Dateiliste(i) = Dateiname

        "Wiederholen der Suche nach einem
        "weiteren Dateinamen
        Dateiname = Dir

        "i um den Wert 1 erhöhen
        i = i + 1

    Loop

    DateilisteAnzahl = i

End Sub
```





Damit das Array direkt beim Öffnen des Formulars gefüllt wird, erfolgt der Start der Funktion in der *Open*-Ereignisprozedur des Formulars *Dateiliste2*:

```
Private Sub Form_Open(Cancel As Integer)

    DateiArrayFüllen

End Sub
```

Die eigentliche Prozedur zum Füllen des Listenfeldes trägt den Namen *ListeFüllen*:

```
Private Function ListeFüllen(St As Control, ID, Zeile,
    Spalte, Code) As Variant

    Select Case Code

        "Initialisieren
    Case 0: ListeFüllen = True
        "Öffnen
    Case 1: ListeFüllen = 1
        "Zeilenanzahl
    Case 3: ListeFüllen = DateilisteAnzahl
        "Spaltenanzahl
    Case 4: ListeFüllen = 1
        "Spaltenbreite
    Case 5: ListeFüllen = -1
        "Listeneintrag
    Case 6: ListeFüllen = Dateiliste(Zeile)
        "Formatzeichenfolge
    Case 7: ListeFüllen = Null
    End Select

End Function
```

Der Name dieser Funktion muss nun nur noch unter der *Herkunftstyp*-Eigenschaft der Liste *IstDateiliste* eingetragen werden. Sobald das Formular nun geöffnet wird, füllt die Prozedur *DateiArrayFüllen* das Array mit den Dateinamen. Da dies im Zuge des *Open*-Ereignisses passiert und das Formular zu diesem Zeitpunkt noch nicht sichtbar ist, wird die Funktion *ListeFüllen* zunächst nicht aktiv. Erst nach Abarbeitung der *Open*-Ereignisprozedur fragt Access die einzelnen Listeneinträge von der Funktion *ListeFüllen* ab und trägt diese in die Liste ein.

Hinweis



Bei der Programmierung der Funktion zum Füllen der Liste sollten Sie berücksichtigen, dass Access die Listeneinträge in beliebiger Reihenfolge abfragt. Sie müssen also innerhalb der Prozedur bei jedem Aufruf auf alle Listeneinträge zugreifen können. Dies ist auch der Grund dafür, warum die Dateiliste zunächst in ein Datenfeld eingelesen wird. Auch während das Formular geöffnet ist, kommt es zu erneuten Anfragen an die Prozedur. Dies ist beispielsweise dann der Fall, wenn ein Listenfeld markiert oder die Bildlaufleiste der Liste betätigt wird!



Listenfelder mit Mehrfachauswahl

Mithilfe der *Mehrfachauswahl*-Eigenschaft kann ein Listenfeld so konfiguriert werden, dass mehrere Listeneinträge gleichzeitig markiert werden können. Es stehen folgende Einstellungen zur Verfügung:

Mögliche Einstellungen der *Mehrfachauswahl*-Eigenschaft

Einstellung	Beschreibung
Keine	Es kann nur ein einzelner Eintrag markiert sein.
<i>Einzeln</i>	Es können mehrere Einträge gleichzeitig markiert werden. Das Markieren (oder das Löschen einer Markierung) erfolgt hierbei durch einfaches Anklicken mit der Maus.
<i>Erweitert</i>	Es können ebenfalls mehrere Einträge gleichzeitig markiert werden. Beim Markieren einzelner Einträge muss jedoch die Strg -Taste gedrückt gehalten werden. Mithilfe der  -Taste lassen sich ganze Bereiche markieren.

Tabelle 7.4

Obwohl Listenfelder mit Mehrfachauswahl auch an Datenfelder gebunden werden können, sollten Sie solche Listenfelder stets ungebunden anlegen. Wenn mehrere Einträge markiert sind, kann Access nicht entscheiden, welche Daten in dem zugrunde liegenden Feld gespeichert werden sollen. Access trägt daher stets den Inhalt *NULL* ein.

Um herauszufinden, welche Einträge markiert sind, stellen Listenfelder eine Auflistung mit dem Namen *ItemsSelected* zur Verfügung. Diese Auflistung enthält Elemente des Datentyps *Variant*, wobei jedes Element die Positionsnummer (Index) eines markierten Listeneintrags repräsentiert. Das erste Element hat hierbei den Index 0. Wenn beispielsweise in einem Listenfeld der erste, der dritte und der achte Eintrag markiert sind, enthält die Auflistung drei Elemente mit den Werten 0, 2 und 7.

In vielen Fällen sind allerdings nicht die Positionsnummern, sondern die Inhalte der gebundenen Spalte der einzelnen Einträge von Interesse. Die Umsetzung erfolgt hierbei mithilfe der *ItemData*-Methode:

```
<Listenfeld>.ItemData(<Index>)
```

Die *ItemData*-Methode ermittelt zu einem bestimmten *<Index>* den Inhalt der gebundenen Spalte.



In der Beispieldatenbank finden Sie unter dem Namen *Kunden drucken* ein Formular, das den Ausdruck bestimmter Kunden erlaubt. Das Formular enthält eine Liste mit dem Namen *lstKunden*, bei der die *Mehrfachauswahl*-Eigenschaft auf *Einzeln* eingestellt ist. Der Liste liegt die Tabelle *Kunden* zugrunde, wobei mittels einer Abfrage die Felder *Name* und *KundenNr* ausgewählt werden. Die zweite Spalte ist die gebundene Spalte des Listenfeldes. Beim Verweis auf die Liste wird also eine Kundennummer zurückgegeben.

Abb. 7.10

Das Formular
Kunden drucken



Über eine Schaltfläche mit dem Namen *btnDrucken* können die ausgewählten Kunden gedruckt werden. Zu diesem Zweck befindet sich in der Beispieldatenbank ein Bericht mit dem Namen *Kundenliste*.

Die Ereignisprozedur der Schaltfläche hat folgenden Aufbau:

```
Private Sub btnDrucken_Click()

Dim Element As Variant
Dim Bedingung As String
Dim KundenNr As Variant

    "Ist überhaupt ein Eintrag markiert?
    If lstKunden.ItemsSelected.Count = 0 Then Exit Sub

    For Each Element In lstKunden.ItemsSelected

        KundenNr = lstKunden.ItemData(Element)

        Bedingung = Bedingung & _
        "KundenNr = " & KundenNr & " OR "

    Next Element
```





```
"Rechtes OR abschneiden
Bedingung = Left(Bedingung, Len(Bedingung) - 4)

DoCmd.OpenReport ReportName:="Kundenliste", _
    WhereCondition:=Bedingung, _
    View:=acPreview
```

End Sub

Als Erstes wird überprüft, ob überhaupt ein Eintrag markiert ist. Dies wird mithilfe der *Count*-Eigenschaft bewerkstelligt, die die Anzahl der Elemente innerhalb der *ItemsSelected*-Auflistung – also die Anzahl der markierten Einträge – angibt. Sollte kein Eintrag markiert sein, bricht die Prozedur ab.

Als Nächstes werden alle Einträge der Auflistung mittels einer *For Each*-Schleife durchlaufen. Zu jedem Element wird die *KundenNr* über die *ItemData*-Methode ermittelt. Anschließend wird diese Kundennummer zu einer Bedingung hinzugefügt. Wenn beispielsweise drei Kunden mit den Nummern 4, 5 und 8 ausgewählt sind, erstellt die Schleife folgende Bedingung:

```
KundenNr = 4 OR KundenNr = 5 OR KundenNr = 8 OR
```

Diese Bedingung trifft genau auf die drei markierten Kunden zu. Nach Abarbeitung der Schleife enthält die Bedingung am Ende ein *OR* zu viel, das mithilfe der *Left*-Anweisung abgeschnitten wird. Über die *Len*-Anweisung wird dazu die Länge der Bedingung ermittelt und um den Wert 4 verringert. Der Wert 4 ist erforderlich, da die Bedingung am Ende ein Leerzeichen, den Text *OR* und ein weiteres Leerzeichen – also insgesamt vier Zeichen – enthält. Der berechnete Wert wird anschließend zusammen mit der *Left*-Funktion eingesetzt, um die Bedingung entsprechend zu verkürzen.

Nachdem die Bedingung zusammengestellt ist, wird der Bericht *Kundenliste* über die *OpenReport*-Methode geöffnet und die Bedingung als Parameter *WhereCondition* übergeben.

Als *WhereCondition* darf eine Zeichenkette mit einer maximalen Länge von 32.768 Zeichen übergeben werden. Diese Größe dürfte auch umfangreichen Bedingungen genügen, wie sie bei sehr vielen markierten Kunden entstehen würden.



Hinweis

Kombinationsfelder

Wie bereits erwähnt, besteht ein Kombinationsfeld aus einer Liste, die zusätzlich ein Textfeld besitzt. Die Liste des Kombinationsfeldes wird in diesem Zusammenhang auch als *Dropdown-Liste* bezeichnet, da die Liste bei Bedarf heruntergeklappt werden kann. Der einzige Unterschied zu Listeneinträgen ist die Art und Weise, wie der momentan ausgewählte Listeneintrag optisch präsentiert wird. Bei Listeneinträgen wird dies durch eine Markierung des betreffenden Listeneintrags erledigt. Kombinationsfelder stellen dagegen den ausgewählten Listeneintrag in dem Textfeld dar. Ein weiterer Unterschied besteht darin, dass Kombinationsfelder keine Mehrfachauswahl zulassen.



BeiNichtInListe-
Ereigniseigenschaft

Zum Füllen der Dropdown-Liste stehen die gleichen Möglichkeiten zur Verfügung, die Sie bereits bei den Listefeldern kennen gelernt haben. Auch die Ereignisseigenschaften stimmen mit denen von Listefeldern überein. Einen kleinen aber wichtigen Unterschied stellt das Ereignis *NotInList* (*BeiNichtInListe*-Ereigniseigenschaft) dar, das nur bei Kombinationsfeldern verfügbar ist. Dieses Ereignis wird von Access ausgelöst, wenn in das Kombinationsfeld ein Text eingegeben wird, der nicht in der Dropdown-Liste enthalten ist. Als Reaktion auf dieses Ereignis können Sie den neu eingegebenen Text nun beispielsweise der Dropdown-Liste hinzufügen und den Eintrag so in Zukunft direkt verfügbar machen.

Hinweis



Damit Access das *NotInList*-Ereignis auslöst, muss die Eigenschaft *NurListeneinträge* des Listefeldes auf *Ja* eingestellt sein.

Ein Beispiel für die Anwendung des *NotInList*-Ereignisses finden Sie im Formular *Rechnungen* der Beispieldatenbank.

Abb. 7.11

Das Formular
Rechnungen

ArtikelNr.	Bemerkung	Anzahl	Einzelpreis	Gesamtpreis
1	Zahnbürste, einfach	2	20,00 DM	40,00 DM
2	Zahnbürste, luxus	4	33,30 DM	133,20 DM

Summe: 173,20 DM

Das Formular enthält ein Unterformular, in dem die einzelnen Positionen der Rechnung aufgeführt sind. Das Feld *ArtikelNr* dieses Unterformulars ist ein Kombinationsfeld, über das eine *Artikelbezeichnung* ausgewählt werden kann. Das Kombinationsfeld ist hierbei an die Spalte *ArtikelNr* der Tabelle *Artikel* gebunden.

Dropdown-Liste

Aus der Dropdown-Liste kann nun für eine bestimmte Rechnungsposition eine Artikelbezeichnung ausgewählt werden. Wenn allerdings kein Text aus der Liste ausgewählt, sondern eine neue, noch nicht vorhandene Artikelbezeichnung eingegeben wird, erscheint eine Fehlermeldung. Die Aufgabe einer *NotInList*-Ereignisprozedur muss nun sein, die eingegebene Artikelbezeichnung als neuen Artikel in die Tabelle *Artikel* aufzunehmen.



Eine Ereignisprozedur, die für die *BeiNichtInListe*-Ereigniseigenschaft angelegt wird, hat den folgenden Aufbau:

```
Sub ArtikelNr_NotInList (NewData As String, Response _
    As Integer)

End Sub
```

Der Parameter *NewData* enthält den Text, den der Anwender in das Kombinationsfeld eingegeben hat. In dem Beispiel würde dies der neu eingegebenen Artikelbezeichnung entsprechen.

Über den zweiten Parameter *Response* haben Sie die Gelegenheit, das weitere Vorgehen von Access zu beeinflussen. Dazu kann während der Abarbeitung der Prozedur eine der folgenden Konstanten angegeben werden:

Der *NewData*-Parameter

Der *Response*-Parameter

Konstanten des Parameters *Response*

Konstante	Beschreibung
<i>acDataErrDisplay</i>	Es wird die Standardfehlermeldung von Access angezeigt. Diesen Parameter sollten Sie angeben, wenn Sie den eingegebenen Text der Dropdown-Liste nicht hinzufügen.
<i>acDataErrContinue</i>	Es wird keine Meldung angezeigt.
<i>acDataErrAdded</i>	Es wird ebenfalls keine Meldung angezeigt. Access fragt die Datenquelle des Kombinationsfeldes allerdings neu ab und untersucht, ob der eingegebene Text während der Abarbeitung der Ereignisprozedur hinzugefügt wurde. Diese Konstante sollten Sie also nur dann angeben, wenn Sie den Listeneintrag wirklich hinzugefügt haben.

Tabelle 7.5

Es folgt die vollständige *NotInList*-Ereignisprozedur, die Sie im Formular *Rechnungen, Unterformular Rechnungspositionen* der Beispieldatenbank finden:

```
Private Sub ArtikelNr_NotInList(NewData As String, _
    Response As Integer)

Dim Meldung As String
Dim DB As Database
Dim Artikel As Recordset
```





```

Meldung = "Sie haben einen Artikel eingegeben, " & _
          "der noch nicht vorhanden ist. Soll " & _
          "der Artikel hinzugefügt werden?"

If MsgBox(Meldung, vbYesNo, "Neuer Artikel") _
    = vbYes Then

    "Artikel hinzufügen

    Set DB = DBEngine(0)(0)
    Set Artikel = DB.OpenRecordset("Artikel")

    Artikel.AddNew
        Artikel!Artikelbezeichnung = NewData
    Artikel.Update

    "DropDown-Liste erneut abfragen
    Response = acDataErrAdded

Else "Nein

    "Standard-Fehlermeldung anzeigen
    Response = acDataErrDisplay

End If

End Sub

```

Die Prozedur zeigt zunächst eine Meldung an, um sicherzustellen, dass der Anwender wirklich einen neuen Artikel hinzufügen möchte und sich nicht etwa vertippt hat. Wird diese Meldung positiv quittiert, fügt die Prozedur der Tabelle *Artikel* einen neuen Datensatz hinzu. Dem Feld *Artikelbezeichnung* wird hierbei der neu eingegebene Text aus der Variablen *NewData* zugewiesen. Schließlich wird Access über den *Response*-Parameter angewiesen, die Tabelle *Artikel* neu abzufragen und damit die Dropdown-Liste des Kombinationsfeldes zu aktualisieren.

Seitenumbrüche

Steuerelemente des Typs *Seitenumbruch* haben die Aufgabe, ein Formular in mehrere Bereiche zu unterteilen. Zwischen diesen Seiten kann der Anwender dann mithilfe der Tasten  und  wechseln. In der Beispieldatenbank finden Sie ein entsprechendes Kundenformular mit dem Namen *Seitenumbruchtest1*. Dieses Formular ist mithilfe zweier Seitenumbrüche in insgesamt drei Seiten unterteilt.



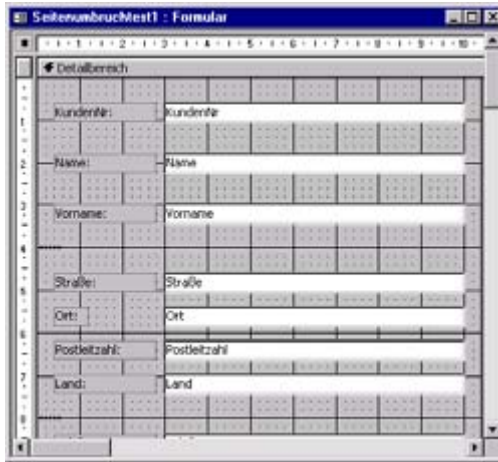


Abb. 7.12

Das Formular *Seitenumbruchtest1* in der Entwurfsansicht

Nach dem Öffnen des Formulars können Sie über die Tasten **Bild ↓** und **Bild ↑** die einzelnen Seiten auswählen.

Der Sprung zu einer bestimmten Seite kann auch von VBA aus erledigt werden. Dazu stellen alle Formulare die *GotoPage*-Methode zur Verfügung:

GotoPage-Methode

```
<Form-Objekt>.GotoPage <PageNumber> [, <Right>, <Down>]
```

Das Argument *<PageNumber>* bestimmt hierbei die Seitennummer, zu der gewechselt werden soll. Die Angabe der optionalen Argumente *<Right>* und *<Down>* ist dagegen nur erforderlich, falls die Breite oder Höhe eines Formulars geringer ist als die Seitenausmaße. Sie bestimmen hiermit die horizontale und vertikale Position des Seitenausschnitts, den Access anzeigen soll.

Ein praktisches Anwendungsbeispiel dieser Methode finden Sie in dem Formular *Seitenumbruchtest2*. Im Kopf dieses Formulars befinden sich drei Umschaltflächen, die in der Optionsgruppe *ogrSeite* angeordnet sind. Wenn Sie eine dieser Umschaltflächen markieren, wechselt Access zu der passenden Seite.

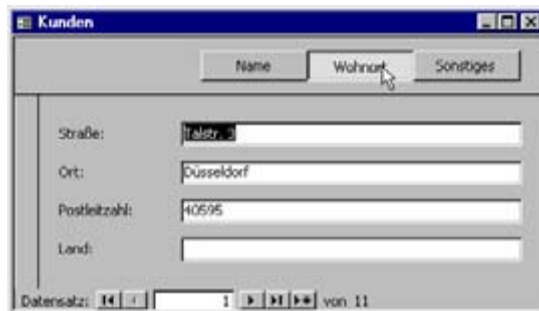


Abb. 7.13

Das Formular *Seitenumbruchtest2*





Die drei Umschaltflächen *Name*, *Wohnort* und *Sonstiges* haben jeweils die Optionswerte 1, 2 und 3. Um von der Markierung einer Schaltfläche informiert zu werden, kann das *Click*-Ereignis der Optionsgruppe verwendet werden. Die Ereignisprozedur hat folgenden Aufbau:

```
Private Sub ogrSeite_Click()  
  
    Me.GoToPage Me!ogrSeite  
  
End Sub
```

Wenn der Anwender beispielsweise die Schaltfläche *Wohnort* anklickt, erhält die Optionsgruppe den Wert 2. Dieser Wert wird der *GotoPage*-Methode als Parameter übergeben, so dass sie zur zweiten Formularseite wechselt.

Hinweis



Wenn das letzte Steuerelement einer Seite den Fokus hat und Sie die -Taste drücken, wechselt Access zum ersten Steuerelement der Folgeseite. Hierbei wird allerdings nicht vollständig zur zweiten Seite gewechselt, sondern der sichtbare Formelarausschnitt nur etwas nach unten verschoben. Es ist also der Rest der vorherigen und der Anfang der nächsten Seite sichtbar, was natürlich das optische Erscheinungsbild stört. Um dies zu verhindern, stellen Sie die *Zyklus*-Eigenschaft des Formulars auf *Aktuelle Seite* ein. Wenn Sie nun beim letzten Steuerelement die -Taste drücken, wechselt Access zum ersten Steuerelement der gleichen Seite.

Register

Eine ähnliche Aufgabe wie die Seitenumbrüche hat das *Register*-Steuerelement. Mit ihm ist es möglich, die Datenfelder eines Formulars in verschiedene Bereiche zu unterteilen und auf mehreren Seiten des Registers anzuordnen. Das *Register*-Steuerelement ist dabei wesentlich komfortabler zu handhaben als Seitenumbrüche, da die Seitengröße durch das Steuerelement vorgegeben und nicht mühsam durch Seitenumbrüche anzupassen ist.

In der Beispieldatenbank finden Sie das Formular *Register*. Es enthält das Register *tabKunden* mit drei Seiten, auf denen die Kundendaten nach *Name*, *Adresse* und *Sonstiges* gruppiert sind. Jeder Seite des *Register*-Steuerelements ist automatisch ein Seitenindex zugeordnet. Die Seite ganz links in der untersten Reihe hat dabei den Index 0, die letzte Seite den Index *Seitenanzahl - 1*. Der Wert des *Register*-Steuerelements ist die jeweils aktuelle Seite, die gerade angezeigt wird. Sie können diesen Wert abfragen oder verändern.

Hinweis



Die Eigenschaft *PageIndex* wird von Access automatisch nach der Anordnung der Seiten im *Register*-Steuerelement vergeben. Durch Ändern der *PageIndex*-Eigenschaft einer Seite in den Grenzen von 0 bis *Seitenanzahl - 1* ändern Sie die Anordnung der Seiten im *Register*-Steuerelement.



Ein praktisches Beispiel für den Zugriff auf das *Register*-Steuerelement finden Sie im Formular *Rechnungen und Register*. Das Formular enthält ein Kombinationsfeld, mit dem Sie den Kunden für die aktuelle Rechnung festlegen. Darunter befindet sich die Schaltfläche *btnAdresse*. Diese öffnet das Formular *Register* und zeigt die Adresse des entsprechenden Kunden an.

Abb. 7.14 ○

Das Formular
*Rechnungen und
Register*

The screenshot shows the Microsoft Access application window with the 'Rechnungen und Register' form open. The form has a title bar 'Dr. Zahnlos' and a menu bar with options like 'Datei', 'Bearbeiten', 'Ansicht', etc. The main area contains a subform 'Dr. Zahnlos' with a tab control showing 'Name', 'Adresse', and 'Sonstiges'. The 'Adresse' tab is active, showing fields for 'Strasse', 'Ort', 'Postleitzahl', and 'Land'. Below the subform is a table 'Rechnungspositionen' with columns: 'Artikel', 'Bemerkung', 'Anzahl', 'Einzelpreis', and 'Gesamtpreis'. The table contains two rows: 'Zahnbürste, einfach' with a quantity of 2 and a unit price of 20,00 DM, and 'Zahnbürste, kunst' with a quantity of 4 and a unit price of 33,30 DM. The total sum is 173,20 DM. The form also includes fields for 'Rechnungsdatum', 'Druckdatum', and 'Versandart'.

Die *Click*-Ereignisprozedur der Schaltfläche *btnAdresse* hat folgenden Aufbau:

```
Private Sub btnAdresse_Click()

    Dim pIndex As Integer
    Dim KundenForm As Form

    If IstGeöffnet("Register") = False Then _
        DoCmd.OpenForm "Register", , , , , acHidden

    Set KundenForm = Forms!Register

    pIndex = _
        KundenForm!tabKunden.Pages _
        ("pagAdresse").PageIndex
```





```

If FormularDatensatzAuswählen(KundenForm, _
  "KundenNr=" & Me!KundenNr) = True Then
    KundenForm!tabKunden = pIndex
    KundenForm.Visible = True
End If

```

End Sub

Zunächst wird, falls nicht bereits geladen, das Formular *Register* versteckt geöffnet. Anschließend ermittelt die Prozedur den Seitenindex der Seite *pagAdresse* des *Register*-Steuerelements. Die bereits bekannte Funktion *FormularDatensatzAuswählen* wechselt im Formular *Register* zu dem im Kombinationsfeld eingetragenen Kunden. Falls die Aktion erfolgreich ist, wird die Seite *pagAdresse* des Registers aktiviert und das Formular sichtbar gemacht.

Hinweis



In der obenstehenden Prozedur wird zunächst der Seitenindex der Seite *pagAdresse* ermittelt und dann dem *Register*-Steuerelement zugewiesen. Da Sie den Seitenindex der Seite kennen, könnten Sie die Indexnummer auch direkt zuweisen. Der Vorteil der obigen Methode ist jedoch, dass die Prozedur auch noch funktioniert, wenn Sie die Seitenanordnung im Register und damit die Indizierung der Seiten ändern. Natürlich gilt dies nur, solange Sie die Seite *pagAdresse* nicht umbenennen.

Register nachträglich einsetzen

Wenn Sie ein bestehendes Formular, das Seitenumbrüche zur Gliederung der Steuerelemente benutzt, nachträglich mit einem *Register*-Steuerelement realisieren wollen, gehen Sie wie folgt vor:

- ☐ Erzeugen Sie ein *Register*-Steuerelement in dem Formular, das die Seitenumbrüche enthält.
- ☐ Legen Sie so viele neue Seiten an, wie Sie zur Aufteilung der Daten benötigen.
- ☐ Markieren Sie die Steuerelemente *einer* ehemaligen Seite.
- ☐ Wählen Sie den Menübefehl *Bearbeiten > Ausschneiden*.
- ☐ Markieren Sie die Seite des *Register*-Steuerelements, in die Sie die Steuerelemente einfügen wollen.
- ☐ Wählen Sie den Menübefehl *Bearbeiten > Einfügen*.
- ☐ Wiederholen Sie die Schritte 3 bis 7 für alle Seiten des Formulars.
- ☐ Löschen Sie die *Seitenumbruch*-Steuerelemente.



Hyperlinks

Die große Verbreitung des Internets macht auch vor Microsoft Access nicht halt. In Access wurde der neue Felddatentyp *Hyperlink* eingeführt, mit dem Sie WWW-Adressen, FTP-Adressen oder auch Dateipfade auf dem eigenen Rechner oder im Netz speichern können. In der Datenblattansicht einer Tabelle, die ein solches Feld enthält, wird der Hyperlink so dargestellt, wie er auch im Internet Explorer erscheinen würde (normalerweise in Hellblau und unterstrichen). Bewegen Sie den Mauszeiger auf einen Hyperlink, verwandelt sich dieser in eine Hand.

Wenn Sie auf einen Hyperlink klicken, wird die angegebene Adresse mit der entsprechenden Anwendung geöffnet. Bei einer WWW-Adresse ist dies der Internet Explorer bzw. der Standard-Webbrowser Ihres Systems. Sollten Sie aber beispielsweise eine Word-Datei mit der Endung *.doc* angeben, wird automatisch WinWord gestartet und das Dokument geladen.

Hyperlinks einer Tabelle

Mittels VBA können Sie auf die Hyperlinks, die in einem Datenfeld einer Tabelle gespeichert sind, zugreifen, indem Sie die *Hyperlink*-Eigenschaft des Datenfeldes nutzen. Formal verwenden Sie die Eigenschaft wie folgt:

```
<Steuerelement>.Hyperlink
```

Die *Hyperlink*-Eigenschaft ist für Textfelder, Kombinationsfelder, Befehlsschaltflächen, Bezeichnungsfelder und *Bild*-Steuerelemente verfügbar.



Hinweis

In der Beispieldatenbank zu diesem Kapitel finden Sie ein Formular mit dem Namen *Hyperlinks1*. Als Datenquelle dient die Tabelle *Hyperlinks*, bei der das Datenfeld *WWW-Adresse* vom Typ *Hyperlink* ist.

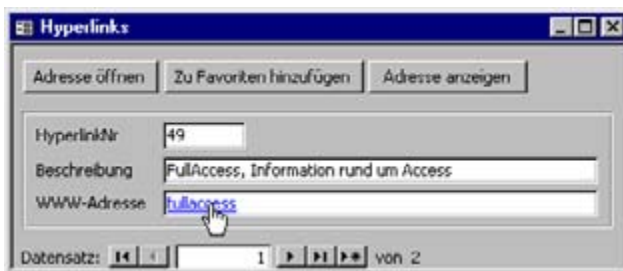


Abb. 7.15

Das Formular
Hyperlinks1

Über die ersten zwei Befehlsschaltflächen öffnen Sie entweder die WWW-Adresse des aktuellen Datensatzes oder fügen die Adresse den Favoriten hinzu. Um eine Adresse zu öffnen, verwenden Sie die *Follow*-Methode der *Hyperlink*-Eigenschaft. Mit der *AddToFavorites*-Methode der *Hyperlink*-Eigenschaft fügen Sie eine Adresse den Favoriten hinzu.

○ *AddToFavorites*-
Methode
Follow-Methode





Die *Click*-Ereignisprozeduren der Schaltflächen lauten dementsprechend:

```
Private Sub btnÖffnen_Click()  
  
    Me!WWWAdresse.Hyperlink.Follow  
  
End Sub  
  
Private Sub btnFavoriten_Click()  
  
    Me!WWWAdresse.Hyperlink.AddToFavorites  
  
End Sub
```

Bei der *Follow*-Methode können Sie eine Reihe optionaler Parameter angeben:

```
<Objekt>.Hyperlink.Follow([(<NewWindow>],  
[<AddHistory>], [<ExtraInfo>], [<Method>],  
[<HeaderInfo>]))
```

○ **<NewWindow>-
Parameter**

Die wichtigsten Parameter sind dabei *<NewWindow>* und *<AddHistory>*. Weisen Sie dem Parameter *<NewWindow>* den Wert *True* zu, um das angegebene Dokument in einem neuen Fenster zu öffnen. Mit *False* wird das aktuelle Fenster verwendet. Dies ist die Standardeinstellung.

○ **<AddHistory>-
Parameter**

Mittels des Parameters *<AddHistory>* wird der Hyperlink dem Ordner *Verlauf* des *Windows*-Verzeichnisses hinzugefügt. Wenn Sie das nicht wollen, müssen Sie statt der Standardeinstellung *True* die Einstellung *False* als Parameter angeben.

Aufbau einer Adresse

Der Inhalt eines *Hyperlink*-Feldes kann aus drei Teilen bestehen. Dies sind der Anzeigetext, der im Textfeld des Formulars oder der Tabelle für den Benutzer sichtbar ist, die eigentliche Adresse und eine Unteradresse, die für Dateizugriffe verwendet wird. Der formale Aufbau sieht folgendermaßen aus:

Anzeigetext#Adresse#Unteradresse

○ **HyperlinkPart-
Funktion**

Um einen Teil eines bestehenden Hyperlinks mittels VBA zu ermitteln, stellt das *Application*-Objekt die Funktion *HyperlinkPart* zur Verfügung.

```
[<Objekt>].HyperlinkPart(<Hyperlink>, <Part>)
```

<Objekt> ist das *Application*-Objekt. Die Angabe des Objekts ist optional und kann deshalb weggelassen werden:

```
HyperlinkPart(<Hyperlink>, <Part>)
```



Mit dem Parameter *<Hyperlink>* geben Sie den Hyperlink an, von dem Sie einen Teil ermitteln wollen. Der Parameter *<Part>* entscheidet dann, welcher Teil der Adresse als String zurückgegeben wird. In der folgenden Tabelle finden Sie die möglichen Konstantenwerte des Parameters *Part* aufgelistet.

Konstanten des Parameters *Part*

Konstante	Wert	Beschreibung
<i>acDisplayedValue</i>	0	Der unterstrichene Text, der in einem Hyperlink angezeigt wird. Dies ist der Standardwert.
<i>acDisplayText</i>	1	Anzeigetext eines Hyperlink-Feldes.
<i>acAddress</i>	2	Adresse eines Hyperlink-Feldes.
<i>acSubAddress</i>	3	Unteradresse eines Hyperlink-Feldes.

Tabelle 7.6

Im Formular *Hyperlinks1* der Beispieldatenbank öffnen Sie über die Schaltfläche *btnAdresseAnzeigen* ein Meldungsfenster, in dem die Adresse des aktuellen Hyperlinks angezeigt wird. Die *Click*-Ereignisprozedur der Schaltfläche lautet:

```
Private Sub btnAdresseAnzeigen_Click()

    MsgBox HyperlinkPart(Me!WWWAdresse, acAddress)

End Sub
```

Hyperlinks mit Bezeichnungsfeldern, Befehlsschaltflächen und *Bild*-Steuerelementen

Neben Text- und Kombinationsfeldern, die Hyperlinks aus einer Tabelle anzeigen, steht die *Hyperlink*-Eigenschaft nur noch Bezeichnungsfeldern, Befehlsschaltflächen und *Bild*-Steuerelementen zur Verfügung. Mit diesen Steuerelementen ist es möglich, feste Hyperlink-Adressen in einem Formular anzulegen. So können Sie beispielsweise eine ansprechende Grafik als Hyperlink zu einer WWW-Adresse verwenden. Die Eigenschaften *Hyperlink-Adresse* (Engl.: *HyperlinkAddress*) und *Hyperlink-Unteradresse* (Engl.: *HyperlinkSubAddress*) finden Sie im *Eigenschaften*-Fenster des jeweiligen Steuerelements im Register *Format*.

Sie können die *Hyperlink*-Eigenschaften dieser Steuerelemente jedoch auch dynamisch festlegen. Ein Beispiel dafür finden Sie im Formular *Hyperlinks2* der Beispieldatenbank zu diesem Kapitel.



Abb. 7.16

Das Formular
Hyperlinks2

Das Formular basiert auf der Tabelle *Lieferanten*, die zusätzlich zu den allgemeinen Daten noch das Tabellenfeld *Hotline* vom Typ *Hyperlink* enthält. Die Hyperlinks dieses Feldes werden jedoch nicht in einem Textfeld angezeigt, sondern sollen über eines der drei Steuerelemente im Formulkopf aktiviert werden.

Zu diesem Zweck wird beim Anzeigen eines Datensatzes zunächst mittels der *HyperlinkPart*-Funktion der Adressteil des Datenfeldes *Hotline* des aktuellen Datensatzes ermittelt. Anschließend wird die Adresse den drei Steuerelementen über die Eigenschaft *HyperlinkAddress* zugewiesen. Die Ereignisprozedur der *BeimAnzeigen*-Eigenschaft des Formulars lautet folgendermaßen:

```
Private Sub Form_Current()

    Dim WWWAdresse As String

    WWWAdresse = HyperlinkPart(Me!Hotline, acAddress)

    Me!imgWWWAdresse.HyperlinkAddress = WWWAdresse
    Me!lblWWWAdresse.HyperlinkAddress = WWWAdresse
    Me!btnWWWAdresse.HyperlinkAddress = WWWAdresse

End Sub
```





Der Text auf der Schaltfläche oder im Bezeichnungsfeld wird nicht automatisch wie bei Textfeldern angezeigt, sondern muss weiterhin mittels der *Caption*-Eigenschaft dem Steuerelement zugewiesen werden. Auch die hellblaue Farbe und das Unterstreichen des Textes legen Sie über die Standardeigenschaften des Steuerelements fest.



Hinweis

Verweise auf Dateien

Im Beispielformular verweisen die Steuerelemente des Formulars auf eine Internetadresse. Es ist aber auch möglich, einen Verweis auf eine Datei Ihres Computers oder eines Netzwerkrechners anzugeben.

Die Eigenschaft *HyperlinkAddress* gibt in diesem Zusammenhang den vollständigen Pfad der Datei an, zum Beispiel:

```
<Steuerelement>.HyperlinkAddress = _
    "C:\Daten\Präsentation.ppt"
```

Zusätzlich zu dem Dateinamen lässt sich eine Unteradresse über die Eigenschaft *HyperlinkSubAddress* definieren, deren Bedeutungen für die Standard-Office-Anwendungen in der folgenden Tabelle aufgeführt sind.

Werte für die Subadresse eines Hyperlinks

Wert	Beispiel	Dokumenttyp
<Textmarke>	Test	Word-Dokument (*.doc)
<Tabelle><Bereich>	Tabelle1!Artikel	Excel-Dokument (*.xls)
<Foliennummer>	3	PowerPoint-Dokument (*.ppt)
<Objekttyp> <Objektname>	Form Artikel	Access-Objekt (*.mdb)

Tabelle 7.7

So wird durch den Zusatz

```
<Steuerelement>.HyperlinkSubAddress = "10"
```

die zehnte Folie des PowerPoint-Dokuments *Präsentation.ppt* beim Aktivieren des Hyperlinks geöffnet.

Wie aus *Tabelle 7.7* ersichtlich, ist es auch möglich, Access-Objekte der aktuellen oder einer anderen Datenbank mit einem Hyperlink zu öffnen. Handelt es sich um ein Objekt der aktuellen Datenbank, geben Sie nur die Subadresse ein. Wollen Sie ein Objekt einer anderen Datenbank öffnen, geben Sie zusätzlich mit der *HyperlinkAddress*-Eigenschaft den Pfad dieser MDB-Datei an.





Berichte programmieren



Berichte öffnen, drucken und schließen	418
Beeinflussung des Layouts	421
Rechnen in Berichten	425
Zeichnen auf Berichten	428





Die Aufgabe von Berichten ist, Daten aus Tabellen optisch ansprechend auf Papier zu bringen. Die Hauptaufgabe beim Entwurf eines Berichts liegt dementsprechend in der Positionierung und Anordnung der einzelnen Steuerelemente.

Da Berichte eine reine Präsentationsaufgabe haben, stellen sie auch nur wenig Möglichkeiten zur Programmierung zur Verfügung. So haben beispielsweise Steuerelemente, die auf Berichten angeordnet werden, keine Ereignisseigenschaften und lösen damit auch keine Ereignisse aus. Die Aufgaben des Programmierers beschränken sich daher auf die Beeinflussung des Layouts.

In diesem Kapitel erfahren Sie, wie Sie mit Berichten umgehen und diese in Ihre Datenbankanwendung integrieren. Als Datenzugriffsmodell wird DAO verwendet.

Alle Beispiele zu diesem Kapitel finden Sie in der Beispieldatenbank *Kapitel8.mdb*.

Berichte öffnen, drucken und schließen

Berichte öffnen



Zum Öffnen bzw. Drucken eines Berichts stellt Access die Methode *OpenReport* des *DoCmd*-Objekts zur Verfügung, die der Methode *OpenForm* zum Öffnen eines Formulars stark ähnelt. Die *OpenReport*-Methode hat folgende Syntax:

```
DoCmd.OpenReport <ReportName> [, <View>]  
[ , <FilterName>] [ , <WhereCondition>]
```

View



Den Namen des Berichts, den Sie öffnen möchten, geben Sie unter *<ReportName>* an. Das zweite Argument mit dem Namen *<View>* bestimmt, in welcher Ansicht der Bericht geöffnet wird. Es stehen folgende Möglichkeiten zur Verfügung:

Konstanten des Parameters <View>



Konstante	Beschreibung
<i>acViewNormal</i>	Der Bericht wird direkt auf dem Drucker ausgegeben. Dies ist die Standardeinstellung, falls Sie das Argument <i><View></i> nicht angeben.
<i>acViewDesign</i>	Der Bericht wird in der Entwurfsansicht geöffnet.
<i>acViewPreview</i>	Der Bericht wird in der Seitenansicht geöffnet.

Tabelle 8.1



In der Beispieldatenbank finden Sie einen Bericht mit dem Namen *Kundenliste*, der eine Liste aller Kunden ausgibt. Um diesen Bericht direkt auszudrucken, geben Sie eine der beiden folgenden Anweisungen an:



```
DoCmd.OpenReport ReportName:="Kundenliste"  
DoCmd.OpenReport ReportName:="Kundenliste",  
View:=acViewNormal
```

Wenn Sie den Bericht dagegen zuerst in der Seitenansicht auf dem Bildschirm begutachten möchten, verwenden Sie die folgende Anweisung:

```
DoCmd.OpenReport ReportName:="Kundenliste",  
View:=acViewPreview
```

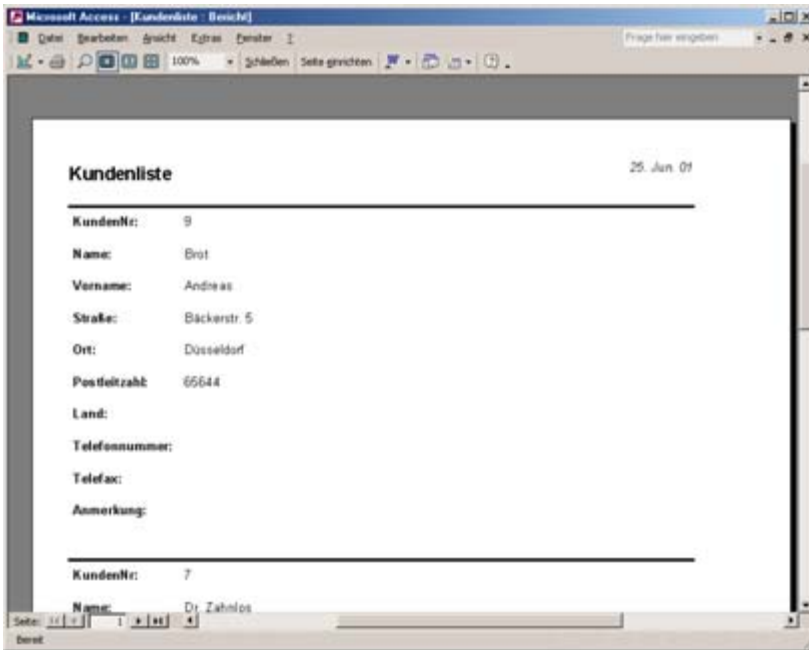


Abb. 8.1

Der Bericht *Kundenliste* in der Seitenansicht

Zum Schließen eines Objekts haben Sie bereits im *Kapitel 6 Formulare programmieren* die Methode *Close* kennen gelernt, die auch auf Berichte angewendet werden kann. Voraussetzung für den Einsatz dieser Methode ist allerdings, dass der Bericht entweder in der Seitenansicht oder der Entwurfsansicht geöffnet wurde. Die folgende Anweisung schließt beispielsweise den Bericht *Kundenliste*:

```
DoCmd.Close acReport, "Kundenliste"
```

Genau wie Formulare stellen auch Berichte die beiden Ereignisseigenschaften *BeimÖffnen* und *BeimSchließen* zur Verfügung, mit deren Hilfe auf die beiden Ereignisse *Open* und *Close* reagiert werden kann. In der Praxis spielen diese Ereignisse im Zusammenhang mit Berichten allerdings kaum eine Rolle.

Berichte schließen



Hinweis





Drucken einzelner Datensätze

Bedingung angeben

Sobald ein Bericht geöffnet wird, enthält dieser alle Datensätze der angegebenen Datenquelle. Mithilfe der beiden Parameter `<FilterName>` und `<WhereCondition>` ist es möglich, die Datenquelle zu filtern und damit nur bestimmte oder sogar nur einen einzelnen Datensatz zu drucken. Die beiden Argumente können hierbei auf die gleiche Weise wie beim Öffnen eines Formulars verwendet werden. Die folgende Anweisung führt beispielsweise dazu, dass Access den Bericht *Kundenliste* in der Seitenansicht öffnet und hierbei nur den Kunden mit der *KundenNr* 8 ausgibt:

```
DoCmd.OpenReport ReportName:="Kundenliste", _
    View:=acViewPreview, _
    WhereCondition:="KundenNr = 8"
```

Zum praktischen Einsatz kommt diese Methode im Formular *Kunde drucken* der Beispieldatenbank, das die Auswahl eines beliebigen Kunden zwecks Ausdruck erlaubt.

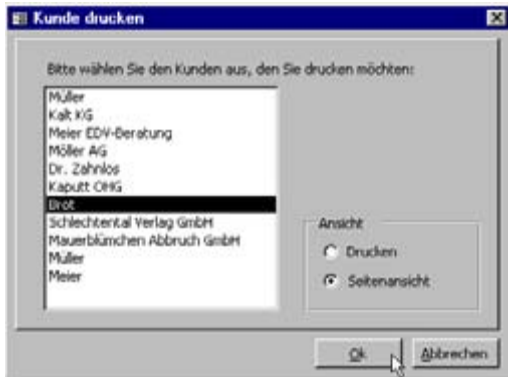


Abb. 8.2

Das Formular
Kunde drucken

Das Formular besitzt eine Kundenliste mit dem Namen *IstKundenliste*, über die Sie einen bestimmten Kunden auswählen können. Das Listenfeld ist hierbei an das Datenfeld *KundenNr* gebunden, so dass ein Verweis auf die Liste immer die Kundennummer des momentan ausgewählten Kunden zum Ergebnis hat.

Außerdem befindet sich auf dem Formular eine Optionsgruppe, die dem Anwender die Auswahl der Ansicht erlaubt. Das Optionsfeld zum Ausdrucken hat hierbei den Optionswert 1, das für die Seitenansicht den Optionswert 2.

OK-Schaltfläche

Durch einen Klick auf die *OK*-Schaltfläche wird der Bericht mithilfe der folgenden Ereignisprozedur geöffnet:

```
Private Sub btnOk_Click()
```

```
Dim Ansicht As Integer
```





```

If IsNull(Me!lstKundenliste) Then Exit Sub

If Me!ogrAnsicht = 1 Then _
    Ansicht = acViewNormal Else _
    Ansicht = acViewPreview

DoCmd.OpenReport ReportName:="Kundenliste", _
    View:=Ansicht, _
    WhereCondition:="KundenNr = " & _
    Me!lstKundenliste

DoCmd.Close acForm, Me.Name

End Sub

```

Die Prozedur überprüft zunächst, ob in der Liste überhaupt ein Kunde ausgewählt wurde. Sollte dies nicht der Fall sein, wird die Ereignisprozedur sofort verlassen. Im nächsten Schritt wird die ausgewählte Ansicht der Optionsgruppe *ogrAnsicht* ermittelt und die entsprechende Konstante in der Variablen *Ansicht* abgelegt. Das Öffnen des Berichts in der passenden Ansicht erledigt schließlich die *OpenReport*-Anweisung. Als Bedingung wird hierbei der Ausdruck

```
"KundenNr = " & Me!lstKundenliste
```

angegeben, der auf die Kundennummer des momentan in der Liste ausgewählten Kunden zugreift. Nachdem der Bericht geöffnet wurde, schließt die Prozedur schließlich noch das Formular *Kunde drucken*.

Wenn die Datenquelle, die dem Bericht zugrunde liegt, keine Datensätze enthält, löst Access für den Bericht das *NoData*-Ereignis aus, auf das Sie mittels der *Bei Ohne Daten*-Ereigniseigenschaft reagieren können. Die Ereignisprozedur stellt den Parameter *Cancel* zur Verfügung, über den Sie die Ausgabe des Berichts abbrechen können. Sie vermeiden so den Ausdruck leerer Berichte.



Hinweis

Beeinflussung des Layouts

Wie bereits eingangs dieses Kapitels erwähnt, stellen Berichte nur einige wenige Ereigniseigenschaften zur Verfügung, die speziell zur Beeinflussung des Layouts dienen. Ein wichtiges Ereignis ist in diesem Zusammenhang das *Format*-Ereignis (*BeimFormatieren*-Ereigniseigenschaft), das von allen Bereichen eines Berichts zur Verfügung gestellt wird.

Bevor Access einen bestimmten Bereich auf dem Drucker oder in der Seitenansicht ausgeben kann, müssen die einzelnen Steuerelemente in dem Bereich angeordnet und formatiert werden. Access beachtet hierbei beispielsweise, dass sich verschiedene Steuerelemente je nach Inhalt in der Größe ändern können (*Vergrößern* > *Verkleinern*-Eigenschaften).



Die
BeimFormatieren-
Ereigniseigenschaft



Bei jedem Formatierungsvorgang eines Bereichs löst Access das *Format*-Ereignis aus, über das Einfluss auf die Formatierung genommen werden kann. Die Ereignisprozedur zum *Format*-Ereignis hat den folgenden Aufbau:

```
Private Sub <Bereichsname>_Format (Cancel As Integer, _
    FormatCount As Integer)

End Sub
```

Cancel-Parameter

Mithilfe des Parameters *Cancel* können Sie die Formatierung eines Bereichs abbrechen. In der Praxis bedeutet dies, dass Access den Bereich nicht auf dem Drucker ausgibt, aber auch keine weiße Fläche hinterlässt.

FormatCount-Parameter

Der zweite Parameter mit dem Namen *FormatCount* gibt an, ob ein Bereich mehrmals formatiert wurde. Dies tritt beispielsweise dann auf, wenn ein Bereich nicht mehr vollständig auf die aktuelle Seite passt und daher teilweise oder vollständig auf die nächste Seite verschoben wird. In diesem Fall löst Access das *Format*-Ereignis für den betreffenden Bereich einmal mit dem *FormatCount*-Wert 1 und ein zweites Mal mit dem *FormatCount*-Wert 2 aus.

Datensätze hervorheben

Ein praktisches Beispiel für den Einsatz des *Format*-Ereignisses ist das Hervorheben bestimmter Datensätze. In der Beispieldatenbank finden Sie einen Bericht mit dem Namen *Kundenliste2*, der wieder eine Liste aller Kunden ausgibt. Der Unterschied zu dem vorhergehenden Bericht besteht nun darin, dass alle Kunden, deren Postleitzahl mit 4 beginnt, durch einen Rahmen hervorgehoben werden.

Abb. 8.3

Die Seitenansicht des Berichts *Kundenliste2*

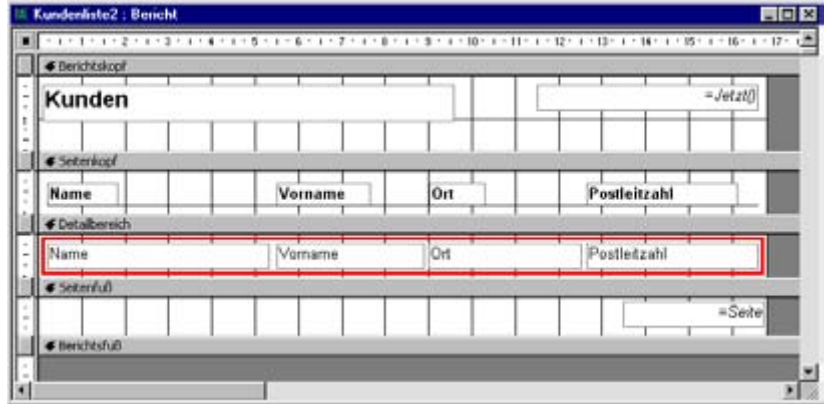
Name	Vorname	Ort	Postleitzahl
Müller	Willi	Düsseldorf	45345
Kalt KG		Hamburg	24643
Meier EDV-Beratung		Düsseldorf	53655
Müller AG		Köln	53457
Dr. Zahnlos		Köln	53434
Kaputt OHG		Düsseldorf	36345
Brost	Andreas	Düsseldorf	65644
Schlechtental Verlag GmbH		Köln	53457
Maerklumchen Abbruch Gmb		Düsseldorf	40595
Müller	Dieter	Düsseldorf	40595
Meier	Fritz	Köln	40595



Die Hervorhebung des Datensatzes wird durch ein normales Rechteck ermöglicht, das in der Entwurfsansicht des Berichts innerhalb des Detailbereichs erstellt wurde. Dieses Rechteck trägt den Namen *recRahmen*. Damit die Datenfelder hinter dem Rechteck sichtbar sind, muss die *Hintergrundart*-Eigenschaft des Rechtecks auf *Transparent* eingestellt sein.

Abb. 8.4 ○

Der Bericht in der Entwurfsansicht



Dieses Rechteck muss nun abhängig vom Inhalt des Feldes *Postleitzahl* ein- oder ausgeblendet werden. Diese Aufgabe wird von der folgenden Ereignisprozedur erledigt, die an das *Format*-Ereignis des Detailbereichs geknüpft ist:

○ *Format*-
Ereignisprozedur

```
Private Sub Detail1_Format(Cancel As Integer,
FormatCount As Integer)

    If Left(Me!Postleitzahl, 1) = "4" Then

        Me!recRahmen.Visible = True

    Else

        Me!recRahmen.Visible = False

    End If

End Sub
```

Innerhalb der Prozedur wird mithilfe des Ausdrucks *Me!Postleitzahl* auf das Feld *Postleitzahl* des Datensatzes verwiesen, der momentan zur Formatierung vorbereitet wird. Dieser Datensatz kann mit dem jeweils aktuellen Datensatz von Formularen verglichen werden. Mithilfe einer *If*-Abfrage wird untersucht, ob die *Postleitzahl* mit der Ziffer 4 beginnt. Sollte dies der Fall sein, blendet Access das Rechteck mithilfe der *Visible*-Eigenschaft (Deutsch: *Sichtbar*-Eigenschaft) ein. Andernfalls wird es ausgeblendet.





Datensatzwechsel

Die *NextRecord*-Eigenschaft

Nachdem Access den Detailbereich eines Berichts zur Ausgabe formatiert und ausgedruckt hat, wird automatisch immer der nachfolgende Datensatz der Datenquelle zum aktuellen Datensatz gemacht. Dies können Sie verhindern, indem Sie der *NextRecord*-Eigenschaft während der Abarbeitung des *Format*-Ereignisses den Wert *False* zuweisen.

Mehrere Datensatzkopien drucken

Eine praktische Anwendung ist beispielsweise das automatische Erstellen von Kopien bestimmter Datensätze. In der Beispieldatenbank finden Sie einen Bericht mit dem Namen *NextRecord-Test*, der alle Kunden, deren Postleitzahlen mit der Ziffer 4 beginnen, doppelt ausgibt. Dies könnte beispielsweise beim Bedrucken von Aufklebern erforderlich sein.

Die *Format*-Ereignisprozedur des Detailbereichs hat den folgenden Aufbau:

```
Private Sub Detail1_Format (Cancel As Integer,
FormatCount As Integer)
```

```
    If Left(Me!Postleitzahl, 1) = "4" And Kopie = 0 Then
```

```
        Kopie = 1
```

```
        Me.NextRecord = False
```

```
    Else
```

```
        Kopie = 0
```

```
    End If
```

```
End Sub
```

Die Funktion dieser Prozedur basiert auf der privaten Variablen *Kopie*, die in folgender Weise im Deklarationsteil des Berichtsmoduls deklariert wurde:

```
Private Kopie As Integer
```

Beim Öffnen des Berichts wird diese Variable zunächst einmal von Access automatisch mit dem Wert 0 initialisiert. Sollte Access beim Formatieren des Detailbereichs nun auf einen Datensatz treffen, dessen Postleitzahl mit der Ziffer 4 beginnt, wird der Wert der Variablen *Kopie* auf 1 erhöht. Der nachfolgende Datensatzvorschub wird außerdem mithilfe der Anweisung

```
Me.NextRecord = False
```

unterdrückt. Wenn Access nun den nächsten Detailbereich formatiert, ist der gleiche Datensatz wie bei dem vorhergehenden Detailbereich aktiv. Der Datensatz wird also ein zweites Mal gedruckt. Da die Variable *Kopie* nun nicht mehr den Wert 0, sondern 1 hat, wird der nachfolgende Datensatzvorschub nicht mehr verhindert. Stattdessen wird die Variable wieder auf den Initialisierungswert 0 gesetzt und der Druckvorgang kann mit dem nächsten Datensatz fortgesetzt werden.



Neben der Möglichkeit, den Datensatzvorschub zu unterdrücken, erlaubt Access auch, den Vorschub auf den nächsten Druckbereich zu verhindern. Dazu muss der *MoveLayout*-Eigenschaft einfach der Wert *False* zugewiesen werden. In diesem Fall wird der nachfolgende Datensatz direkt über den ersten Datensatz gedruckt.

Datensatzvorschub unterdrücken

Eine dritte Eigenschaft mit dem Namen *PrintSection* erlaubt es, den Ausdruck eines Bereichs zu verhindern. Diese Eigenschaft hat damit die gleiche Funktion wie der *Cancel*-Parameter des *Format*-Ereignisses.

Rechnen in Berichten

Bei Berichtsbereichen stellt Access neben dem *Format*-Ereignis noch ein zweites Ereignis mit dem Namen *Print* zur Verfügung. Die Reaktion auf dieses Ereignis erfolgt mithilfe der *BeimDrucken*-Ereigniseigenschaft.

Das *Print*-Ereignis

Das *Print*-Ereignis wird von Access für einen bestimmten Bereich ausgelöst, wenn die Formatierung bereits abgeschlossen ist und bevor mit der Ausgabe des Bereichs begonnen wird. Die Ereignisprozedur hat den folgenden Aufbau:

```
Private Sub <Bereichsname>_Print (Cancel As Integer,
PrintCount As Integer)
```

```
End Sub
```

Die beiden Argumente entsprechen denen des *Format*-Ereignisses. So kann der Ausdruck des Bereichs beispielsweise verhindert werden, indem der *Cancel*-Eigenschaft während der Abarbeitung des Ereignisses der Wert *True* zugewiesen wird. In diesem Fall gibt Access anstelle des Bereichs eine leere Fläche aus.

Mithilfe des *PrintCount*-Parameters lässt sich feststellen, wie oft der Bereich ausgegeben wurde. Sollte sich der aktuelle Bereich beispielsweise über zwei Seiten erstrecken, wird die Ereignisprozedur einmal mit dem *PrintCount*-Wert 1 und ein zweites Mal mit dem *PrintCount*-Wert 2 aufgerufen.

Der *PrintCount*-Parameter

Der praktische Einsatzbereich dieses Ereignisses liegt vor allem in der Durchführung von Berechnungen. Da beim Eintritt des Ereignisses sowohl die Formatierung des Bereichs abgeschlossen ist, als auch die in dem Bereich zu druckenden Daten feststehen, können bei der Berechnung keine Fehler unterlaufen. Die doppelte Berücksichtigung eines Datensatzes kann dadurch ausgeschlossen werden, dass die Berechnung nur dann ausgeführt wird, wenn die *PrintCount*-Eigenschaft den Wert 1 hat.

In der Beispieldatenbank finden Sie einen entsprechenden Bericht zum Ausdruck von Rechnungen. Bei längeren Rechnungen, die sich über mehr als eine Seite erstrecken, wird im Fuß eine Seitensumme ausgegeben.

Seitensummen berechnen





Der Bericht ist für den Ausdruck einer einzelnen Rechnung vorgesehen. Wenn der Bericht geöffnet wird, müssen die zugrunde liegenden Datensätze daher mit einer Bedingung auf genau eine Rechnung eingeschränkt werden. Im Formular *Hauptmenü* finden Sie eine Schaltfläche mit der Beschriftung *RechnungNr 1 in der Seitenansicht öffnen*, die mithilfe der folgenden Anweisung die Rechnung mit der Nummer 1 in der Seitenansicht öffnet:

```
DoCmd.OpenReport "Rechnung", acViewPreview, , _
    "Rechnungen.RechnungsNr = 1"
```

Hinweis



Die Angabe des Tabellennamens innerhalb der Bedingung ist erforderlich, da die Datenquelle der Rechnung mehrere Felder mit dem Namen *RechnungsNr* enthält.

Zur Anzeige der Seitensumme enthält der Bericht im Seitenfuß ein ungebundenes Textfeld mit dem Namen *txtZwischensumme*. Im Berichtfuß befindet sich ein zweites Steuerelement mit dem Namen *txtRechnungsbetrag*, das die Anzeige des kompletten Rechnungsbetrags erledigt. Dieser lässt sich relativ einfach mithilfe des folgenden Ausdrucks berechnen:

```
=Summe ([Gesamtpreis])
```

Dieser Ausdruck berechnet aus den Gesamtpreisen der einzelnen Rechnungspositionen die Gesamtsumme der Rechnung. Der Ausdruck befindet sich in der *Steuerelementinhalt*-Eigenschaft des Textfeldes, so dass die Berechnung von Access automatisch erledigt wird.

Zwischensumme berechnen



Die Berechnung der Zwischensumme kann leider nicht mithilfe einer in Access eingebauten Funktion erfolgen. Stattdessen müssen die Einzelpreise der einzelnen Positionen einer Rechnungsseite summiert und das Ergebnis in das Textfeld *txtZwischensumme* geschrieben werden. Diese Berechnung erledigt die folgende Ereignisprozedur, die an das *Print*-Ereignis des Detailbereichs geknüpft ist:

```
Private Sub Detail1_Print(Cancel As Integer, _
    PrintCount As Integer)

    If PrintCount = 1 Then

        Me!txtZwischensumme = Me!txtZwischensumme + _
            Me!Gesamtpreis

    End If

End Sub
```

Um sicherzustellen, dass eine Rechnungsposition nicht doppelt berücksichtigt wird, prüft die Prozedur zunächst den Inhalt des *PrintCount*-Parameters auf den Wert 1. Der nachfolgende Ausdruck addiert den aktuellen Gesamtpreis zu dem Wert des Textfeldes *txtZwischensumme*. Das Textfeld wird in diesem Fall also quasi als Varia-



ble verwendet, die den aktuellen Zwischenwert speichert. Erst dann, wenn Access den Seitenfuß ausdruckt und die Berechnung der Zwischensumme damit abgeschlossen ist, wird der Inhalt des Feldes sichtbar.

Damit der richtige Wert berechnet werden kann, muss dem Textfeld *txtZwischensumme* beim Beginn einer neuen Seite immer der Wert 0 zugewiesen werden. Dies erledigt die folgende Ereignisprozedur, die an das *Print*-Ereignis des Seitenkopfes geknüpft ist:

```
Private Sub Seitenkopf0_Print(Cancel As Integer, _
    PrintCount As Integer)

    Me!txtZwischensumme = 0

End Sub
```

Im jetzigen Stadium des Berichts ergibt sich das folgende Problem: Wenn Access den Berichtsfuß der Rechnung und damit den Rechnungsbetrag ausgibt, darf der Seitenfuß mit der Zwischensumme nicht zusätzlich noch erscheinen. Um dies zu unterbinden, muss einfach die *Seitenfuß*-Eigenschaft des Berichts auf *Außer Berichtsfuß* eingestellt werden. Der Bericht ist damit fertig gestellt.

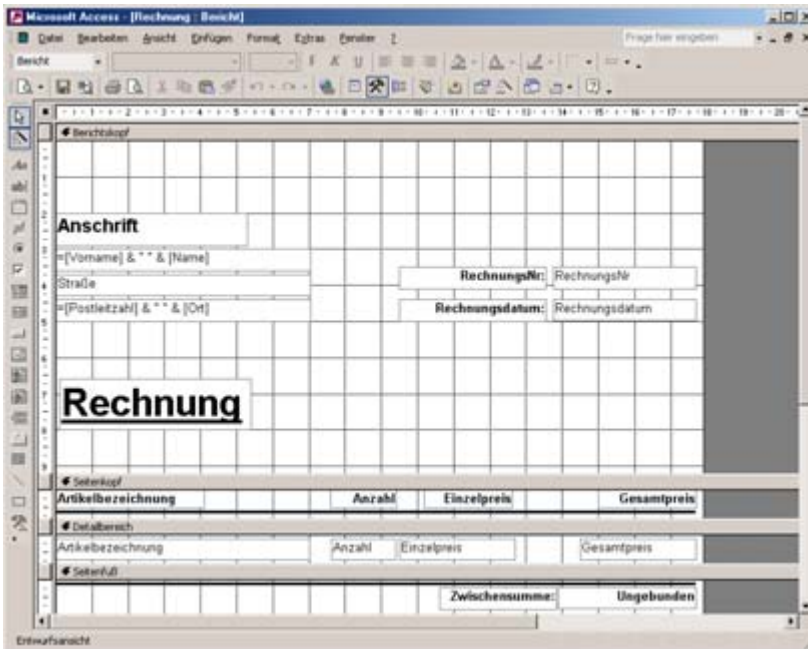


Abb. 8.5

Der Bericht *Rechnung* in der Entwurfsansicht





Zeichnen auf Berichten

Die *Line*-Methode

Eine etwas ungewöhnliche Methode zur Gestaltung von Berichten stellt die Möglichkeit dar, mithilfe einiger spezieller Grafikmethoden auf Berichten zu zeichnen. Eine dieser Methoden ist beispielsweise die *Line*-Methode, die das Zeichnen von Linien oder Rechtecken ermöglicht:

```
<Report-Objekt>.Line [[Step](<x1>, <y1>)] - [Step] _  
(<x2>, <y2>) [, [<Farbe>][,B[F]]]
```

Die einzelnen Argumente dieser Methode haben folgende Bedeutung:

Die Argumente der *Line*-Methode

Argument	Beschreibung
<Bericht-Objekt>	Der Bericht, auf dem die Linie bzw. der Bericht gezeichnet werden soll.
Step	Berechnet die Koordinaten relativ zum aktuellen Grafikkursor.
<x1>, <y1>	Die Koordinatenpunkte des Anfangs der Linie bzw. der linken oberen Ecke des Rechtecks. <x1> bestimmt die horizontale, <y1> die vertikale Komponente.
<x2>, <y2>	Die Koordinatenpunkte des Endpunkts der Linie bzw. der rechten unteren Ecke des Rechtecks. <x2> bestimmt die horizontale, <y2> die vertikale Komponente.
<Farbe>	Die Zeichenfarbe
B	Bewirkt, dass anstelle einer Linie ein Rechteck gezeichnet wird.
F	Wurde die Option B verwendet, kann das Rechteck durch zusätzliche Angabe der Option F mit der ausgewählten Farbe ausgefüllt werden.

Tabelle 8.2

Die x- und y-Koordinaten werden in der Einheit Twips angegeben. Ein Zentimeter entspricht hierbei 567 Twips.

Die *Format*-Ereignisprozedur

Die *Line*-Methode kann beispielsweise wieder dazu verwendet werden, bestimmte Datensätze hervorzuheben. Anstatt ein *Rechteck*-Steuerelement bei Bedarf ein- oder auszublenden, kann das Rechteck nun direkt gezeichnet werden. In der Beispieldatenbank finden Sie einen entsprechenden Bericht mit dem Namen *Kundenliste3*. Die *Format*-Ereignisprozedur des Detailbereichs hat den folgenden Aufbau:



```
Private Sub Detail1_Format(Cancel As Integer, _
    FormatCount As Integer)

Const Rand = 50

If Left(Me!Postleitzahl, 1) = "4" Then

    Me.Line (Rand, Rand)-(Me.Width - Rand * 2, _
        Me.Height - Rand * 2), QBColor(12), B

End If

End Sub
```

Als Erstes erfolgt wieder die Überprüfung, ob die aktuelle Postleitzahl mit der Ziffer 4 beginnt und der Datensatz damit hervorgehoben werden soll. Wenn dies der Fall ist, zeichnet die *Line*-Methode innerhalb des Detailbereichs ein Rechteck. Die Breite und Höhe des aktuellen Bereichs wird hierbei mithilfe der *Width*- und der *Height*-Eigenschaft ermittelt, die sich nicht auf den kompletten Bericht, sondern den aktuellen Detailbereich beziehen.

Abb. 8.6

Die Seitenansicht des
Berichts *Kundenliste3*

Name	Vorname	Ort	Postleitzahl
Müller	Wili	Dusseldorf	45345
Kalt KG		Hamburg	24643
Meier EDV-Beratung		Dusseldorf	53665
Möller AG		Köln	53457
Dr. Zahnlos		Köln	53434
Kaputt OHG		Dusseldorf	35345
Brot	Andreas	Dusseldorf	65644
Schlechtental Verlag GmbH		Köln	53457
Mauerblümchen Abbruch Gmb		Dusseldorf	40595
Müller	Dieter	Dusseldorf	40595
Meier	Fritz	Köln	40595



Auswahl der Linienart

Die Linienbreite und die Art der Linie lässt sich mithilfe der beiden Eigenschaften *DrawWidth* und *DrawStyle* festlegen.

Die *DrawWidth*-Eigenschaft legt die Breite der zu zeichnenden Linie in der Einheit *Pixel* fest. Ein Pixel entspricht hierbei genau einem Punkt auf Ihrem Drucker.

Die *DrawStyle*-
Eigenschaft

Die *DrawStyle*-Eigenschaft legt dagegen fest, ob die Linie z.B. durchgezogen oder gestrichelt gezeichnet wird. Diese Einstellung wird gemäß folgender Tabelle vorgenommen:

Einstellungen der Eigenschaft *DrawStyle*

Einstellung	Linienart
0	Durchgezogen (Standardeinstellung)
1	Striche
2	Punkte
3	Strich-Punkt
4	Strich-Punkt-Punkt
5	Unsichtbar, transparent
6	Unsichtbar, durchgezogen

Tabelle 8.3

Hinweis



Beachten Sie, dass die Einstellungen der Eigenschaften *DrawWidth* und *DrawStyle* nur von VBA aus und nicht beim Entwurf eines Berichts vorgenommen werden können.

Im Bericht *Kundenliste4* wird die folgende Ereignisprozedur dazu verwendet, ein dünnes gestricheltes Rechteck um die betreffenden Kunden zu zeichnen:

```
Private Sub Detail1_Format(Cancel As Integer,  
FormatCount As Integer)
```

```
Const Rand = 50
```

```
    If Left(Me!Postleitzahl, 1) = "4" Then
```

```
        Me.DrawWidth = 2
```

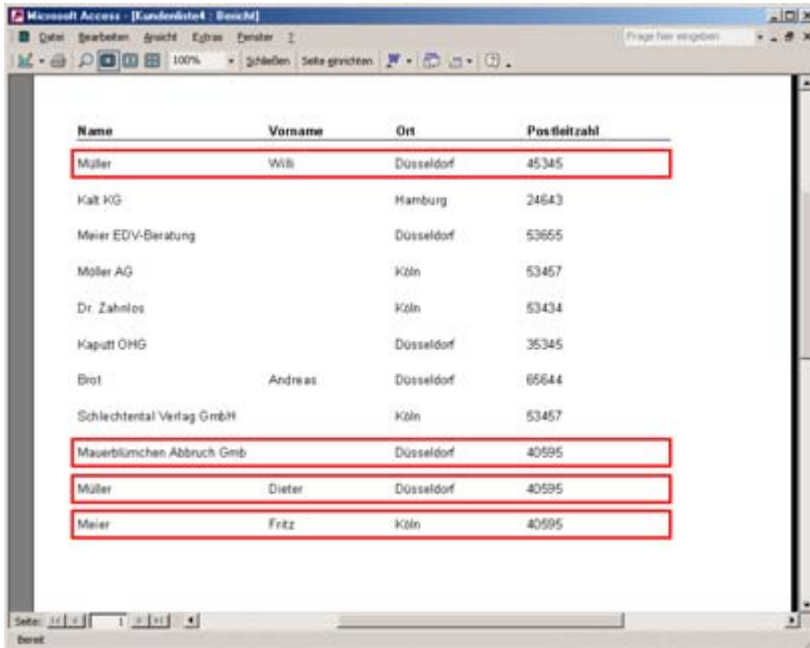
```
        Me.DrawStyle = 1
```



```
Me.Line (Rand, Rand)-(Me.Width - Rand * 2, _
        Me.Height - Rand * 2), QBColor(12), B
```

```
End If
```

```
End Sub
```



Name	Vorname	Ort	Postleitzahl
Müller	Willy	Düsseldorf	45345
Kalt KG		Hamburg	24643
Meier EDV-Beratung		Düsseldorf	53665
Möller AG		Köln	53457
Dr. Zahnlos		Köln	53434
Kaputt OHG		Düsseldorf	35345
Brot	Andreas	Düsseldorf	65644
Schlechtental Verlag GmbH		Köln	53457
Mauerblümchen Abbruch Gmb		Düsseldorf	40595
Müller	Dieter	Düsseldorf	40595
Meier	Fritz	Köln	40595

Abb. 8.7

Die Seitenansicht des Berichts *Kundenliste4*

Beeinflussung der Füllfarbe

Mithilfe der *Line*-Methode lassen sich auch Rechtecke zeichnen, die mit einer Füllfarbe ausgefüllt sind. Zu diesem Zweck muss zusätzlich zur Option *B* die Option *F* angegeben werden, wie das folgende Beispiel zeigt:

```
Me.Line (0, 0)-(Me.Width , Me.Height), QBColor(12), BF
```

Rechteck und Füllfarbe sind in diesem Fall identisch. Wenn die Füllfarbe eine andere als die Rahmenfarbe sein soll, legen Sie vorher die Füllfarbe mithilfe der Eigenschaft *FillColor* fest. Die Option *F* der *Line*-Methode muss dann jedoch entfernt werden.



Die *FillStyle*-Eigenschaft

Neben der Füllfarbe lässt sich mithilfe der *FillStyle*-Eigenschaft bestimmen, ob Access das Rechteck vollständig oder beispielsweise nur schraffiert ausfüllt. Der Eigenschaft kann einer der folgenden Werte zugewiesen werden:

Einstellungen der Eigenschaft *FillStyle*

Einstellung	Beschreibung
0	Undurchsichtig
1	Transparent (Standard)
2	Horizontale Linien
3	Vertikale Linien
4	Steigende Diagonalen
5	Fallende Diagonalen
6	Kreuzschraffur
7	Diagonale Kreuzschraffur

Tabelle 8.4

Das folgende Beispiel zeichnet ein Rechteck mit einem roten Rand und einer grünen Füllfarbe:

```
Me.FillColor = QBColor(10)
Me.FillStyle = 0
Me.Line (0, 0)-(Me.Width , Me.Height), QBColor(12), BF
```

In diesem Beispiel ist die Eigenschaft *FillStyle* auf 0 (Undurchsichtig) gesetzt. Wenn Sie der *FillStyle*-Eigenschaft keinen Wert zuweisen, verwendet Access die Standardeinstellung 1 (Transparent), so dass die Füllfarbe nicht angezeigt wird.

Zeichnen von Kreisen und Punkten

Neben dem Zeichnen von Rechtecken und Linien mithilfe der *Line*-Methode stellt Access noch zwei weitere Methoden zur Verfügung, mit deren Hilfe sich Kreise und einzelne Punkte zeichnen lassen.

Die *Circle*-Methode

Die Methode zum Zeichnen von Kreisen (oder auch Ellipsen und Bögen) trägt den Namen *Circle*:

```
<Bericht-Objekt>.Circle [Step] (<x>, <y>), _
<Radius>[, [<Color>] [, [<Start>] [, [<End>] _
[, <Aspect>] ] ] ]
```

Die einzelnen Argumente besitzen folgende Funktionen:

Argumente der *Circle-Methode*

Argument	Beschreibung
<Bericht-Objekt>	Objekt vom Typ <i>Report</i> , auf dem der Kreis gezeichnet werden soll.
Step	Berechnet die Koordinaten des Kreismittelpunkts relativ zum aktuellen Grafikkursor.
<x>, <y>	Die Koordinaten für den Mittelpunkt des Kreises
<Radius>	Der Radius des Kreise.
<Color>	Die Farbe des Kreisrandes
<Start>	Bestimmt bei Kreisbögen den Startpunkt des Kreisbogens. Der Startpunkt muss in Radianten angegeben werden und im Wertebereich von -2π bis $+2\pi$ liegen. Der Standardwert ist 0.
<End>	Bestimmt bei Kreisbögen den Endpunkt des Kreisbogens. Dieser muss in Radianten angegeben werden und im Wertebereich von -2π bis $+2\pi$ liegen. Der Standardwert ist $+2\pi$.
<Aspect>	Bestimmt das Verhältnis (Höhe/Breite) der beiden Radien einer Ellipse. Bei der Einstellung 1 (Standardwert) wird ein exakter Kreis gezeichnet.

Tabelle 8.5

Das folgende Beispiel markiert wieder alle Kunden, deren Postleitzahlen mit der Ziffer 4 beginnen. Anstelle des Rechtecks wird nun aber eine Ellipse gezeichnet. Sie finden den vollständigen Bericht in der Beispieldatenbank unter dem Namen *Kundenlistes*:

```
Private Sub Detail1_Format(Cancel As Integer, _
    FormatCount As Integer)

Const Rand = 50

Dim Verhaeltnis As Single
Dim Radius As Single

If Left(Me!Postleitzahl, 1) = "4" Then

    Me.DrawWidth = 10
```

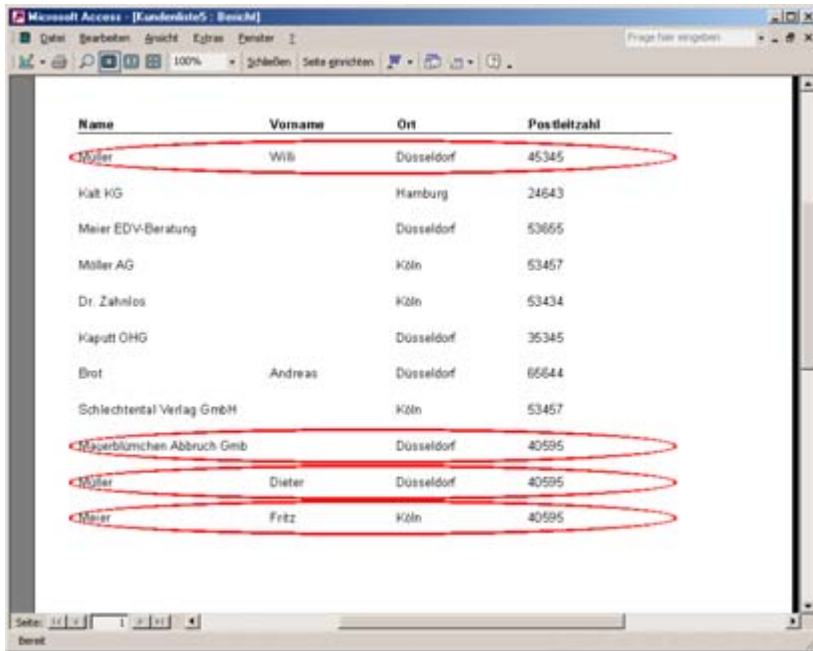


```
Verhältnis = (Me.Height - Rand) / (Me.Width - Rand)
Radius = (Me.Width - Rand) / 2
```

```
Me.Circle ((Me.Width - Rand) / 2, (Me.Height - Rand) _
/ 2), Radius, QBColor(12), , , Verhältnis
```

```
End If
```

```
End Sub
```



Name	Vorname	Ort	Postleitzahl
Müller	Willy	Düsseldorf	45345
Kalt KG		Hamburg	24643
Meier EDV-Beratung		Düsseldorf	53655
Möller AG		Köln	53457
Dr. Zahnlos		Köln	53434
Kaputt OHG		Düsseldorf	35345
Brot	Andreas	Düsseldorf	65644
Schlechtental Verlag GmbH		Köln	53457
Magerblümchen Abbruch Gmb		Düsseldorf	40595
Müller	Dieter	Düsseldorf	40595
Meier	Fritz	Köln	40595

Abb. 8.8

Die Seitenansicht des Berichts *Kundenliste5*

Die *PSet*-Methode

Für die Ausgabe einzelner Punkte ist die Methode *PSet* zuständig:

```
<Bericht-Objekt>.PSet [Step] (<x>, <y>) [, <Color>]
```



Die Argumente haben folgende Bedeutung:

Die Argumente der PSet-Methode

Argument	Beschreibung
<Report-Objekt>	Report-Objekt, auf dem der Punkt gezeichnet werden soll
Step	Berechnet die Koordinaten des Punkts relativ zum aktuellen Grafikcursor.
<x>, <y>	Die Mittelpunktkoordinaten des Punkts, wobei <x> die horizontale und <y> die vertikale Position bestimmt
<Color>	Die Farbe des Punkts

Tabelle 8.6

Positionieren des Grafikcursors

Jeder Bericht hat einen aktuellen Grafikcursor, der einen bestimmten Punkt des Berichts kennzeichnet. Die Position dieses Grafikcursors lässt sich mithilfe der beiden Eigenschaften *CurrentX* und *CurrentY* ermitteln oder auch verschieben.

Im Zusammenhang mit den drei Methoden zum Zeichnen auf Berichten ist Ihnen sicherlich das Schlüsselwort *Step* aufgefallen. Wenn Sie dieses Schlüsselwort angeben, erfolgt die Berechnung der Koordinaten relativ zum aktuellen Grafikcursor. Der folgende Programmausschnitt verschiebt den Grafikcursor beispielsweise auf die Position (500, 1000):

```
Me.CurrentX = 500
Me.CurrentY = 1000
```

Genau an dieser Position wird nun ein Punkt ausgegeben:

```
Me.PSet Step (0, 0)
```

Wenn Sie dagegen die folgende Anweisung angeben, erscheint der Punkt an der Koordinatenposition (600,1100):

```
Me.PSet Step (100, 100)
```

Die *CurrentX*-
und *CurrentY*-
Eigenschaften



Die aktuelle Position des Grafikcursors wird von Access nach dem Zeichnen eines Kreises (*Circle*-Methode) oder einer Linie bzw. eines Rechtecks (*Line* -Methode) automatisch verschoben:

Verschiebung des Grafikcursors

Methode	Neue Position
<i>Circle</i>	Der Grafikcursor wird auf den Mittelpunkt des Kreises verschoben.
<i>Line</i>	Der Grafikcursor wird auf den Endpunkt der Linie bzw. die rechte untere Ecke des Rechtecks verschoben.

Tabelle 8.7

Textausgabe

Die *Print*-Methode

Besonders wichtig ist die Position des Grafikcursors im Zusammenhang mit der *Print*-Methode, die die Ausgabe eines Textes auf dem Bericht ermöglicht. Die Methode besitzt den folgenden Aufbau:

```
<Report-Objekt>.Print [{Spc(<n>) | Tab(<n>)}] _  
    [<Ausdrucksliste>] [{;|,}]
```

Die Argumente dieser Methode haben folgende Bedeutung:

Die Argumente der *Print*-Methode

Argument	Beschreibung
<Report-Objekt>	Der Bericht, auf dem die Textausgabe erfolgen soll
<i>Spc</i> (<n>)	Gibt <n> Leerzeichen aus.
<i>Tab</i> (<n>)	Ermöglicht den Sprung zum <n>-ten Ausgabebereich.
<Ausdrucksliste>	Hier können mehrere, jeweils durch ein Komma, Leerzeichen oder Semikolon getrennte Ausdrücke beliebigen Datentyps angegeben werden, die nacheinander ausgegeben werden sollen. Zwischen den Ausdrücken können auch die Funktionen <i>Spc</i> (<n>) und <i>Tab</i> (<n>) verwendet werden.



Die Argumente der *Print*-Methode (Forts.)

Argument	Beschreibung
<code>[[{;},]]</code>	Bestimmt die Ausgabeposition des nächsten Ausdrucks der <i><Ausdrucksliste></i> . Ein Semikolon bedeutet, dass die nächste Ausgabe direkt hinter dem letzten Zeichen beginnt. Wird ein Komma angegeben, erfolgen weitere Ausgaben am Anfang des nächsten Druckbereichs. Alle 14 Zeichen beginnt ein neuer Druckbereich. Wird keines der Zeichen ; oder , angegeben, erfolgt die nächste Ausgabe in der folgenden Zeile.

Tabelle 8.8

Die Position, an der Access den Text ausgibt, muss vor der Ausführung der *Print*-Methode mithilfe des Grafikcursors festgelegt werden. Die Koordinaten geben hierbei die linke obere Ecke des ersten Ausgabezeichens an.

Nachdem Access den angegebenen Text ausgegeben hat, wird die Position des Grafikcursors auf den Anfang der nächsten Zeile verschoben. Sollten Sie nach der Ausdrucksliste ein Semikolon angeben, verschiebt Access den Grafikcursor lediglich auf das nächste Zeichen. Weitere Ausgaben erfolgen also noch in der gleichen Zeile. Das Komma bedeutet in diesem Fall eine Verschiebung auf den nächsten Druckbereich und bewirkt ebenfalls die Unterdrückung des Zeilenwechsels.

Überschreitet beim Ausdruck auf einen Bericht die Textlänge die Breite des Blattes, wird der überschüssige Text nicht automatisch in die nächste Zeile verschoben, sondern abgeschnitten. Da bei der Ausgabe die Schriftart und die Zeichenhöhe individuell eingestellt werden können, lässt sich für die pro Zeile maximal verfügbare Zeichenanzahl kein fester Wert angeben. Um nun trotzdem die Höhe und Breite einer Textausgabe bei gegebener Schriftart und Schriftgröße herauszufinden, stellt Access die beiden Methoden *TextWidth* und *TextHeight* zur Verfügung:

```
<Breite> = <Report-Objekt>.TextWidth (<Zeichenkette>)
<Höhe> = <Report-Objekt>.TextHeight (<Zeichenkette>)
```

Beide Methoden erwarten eine Zeichenkette, deren Breite bzw. Höhe Sie ermitteln möchten. Bei der Berechnung der Ausmaße berücksichtigt Access die momentan für den Bericht eingestellten Schriftattribute. Das Ergebnis wird in der Einheit Twips ausgegeben.

Auswahl der Schriftattribute

Wie bereits erwähnt, können Schriftart und Schriftbreite mithilfe der beiden Eigenschaften *FontName* und *FontSize* verändert werden. Die Einstellungen wirken sich auf alle Texte aus, die in Zukunft mithilfe der *Print*-Eigenschaft ausgegeben werden. Die folgende *Print*-Ereignisprozedur finden Sie in dem Bericht *Print-Test* der

Die *FontName*- und *FontSize*-Eigenschaften



Beispieldatenbank. Sie ist an den Kopf des Berichts gebunden und gibt hier den Text "Dies ist eine Überschrift" aus. Als Schriftart wird vorher *Arial* und eine Schriftgröße von 40 Punkt ausgewählt:

```
Private Sub Berichtskopf3_Print(Cancel As Integer, _
    PrintCount As Integer)

    Me.FontName = "Arial"
    Me.FontSize = "40"
    Me.CurrentX = 500
    Me.CurrentY = 500
    Me.Print "Dies ist eine Überschrift"

End Sub
```

Abb. 8.9

Der Bericht *Print-Test* in der Seitenansicht



Neben den beiden Eigenschaften *FontName* und *FontSize* können Sie weitere Eigenschaften zur Änderung der Schriftattribute einsetzen. Es folgt eine vollständige Liste:

Eigenschaften zur Veränderung der Schriftattribute

Eigenschaft	Beschreibung
<i>FontName</i>	Bestimmt die Schriftart. Der Name der gewünschten Schriftart muss als Zeichenkette angegeben werden.



Eigenschaften zur Veränderung der Schriftattribute (Forts.)

Eigenschaft	Beschreibung
<i>FontSize</i>	Bestimmt die Schriftgröße. Diese muss in der Einheit <i>Punkt</i> als Zeichenkette angegeben werden.
<i>FontItalic</i>	Bestimmt, ob der Text kursiv (Einstellung <i>True</i>) oder nicht kursiv (Einstellung <i>False</i>) ausgegeben wird.
<i>FontUnderline</i>	Bestimmt, ob der Text unterstrichen (Einstellung <i>True</i>) oder nicht unterstrichen (Einstellung <i>False</i>) ausgegeben wird.
<i>FontBold</i>	Bestimmt, ob der Text fett (Einstellung <i>True</i>) oder nicht fett (Einstellung <i>False</i>) ausgegeben wird.
<i>ForeColor</i>	Bestimmt die Farbe, in der der Text ausgegeben wird. Den entsprechenden Farbcode, den Sie dieser Eigenschaft zuweisen müssen, können Sie z. B. mithilfe der <i>QBColor</i> -Funktion ermitteln.

Tabelle 8.9

Anpassen des Koordinatensystems

Alle Koordinatenangaben, die Sie im Zusammenhang mit den Zeichenfunktionen gemacht haben, waren in der Einheit Twips. Ein Zentimeter entspricht 567 Twips. Auf die Dauer ist es allerdings etwas umständlich, alle Koordinatenpunkte in der Einheit Twips auszudrücken. Access bietet daher die Möglichkeit, für die Bereiche eines Berichts eine andere Einheit wie beispielsweise Zentimeter zu aktivieren. Die entsprechende Einstellung erfolgt hierbei mithilfe der *ScaleMode*-Eigenschaft und ist für alle Bereiche eines Berichts gültig. Der Eigenschaft kann einer der folgenden Werte zugewiesen werden:

Die *ScaleMode*-Eigenschaft

Einstellungen der *ScaleMode*-Eigenschaft

Einstellung	Beschreibung
1	Einheit Twips. 567 Twips entsprechen einem Zentimeter.
2	Einheit Punkt. Ein Punkt entspricht 0,353 mm.
3	Einheit Pixel. Ein Pixel entspricht dem kleinsten darstellbaren Punkt auf dem Bildschirm oder Drucker.
4	Einheit Zeichen. Ein Zeichen entspricht horizontal 120 Twips und vertikal 240 Twips.



Einstellungen der *ScaleMode*-Eigenschaft (Forts.)

Einstellung	Beschreibung
5	Einheit Zoll
6	Einheit Millimeter
7	Einheit Zentimeter

Tabelle 8.10

Sobald Sie der *ScaleMode*-Eigenschaft einen dieser Werte zugewiesen haben, können alle Koordinatenangaben zum Zeichnen auf dem Bereich in der neuen Einheit erfolgen.

Selbstdefinierte Koordinatensysteme

Die *ScaleWidth*- und *ScaleHeight*-Eigenschaften

Neben diesen vordefinierten Einheiten erlaubt es Access jedoch auch, ein eigenes Koordinatensystem nach Ihren Wünschen zu definieren. Dazu muss sowohl für die X- als auch die Y-Achse die Anzahl der Einheiten angegeben werden, in die Sie die Achse unterteilen möchten. Die eigentliche Definition der neuen Einheit wird mithilfe der Eigenschaften *ScaleWidth* (X-Achse) und *ScaleHeight* (Y-Achse) festgelegt. Sie können also für beide Achsen unterschiedliche Unterteilungen definieren. Die folgenden beiden Anweisungen unterteilen den aktuellen Bereich beispielsweise in 100 horizontale und 200 vertikale Einheiten:

```
Me.ScaleWidth = 100
Me.ScaleHeight = 200
```

Auch der Nullpunkt, der sich normalerweise in der linken oberen Ecke eines Bereichs an der Koordinatenposition (0,0) befindet, kann verschoben werden. Dafür sind die beiden Eigenschaften *ScaleLeft* und *ScaleTop* zuständig. Die folgenden beiden Anweisungen verschieben den Nullpunkt beispielsweise in die Mitte des Bereichs:

```
Me.ScaleLeft = 50
Me.ScaleTop = 100
```

Auf Koordinatenpunkte links oberhalb dieses Punktes wird von nun an also mit negativen Werten positioniert.

Sobald Sie ein eigenes Koordinatensystem definieren, wird die *ScaleMode*-Eigenschaft von Access automatisch auf den Wert 0 gesetzt.

Die Definition eines Koordinatensystems kann auch in einem Schritt mithilfe der *Scale*-Methode erfolgen. Diese Methode hat folgenden Aufbau:

```
<Report-Objekt>.Scale [(<x1>,<y1>) - (<x2>,<y2>)]
```

Hinweis





Die Argumente $\langle x1 \rangle$ und $\langle y1 \rangle$ bestimmen hierbei die Werte der linken oberen Ecke des Koordinatensystems, die Argumente $\langle x2 \rangle$ und $\langle y2 \rangle$ dagegen die Werte der rechten unteren Ecke. Um beispielsweise das oben erstellte Koordinatensystem mithilfe der *Scale*-Methode nachzubilden, kann die folgende Anweisung verwendet werden:

```
Me.Scale (-50, -100) - (50, 100)
```





Objektorientierte Programmierung



Klassen in der Praxis	445
Formulare als Objektklassen	459



9



Die Programmiersprache VBA ist im Wesentlichen eine prozedurale Programmiersprache. Dies bedeutet, dass eine Vielzahl von Prozeduren in einem großen Prozedur-Pool zur Verfügung stehen, die sich gegenseitig mit mehr oder weniger strengen Regeln aufrufen können.

Trennung zwischen Daten und Prozeduren

Die große Problematik bei prozeduralen Programmiersprachen ist die Trennung zwischen den Daten, die die Prozeduren bearbeiten, und den eigentlichen Prozeduren. Stellen Sie sich beispielsweise zwei Datensätze vor, die ein Fahrzeug (z. B. einen PKW) beschreiben. Die Speicherung der Daten könnte in Form von Datensätzen einer Tabelle oder auch mithilfe von Variablen innerhalb von VBA realisiert sein.

Ihre Aufgabe ist es nun, eine Prozedur zu entwickeln, die aus dem Tankinhalt des PKWs und dem Benzinverbrauch die durchschnittliche Reichweite berechnet. Die Syntax dieser Prozedur könnte folgendermaßen aussehen:

```
<Reichweite> = ReichweiteBerechnen(<PKW-Datensatz>)
```

Die Prozedur erwartet als Parameter irgendeinen PKW-Datensatz. Sie ist also auf die Bearbeitung nur eines bestimmten Datentyps – nämlich eines PKW-Datensatzes – spezialisiert. Die Prozedur steht damit in einer engen Beziehung zu dem Datentyp *PKW-Datensatz*.

Zusammenlegung von Daten und Prozeduren

Eine Idee der objektorientierten Programmierung ist nun, die Daten und die Prozeduren zur Bearbeitung der Daten zusammenzufassen. Solche Konstrukte werden als *Klassen* oder *Objektklassen* bezeichnet.

Eine Klasse ist die allgemeine Beschreibung eines Objekts. Im PKW-Beispiel könnte eine solche Klasse den Namen PKW tragen. Nach dem Vorbild dieser PKW-Klasse können Sie nun mehrere einzelne PKW-Objekte anlegen, die letztendlich die eigentlichen Fahrzeuge repräsentieren. Diese Unterscheidung zwischen *Klassen* und *Objekten* ist für das Verständnis der objektorientierten Programmierung sehr wichtig. Mithilfe von Klassen definieren Sie die allgemeine Struktur derjenigen Objekte, die Sie mit Ihrer Klasse verwalten möchten. Erst während der Laufzeit Ihres Programms erstellen Sie dann die einzelnen Objekte nach dem Vorbild Ihrer Klasse.

Eigenschaften

In der Praxis tritt nun in der Regel das Problem auf, dass die einzelnen Objekte einer Klasse nicht vollkommen identisch sind. Unterschiedliche PKW-Typen haben beispielsweise einen unterschiedlichen Tankinhalt und damit auch eine unterschiedliche Reichweite. Solche Daten werden in der objektorientierten Terminologie als *Attribute* oder *Eigenschaften* des Objekts bezeichnet. Jedes Objekt einer bestimmten Klasse hat identische Eigenschaften, die jedoch mit unterschiedlichen Werten gefüllt werden können. Dadurch ist es möglich, dass sich jedes Objekt trotz Abstammung von einer gemeinsamen Klasse von anderen Objekten unterscheidet. Der Zustand der einzelnen Objekte ist also unterschiedlich.

Neben Eigenschaften unterstützen Objekte auch *Methoden*. Methoden sind nichts anderes als Befehle an ein bestimmtes Objekt. Methoden bewirken in der Regel, dass sich der Zustand eines Objekts ändert. So könnte ein PKW-Objekt beispielsweise eine Methode namens *Volltanken* zur Verfügung stellen, die den aktuellen Tankinhalt und damit die Eigenschaft *AktuellerTankinhalt* des Objekts auf das Maximum erhöht.



Ein weiteres wichtiges Prinzip der objektorientierten Programmierung ist die *Kapselung* der Daten eines Objekts. Zu einer gut aufgebauten Klasse gehört es, dass von außen kein Zugriff auf die internen Daten des Objekts möglich ist. Jede Zustandsänderung des Objekts muss dagegen über eine Schnittstelle erfolgen, die beispielsweise durch öffentlich zugängliche Methoden oder Eigenschaften realisiert ist. Durch diesen indirekten Zugriff auf die internen Daten hat das Objekt die Möglichkeit, alle von außen gewünschten Änderungen vorher zu kontrollieren. Damit ist sichergestellt, dass das Objekt in keinen ungültigen Zustand versetzt wird. Bei einem PKW-Objekt wäre dies beispielsweise bei einem negativen Tankinhalt der Fall.

Objektkapselung

Klassen in der Praxis

In Access werden Klassen durch so genannte *Klassenmodule* gebildet. Diese unterscheiden sich im Prinzip kaum von den Klassenmodulen, die Sie bereits im Zusammenhang mit Formularen und Berichten kennen gelernt haben. Die hier vorgestellten Klassenmodule sind jedoch eigenständig und damit nicht an ein Formular oder einen Bericht geknüpft. Die Handhabung dieser Klassenmodule erledigen Sie wie bei normalen Standardmodulen über die Kategorie *Module* des Datenbankfensters.

Sie fügen Ihrer Datenbank ein neues Klassenmodul hinzu, indem Sie den Menüpunkt *Einfügen > Klassenmodul* auswählen. Daraufhin öffnet Access die VBA-Entwicklungsumgebung mit dem Modulfenster des Klassenmoduls, das sich vom Äußeren bis auf den Fenstertitel nicht von anderen Modulen unterscheidet.

Abb. 9.1

Ein leeres Klassenmodul



Bei der Definition einer Klasse spielt der Klassenname eine wichtige Rolle. Mithilfe dieses Namens definieren Sie später neue Objekte. Um Ihrem neuen Modul einen Namen zu geben, geben Sie im Eigenschaftenfenster als Eigenschaft (*Name*) den Text *clsPKW* ein.

Klassennamen

Bei dem Kürzel *cls* handelt es sich um eine Namenskonvention, um selbstentwickelte Klassen besser von den bereits in Access eingebauten Klassen (z.B. den DAO- bzw. ADO-Objekten) zu unterscheiden. Diese Konvention erhöht lediglich die Übersichtlichkeit, sie ist aber nicht zwingend vorgeschrieben.



Hinweis



Definition der Klassenattribute

Nachdem Sie Ihre neue Klasse gespeichert haben, ist diese prinzipiell einsatzbereit. Es fehlt allerdings noch die Definition der Klasse. Der erste Schritt ist hierbei in der Regel die Anlage der internen Datenstruktur eines Objekts. Bei einem PKW sind beispielsweise die folgenden Informationen interessant:

- ☐ Fabrikat
- ☐ Typ
- ☐ Zulassungsdatum
- ☐ Farbe
- ☐ maximaler Tankinhalt
- ☐ aktueller Tankinhalt
- ☐ Verbrauch

Um nun diese Informationen Ihrem Klassenmodul als interne Daten hinzuzufügen, geben Sie im Deklarationsbereich folgende Deklarationen an:

```
Private m_Fabrikat As String
Private m_Typ As String
Private m_Zulassungsdatum As Date
Private m_Farbe As String
Private m_MaxTankinhalt As Long "in Litern
Private m_Tankinhalt As Long "in Litern
Private m_Verbrauch As Double "Liter auf 100km
```

Alle sieben Variablen sind mit dem Schlüsselwort *Private* deklariert. So ist sichergestellt, dass keine Prozedur außerhalb des Klassenmoduls auf diese Variablen zugreifen und sie evtl. unkontrolliert verändern kann. Wie bei privaten Variablen üblich, können allerdings die Prozeduren des gleichen Moduls uneingeschränkt auf die Variablen zugreifen und deren Inhalte sowohl auslesen als auch verändern.

Die Namen der einzelnen Variablen beginnen mit dem Präfix *m_*, was ihren besonderen Status kennzeichnet. Das Präfix steht für *Member-Variable*. So werden in der Regel private Variablen eines Klassenmoduls bezeichnet, auf die nur innerhalb der Klasse zugegriffen werden kann. Ähnlich wie bei dem Präfix *cls* von Klassenmodulen handelt es sich auch bei dem Präfix *m_* um eine willkürliche Namenskonvention, die lediglich die Übersichtlichkeit erhöht, aber von Access nicht vorgeschrieben ist.

Hinweis



Die vollständige Klasse finden Sie in der Beispieldatenbank zu diesem Kapitel (*Kapitel9.mdb*).



Einsatz von *Property*-Prozeduren

Im jetzigen Stadium der Klasse ist diese noch nicht sinnvoll einsetzbar. Sie enthält zwar bereits eine interne Datenstruktur, die den Zustand des Objekts repräsentiert. Dieser kann jedoch von außen noch nicht verändert werden, da alle deklarierten Variablen mit dem Attribut *Private* versehen sind.

Eine Lösung des Problems wäre es, die Variablen einfach mit dem Attribut *Public* zu versehen. Solche Variablen erlauben einen unbeschränkten Zugriff, der von dem Objekt nicht registriert oder kontrolliert werden kann. Da dies dem objektorientierten Konzept widerspricht, wird so etwas auch als schlechter Programmierstil bezeichnet.

Doch wie ist nun eine Veränderung des Objektzustands möglich? Die Entwickler von VBA haben zu diesem Zweck so genannte *Property-Prozeduren* vorgesehen, mit denen Sie Ihrer Klasse von außen zugängliche Eigenschaften hinzufügen. Bei *Property*-Prozeduren handelt es sich im Prinzip um normale *Sub*- und *Function*-Prozeduren, die Sie in Ihrem Klassenmodul platzieren. Die primäre Aufgabe der *Property*-Prozeduren ist das kontrollierte Auslesen und Verändern einer Member-Variablen.

Property-Prozeduren treten in der Regel immer paarweise auf. Die erste Prozedur ist für den Lesezugriff auf eine Eigenschaft, die zweite für den Schreibzugriff zuständig. Zusammen bilden die beiden *Property*-Prozeduren eine Eigenschaft Ihrer Klasse mit Lese- und Schreibzugriff.

Der Aufbau einer *Property*-Prozedur ist fast mit dem Aufbau einer *Sub*- oder *Function*-Prozedur identisch. Der Kopf einer *Property*-Prozedur für den Lesezugriff auf eine Eigenschaft hat den folgenden vereinfachten Aufbau:

```
Public Property Get <Eigenschaftsname>() As <Datentyp>

    <Anweisungen>

End Property
```

Die *Property Get*-Prozedur hat im Prinzip den gleichen Aufbau wie eine *Function*-Prozedur. Das Schlüsselwort *Public* kennzeichnet in diesem Zusammenhang, dass die Prozedur öffentlich ist und somit von außen aufgerufen werden kann. Natürlich ist es auch möglich, eine *private Property*-Prozedur zu definieren, die dann nur innerhalb der Klasse genutzt werden kann. Dies ist aber nur in seltenen Fällen sinnvoll.

Das Gegenstück zur *Property Get*-Prozedur ist die *Property Let*-Prozedur. Sie regelt den schreibenden Zugriff auf die Eigenschaft. Der vereinfachte Aufbau einer solchen Prozedur ähnelt einer *Sub*-Prozedur und lautet wie folgt:

```
Public Property Let <Eigenschaftsname>(vNewValue As _
Variant)

    <Anweisungen>

End Property
```

Property-Prozeduren

Aufbau von
Property-Prozeduren





Für die Definition einer Eigenschaft ist es hierbei wichtig, dass die beiden paarweise auftretenden *Property*-Prozeduren jeweils exakt den gleichen Namen haben.

Im Zusammenhang mit der PKW-Klasse ist es beispielsweise erforderlich, das Fabrikat eines Fahrzeugs (bzw. Objekts) sowohl zu ermitteln als auch zu verändern. Zum Zugriff auf die entsprechende Member-Variable *m_Fabrikat* muss also eine neue Eigenschaft angelegt werden, für die sich der Name *Fabrikat* anbietet. Um diese Eigenschaft anzulegen, wählen Sie den Menübefehl *Einfügen > Prozedur* aus. Im Dialogfenster *Prozedur hinzufügen* geben Sie nun den Namen der Eigenschaft – also *Fabrikat* – an. Außerdem markieren Sie die Option *Property* und aktivieren den Geltungsbereich *Public*. Nach Ihrer Bestätigung mit *OK* fügt Access dem Klassenmodul die folgenden beiden *Property*-Prozeduren hinzu:

```
Public Property Get Fabrikat() As Variant
```

```
End Property
```

```
Public Property Let Fabrikat(ByVal vNewValue As Variant)
```

```
End Property
```

Hinweis



Den Namen und Datentyp des Parameters *nNewValue* können Sie beliebig an Ihre Bedürfnisse anpassen.

Die erste Prozedur wird von Access immer dann aufgerufen, wenn Sie den Inhalt der Eigenschaft auslesen möchten. In diesem speziellen Fall muss die Prozedur *Fabrikat* also den Inhalt der Member-Variablen *m_Fabrikat* als Ergebnis zurückliefern. Die vollständige Prozedur hat damit den folgenden Aufbau:

```
Public Property Get Fabrikat() As Variant
```

```
    Fabrikat = m_Fabrikat
```

```
End Property
```

Die zweite *Property*-Prozedur wird von Access bei einem Schreibzugriff aufgerufen, wobei Access den neuen Wert als Parameter übergibt:

```
Public Property Let Fabrikat(ByVal vNewValue As Variant)
```

```
    If Not IsNull(vNewValue) Then
```

```
        m_Fabrikat = vNewValue
```

```
    End If
```

```
End Property
```



Die Besonderheit dieser Prozedur besteht darin, dass sie den neuen Wert der Eigenschaft vor der Zuweisung an die Member-Variable auf den Wert *NULL* überprüft und ggf. keine Änderung vornimmt.

Neben der *Property Let*-Prozedur stellt VBA noch einen dritten *Property*-Prozedurtyp namens *Property Set* zur Verfügung. Dieser muss eingesetzt werden, wenn das Ergebnis der Eigenschaft kein Grunddatentyp, wie z. B. *Variant* oder *Integer*, sondern ein Objektverweis wie *Form* oder *Recordset* ist.



Hinweis

Einsatz des Klassenmoduls

Nachdem Ihre Klasse nun bereits ein öffentliches Element – nämlich eine Eigenschaft namens *Fabrikat* – aufweist, steht dem Einsatz nichts mehr im Wege. Zum Test Ihrer Klasse öffnen Sie zunächst ein neues Standardmodul und legen hier eine Testprozedur an. Im ersten Schritt deklarieren Sie eine Variable mit Ihrer Klasse:

```
Dim MeinPKW As clsPKW
```

Die Variable *MeinPKW* ist nun in der Lage, einen Verweis auf ein Objekt des Typs *clsPKW* anzulegen. Im nächsten Schritt muss ein solches Objekt angelegt und der Variablen *MeinPKW* ein Verweis auf dieses Objekt zugewiesen werden. Dies erledigt die folgende Anweisung:

```
Set MeinPKW = New clsPKW
```

Beide Schritte, nämlich die Deklaration und die Anlage bzw. Zuweisung des Objekts, lassen sich auch in einer einzelnen Anweisung zusammenfassen:

```
Dim MeinPKW As New clsPKW
```

Nachdem Sie das neue Objekt angelegt haben, können Sie auf die Eigenschaft *Fabrikat* zugreifen. Die folgende Anweisung weist der Eigenschaft das Fabrikat *BMW* zu:

```
MeinPKW.Fabrikat = "BMW"
```

Sobald Access während der Ausführung der Testprozedur auf diese Anweisung stößt, ruft es die *Property Let*-Prozedur des Klassenmoduls *clsPKW* auf und übergibt die Zeichenkette *BMW* als Parameter *vNewValue*.

Wenn Sie die Eigenschaft dagegen auslesen, ruft Access die *Property Get*-Prozedur auf, was beispielsweise bei folgender Anweisung der Fall ist:

```
MsgBox MeinPKW.Fabrikat
```

Diese Anweisung zeigt das Fabrikat des PKWs in einem Meldungsfenster an. Die entsprechende *Property Get*-Prozedur liest zu diesem Zweck den Inhalt der Member-Variablen *m_Fabrikat* aus und gibt ihn als Ergebnis der Prozedur zurück.

Deklaration

Anlegen eines Objekts

Zusammenfassung beider Schritte

Schreibzugriff

Lesezugriff





Deklaration einer zweiten Variablen

Natürlich ist es möglich, nicht nur eine, sondern beliebig viele Variablen des Typs *clsPKW* zu deklarieren. Das folgende Beispiel deklariert beispielsweise drei PKW-Objekte, weist diesen unterschiedliche Fabrikate zu und gibt sie anschließend im Meldungsfenster aus:

```
Sub Testprozedur()  
  
    Dim MeinPKW1 As New clsPKW  
    Dim MeinPKW2 As New clsPKW  
    Dim MeinPKW3 As New clsPKW  
  
    MeinPKW1.Fabrikat = "BMW"  
    MeinPKW2.Fabrikat = "VW"  
    MeinPKW3.Fabrikat = "Mercedes"  
  
    MsgBox MeinPKW1.Fabrikat  
    MsgBox MeinPKW2.Fabrikat  
    MsgBox MeinPKW3.Fabrikat  
  
End Sub
```

Anhand dieser Prozedur wird eines besonders deutlich: Alle drei Objekte besitzen einen unterschiedlichen Satz an Daten. Der Grund dafür ist, dass Access bei der Anlage eines neuen Objekts vom Typ *clsPKW* für die sieben Member-Variablen jeweils neue Speicherbereiche reserviert. Jedes Objekt hat damit z. B. eine individuelle Variable *m_Fabrikat* und einen individuellen Zustand, der sich von dem anderer Objekte unterscheiden kann.

Hinweis



Für jedes Objekt muss Access im Hauptspeicher einen bestimmten Speicherplatz reservieren. Wenn Sie ein Objekt nicht mehr benötigen, sollten Sie den Speicherplatz freigeben. Dies erreichen Sie, indem Sie der betreffenden Objektvariablen das Schlüsselwort *Nothing* zuweisen:

```
Set MeinPKW1 = Nothing
```

Sollte die Lebensdauer einer Objektvariablen beendet sein, entfernt Access die betreffenden Objekte automatisch aus dem Hauptspeicher. Dies ist beispielsweise bei einer lokalen Objektvariablen einer Prozedur der Fall, die beendet wird. Aus diesem Grund ist eine explizite Zuweisung des Schlüsselworts *Nothing* in der obigen *Testprozedur* überflüssig, da Access das Entfernen der drei Objekte *MeinPKW1* bis *MeinPKW2* automatisch bei der Beendigung der Prozedur erledigt.

Hinweis



Genau wie bei den fest in Access eingebauten Klassen, wie beispielsweise *Form* oder *Recordset*, können Sie auch Ihre selbstdefinierten Klassen mit dem Objektkatalog analysieren. Dies gibt Ihnen bei umfangreichen Klassen die nötige Übersicht über alle Eigenschaften und Methoden.



Eigenschaften ohne Member-Variablen

Wenn Sie Ihrer Klasse mithilfe zweier *Property*-Prozeduren eine neue Eigenschaft hinzufügen, müssen diese nicht unbedingt auf eine Member-Variable der Klasse zugreifen. Es ist beispielsweise möglich, den Wert der Eigenschaft aus dem Inhalt mehrerer Member-Variablen zu berechnen.

Beim PKW-Beispiel würde dies für eine neue Eigenschaft namens *Reichweite* in Frage kommen, die die aktuelle Reichweite eines PKWs aus dem aktuellen Tankinhalt und dem durchschnittlichen Verbrauch berechnet. Die entsprechende *Property*-Prozedur zum Lesen der Eigenschaft hat den folgenden Aufbau:

```
Public Property Get Reichweite() As Double "in km

    If m_Verbrauch > 0 Then

        Reichweite = m_Tankinhalt / m_Verbrauch

    End If

End Property
```

Die Prozedur überprüft zunächst, ob die Variable *m_Verbrauch* einen Wert > 0 aufweist. Dies ist notwendig, um einen Laufzeitfehler bei einer Division durch Null zu vermeiden. Anschließend berechnet sie die Reichweite und gibt diese als Ergebnis der Prozedur zurück.

In der Praxis kommt es häufig vor, dass Eigenschaften wie die *Reichweite*-Eigenschaft nur beim Lesezugriff sinnvoll einsetzbar sind. Um solche Eigenschaften zu erzeugen, implementieren Sie einfach nur die betreffende *Property Get*-Prozedur. Wenn nun trotzdem ein Schreibzugriff erfolgt, meldet Access einen Laufzeitfehler.

Nur Lesezugriff

Natürlich können Sie diese Vorgehensweise auch bei Eigenschaften einsetzen, die auf Member-Variablen zugreifen, um diese generell vor Schreibzugriff von außen zu schützen. Dies ist beispielsweise bei der Member-Variable *m_Tankinhalt* der Fall, die von außen nicht geändert werden darf (zum Tanken kommen wir gleich). Die entsprechende *Property*-Prozedur hat den folgenden Aufbau:

```
Public Property Get Tankinhalt() As Variant

    Tankinhalt = m_Tankinhalt

End Property
```





Definition von Methoden

Methoden haben den Charakter eines Befehls an ein bestimmtes Objekt. Die Implementierung einer öffentlichen Methode erfolgt durch eine normale *Sub*-Prozedur, die Sie mit dem Attribut *Public* in Ihrem Klassenmodul anlegen. Natürlich dürfen Sie hierbei auch eine Parameterliste angeben.

Die folgende Prozedur fügt der PKW-Klasse eine Methode namens *Tanken* hinzu. Als Parameter erwartet die Methode die getankte Treibstoffmenge:

```
Public Sub Tanken(Menge As Variant)

    If Menge = "Voll" Then

        "Volltanken
        m_Tankinhalt = m_MaxTankinhalt

    Else

        "Läuft der Tank über?
        If m_Tankinhalt + Menge > m_MaxTankinhalt Then

            "Laufzeitfehler auslösen
            Err.Raise vbObjectError + 1000,
            "clsPKW.Tanken", _
                "Der Tank läuft über!"

        Else

            m_Tankinhalt = m_Tankinhalt + Menge

        End If

    End If

End Sub
```

Die Methode überprüft zunächst, ob der Parameter der Methode den Text *Voll* enthält. Sollte dies der Fall sein, setzt die Methode den aktuellen Tankinhalt auf den Wert *m_MaxTankinhalt*. Der PKW ist damit vollgetankt.

Wenn dies nicht der Fall ist, interpretiert die Methode den Parameter *Menge* als Anzahl der getankten Liter. Sie überprüft zunächst, ob die gewünschte Menge überhaupt noch im Benzintank Platz findet. Wenn nicht, löst sie mittels *Raise*-Methode einen Laufzeitfehler aus, den die aufrufende Prozedur verarbeiten muss. Andernfalls erhöht die Methode den Tankinhalt um die angegebene Treibstoffmenge.

Bei einer Objektvariablen namens *MeinPKW1*, der bereits ein gültiges PKW-Objekt zugewiesen wurde, könnte die Methode nun folgendermaßen eingesetzt werden:

```
MeinPKW1.Tanken ("Voll")
```



Anstelle von *Sub*-Prozeduren können Sie innerhalb von Klassenmodulen auch *Function*-Prozeduren einsetzen, die später ebenfalls als Methoden der Klasse erscheinen. Solche Methoden können jedoch ein Ergebnis zurückliefern. Die folgende Methode gibt beispielsweise den Wert *True* zurück, wenn eine bestimmte Menge an Treibstoff noch in den Tank des PKWs passen würde:

Methoden mit
Rückgabewert

```
Public Function IstTankmengeMöglich(Menge As Variant)
As Boolean

    If m_Tankinhalt + Menge > m_MaxTankinhalt Then

        IstTankmengeMöglich = False

    Else

        IstTankmengeMöglich = True

    End If

End Function
```

Der Einsatz der Methode könnte wie folgt aussehen:

```
With MeinPKW1

    If .IstTankmengeMöglich(10) Then

        .Tanken 10

    End If

End With
```

Bei Bedarf kann auf den Einsatz der *With*-Anweisung verzichtet werden:

```
If MeinPKW1.IstTankmengeMöglich(10) Then

    MeinPKW1.Tanken 10

End If
```

Bei der Implementierung der Funktionalität kommt es häufig vor, dass bestimmte Methoden nur einem internen Zweck dienen und von außen nicht zugänglich sein sollen. Definieren Sie solche Methoden einfach mit dem Attribut *Private*.



Hinweis





Initialisierung und Terminierung

Wie bereits erwähnt, legt Access bei der Erzeugung eines neuen Objekts einen kompletten Satz Member-Variablen an, so dass jedes Objekt einen individuellen Zustand aufweisen kann. Access weist hierbei allen Member-Variablen ihren Initialisierungswert zu. Numerische Variablen erhalten also z. B. den Wert 0, und Zeichenketten werden mit einem Leerstring versehen.

Diese Art der Initialisierung ist bei vielen Objekten jedoch nicht zufrieden stellend. So wäre es beispielsweise sinnvoll, die Member-Variable *m_Verbrauch* der PKW-Klasse direkt bei der Erzeugung des Objekts auf einen vernünftigen Wert zu setzen (z. B. 8 Liter auf 100 km).

Anlage einer Ereignisprozedur

Um solche oder auch andere Initialisierungsaufgaben zu erledigen, löst Access beim Erzeugen eines Objekts ein Ereignis namens *Initialize* aus. Es handelt sich hierbei um ein ganz normales Ereignis, bei dem allerdings keine Ereignisseigenschaft zur Verfügung steht. Dies bedeutet, dass Sie eine entsprechende Ereignisprozedur zur Reaktion auf dieses Ereignis nicht von einem Eigenschaftsfenster, sondern ausschließlich vom Modulfenster aus anlegen können. Wählen Sie zu diesem Zweck aus der Liste *Objekt* im Kopf des Modulfensters den Eintrag *Class* und im rechts daneben liegenden Kombinationsfeld das Ereignis *Initialize* aus. Access legt daraufhin die folgende Ereignisprozedur an:

```
Private Sub Class_Initialize()  
End Sub
```

Innerhalb dieser Ereignisprozedur erledigen Sie nun die Initialisierung der gewünschten Member-Variablen:

```
Private Sub Class_Initialize()  
  
    m_Verbrauch = 8  
    m_Zulassungsdatum = Date "Heute"  
    m_MaxTankinhalt = 80  
  
End Sub
```

Wenn Sie nun ein neues Objekt der Klasse *clsPKW* anlegen, löst Access als Erstes das *Initialize*-Ereignis aus. Die drei Member-Variablen erhalten damit direkt zu Beginn sinnvolle Werte.

Terminate-Ereignis

Das Gegenstück zum *Initialize*-Ereignis ist das *Terminate*-Ereignis. Access löst dieses Ereignis aus, sobald ein Objekt aus dem Speicher entfernt wird. Dies ist beispielsweise der Fall, wenn Sie der betreffenden Objektvariablen das Schlüsselwort *Nothing* zuweisen.

Die Anlage einer entsprechenden Ereignisprozedur erfolgt auf die gleiche Weise wie beim *Initialize*-Ereignis. Wählen Sie aus der Ereignisliste im Kopf des Modulfensters einfach den Eintrag *Terminate* aus:



```
Private Sub Class_Terminate()  
End Sub
```

Im Gegensatz zum *Initialize*-Ereignis kommt das *Terminate*-Ereignis nur in relativ wenigen Fällen zum Einsatz. In der Regel wird das *Terminate*-Ereignis dazu verwendet, nicht mehr benötigte Ressourcen wieder freizugeben. Dies könnte beispielsweise ein geöffneter DDE-Kanal sein, den Sie in der *Initialize*-Ereignisprozedur aufgebaut haben.

Im Zusammenhang mit der PKW-Klasse ist der Einsatz des *Terminate*-Ereignisses nicht erforderlich.

Initialisierung mittels *Create*-Methode

In der objektorientierten Hochsprache C++ ist es möglich, direkt bei der Neuanlage eines Objekts wichtige Informationen zu übergeben, die bei der Initialisierung des Objekts benötigt werden. Die Initialisierungsprozedur wird in diesem Zusammenhang auch als *Konstruktor* bezeichnet. Leider stellt VBA kein passendes Gegenstück zum Konstruktor zur Verfügung.

Um die Funktion des Konstruktors nachzubilden, muss die Klasse mit einer Methode versehen werden, die der Anwender des Objekts direkt nach der Initialisierung aufrufen sollte. Als Name für diese Methode bietet sich *Create* an.

Die folgende *Create*-Methode setzt ein neu erstelltes PKW-Objekt in einen definierten Zustand, indem es die wichtigsten Member-Variablen der PKW-Klasse mit Werten füllt.

```
Public Sub Create(Fabrikat As String, Typ As String _  
    , Farbe As String)  
  
    If Len(Fabrikat) = 0 Or Len(Typ) = 0 _  
        Or Len(Farbe) = 0 Then  
  
        Err.Raise vbObjectError + 1002, "clsPKW.Create" _  
            , "Bitte geben Sie das Fabrikat, den Typ _  
            und die Farbe an!"  
  
    Else  
  
        m_Fabrikat = Fabrikat  
        m_Typ = Typ  
        m_Farbe = Farbe  
  
    End If  
  
End Sub
```





Die *Create*-Methode sollte direkt nach der Erzeugung eines neuen Objekts aufgerufen werden:

```
Dim MeinPKW As New clsPKW  
MeinPKW.Create "Porsche", "Boxter", "Dunkelblau"
```

Das Problem bei dieser Lösung besteht darin, dass Access den Aufruf der *Create*-Methode nicht zwingend vorschreibt, wie es z. B. beim Konstruktor von C++ der Fall ist. Es könnte also sein, dass der Benutzer Ihrer Klasse die *Create*-Methode vergisst und das Objekt damit einen unerwünschten oder sogar ungültigen Zustand hat.

Einsatz des *Collection*-Objekts

In der Praxis kommt es häufig vor, dass verschiedene Objektklassen in einer Objekthierarchie angeordnet sind. Wichtigstes Element zum Aufbau solcher Objekthierarchien sind die so genannten *Auflistungen*, die Sie bereits von den starr vorgegebenen Access-Objekten kennen.

Auflistungen sind nichts anderes als normale Klassen, die auf die Verwaltung einer bestimmten anderen Objektklasse spezialisiert sind. So könnte beispielsweise die Auflistung *Parkhaus* mehrere PKW-Objekte aufnehmen und verwalten.

Speziell zu diesem Zweck stellt Access eine vordefinierte Klasse namens *Collection* zur Verfügung, die die Verwaltung von Objekten erheblich erleichtert. Die *Collection*-Objektklasse ist auf keinen speziellen Objekttyp beschränkt und damit universell einsetzbar.

Hinzufügen von Objekten

Die *Add*-Methode

Der neu angelegten Auflistung können Sie nun Objekte hinzufügen. Dazu stellen *Collection*-Objekte die *Add*-Methode zur Verfügung, deren einfachste Syntax den folgenden Aufbau hat:

```
<Collection-Objekt>.Add <Item>
```

Unter *<Item>* muss ein Verweis auf ein beliebiges Objekt angegeben werden.

Das folgende Beispiel erzeugt fünf Objekte des Typs *clsPKW* und weist diese einem *Collection*-Objekt zu:

```
Sub CollectionTest()  
  
Dim EinPKW As clsPKW  
Dim Parkhaus As Collection  
Dim i As Integer  
  
    "Collection-Objekt erzeugen  
    Set Parkhaus = New Collection
```





```
"Fünf PKW-Objekte erzeugen und im Parkhaus einparken  
For i = 1 To 5
```

```
    Set EinPKW = New clsPKW  
    EinPKW.Create "BMW", "Nummer " & CStr(i), "Rot"  
    Parkhaus.Add EinPKW
```

```
Next i
```

```
End Sub
```

Die Prozedur deklariert zunächst eine neue *Collection*-Objektvariable namens *Parkhaus* und weist dieser anschließend mittels *New*-Operator ein neu erzeugtes *Collection*-Objekt zu. Das Objekt ist damit einsatzbereit.

Die folgende Schleife legt schließlich fünf neue *PKW*-Objekte an und fügt diese mittels *Add*-Methode dem *Collection*-Objekt hinzu.

Die Beispielprozedur greift auf die Eigenschaft *Typ* der *PKW*-Objekte zu. Die Implementierung dieser Eigenschaft finden Sie in der Beispieldatenbank zu diesem Kapitel.



Hinweis

Der Zugriff auf die einzelnen Objekte kann entweder mittels einer *For Each*-Schleife oder – wie bei Auflistungen üblich – über einen Index erfolgen. Anders als bei den Standardauflistungen von Access hat das erste Element hierbei allerdings den Index 1. Das folgende Beispiel verwendet eine *For...Next*-Schleife, um das Fabrikat und den Typ aller Fahrzeuge in fünf Meldungsfenstern anzuzeigen:



Zugriff auf die Elemente

```
"Welche PKWs stehen im Parkhaus?"
```

```
For Each EinPKW In Parkhaus
```

```
    MsgBox "Fabrikat: " & EinPKW.Fabrikat _  
        & ", Typ: " & EinPKW.Typ
```

```
Next EinPKW
```

Im Zusammenhang mit einer *For...Next*-Schleife kommt die Eigenschaft *Count* der *Collection*-Klasse zum Einsatz, die die Anzahl der Objekte ermittelt:

```
"Welche PKWs stehen im Parkhaus? (Variante 2)"
```

```
For i = 1 To Parkhaus.Count
```

```
    MsgBox "Fabrikat: " & Parkhaus(i).Fabrikat _  
        & ", Typ: " & Parkhaus(i).Typ
```

```
Next i
```





Als Alternative zum Index stellt Ihnen Access auch die Möglichkeit zur Verfügung, auf ein bestimmtes Objekt über einen Schlüssel – vergleichbar mit einem Objekt-namen – zuzugreifen. Voraussetzung dafür ist, dass Sie jedem einzelnen Objekt bereits beim Hinzufügen über die *Add*-Methode einen Namen zuordnen. Die *Add*-Methode stellt zu diesem Zweck einen zweiten optionalen Parameter zur Verfügung:

```
<Collection-Objekt>.Add <Item> [, <Key>]
```

Unter *<Key>* geben Sie eine Zeichenkette an, die für die Auflistung eindeutig ist. Das folgende Beispiel fügt der Auflistung *Parkhaus* fünf Fahrzeuge hinzu und weist diesen die Schlüssel *Fahrzeug1* bis *Fahrzeug5* zu:

```
Sub CollectionTest2()  
  
Dim EinPKW As clsPKW  
Dim Parkhaus As Collection  
Dim i As Integer  
  
    "Collection-Objekt erzeugen  
    Set Parkhaus = New Collection  
  
    "Fünf PKW-Objekte erzeugen und im Parkhaus einparken  
    For i = 1 To 5  
  
        Set EinPKW = New clsPKW  
        EinPKW.Create "BMW", "Nummer " & CStr(i), "Rot"  
        Parkhaus.Add EinPKW, "Fahrzeug" & CStr(i)  
  
    Next i  
  
End Sub
```

Der Zugriff auf ein Objekt kann nun alternativ zum Index auch über den Namen erfolgen. Die folgende Programmzeile zeigt den Typ des Fahrzeugs *Fahrzeug2* in einem Meldungsfenster an:

```
MsgBox Parkhaus!Fahrzeug2.Type
```

Entfernen von Objekten

Das Entfernen eines Elements aus einer benutzerdefinierten Auflistung bewerkstelligen Sie mittels der *Remove*-Methode:

```
<Collection-Objekt>.Remove(<Index>)
```

Unter *<Index>* geben Sie entweder den numerischen Index eines Objekts oder den Schlüssel an. Nachdem Access ein Element gelöscht hat, rücken alle nachfolgenden Elemente der Auflistung automatisch auf. Die Indizes werden für diese Elemente also neu vergeben.

Die *Remove*-
Methode



Die folgende Tabelle enthält noch einmal eine Übersicht über die Methoden und Eigenschaften der *Collection*-Klasse:

Elemente der *Collection*-Klasse

Element	Beschreibung
<i>Add</i> -Methode	Fügt der Auflistung ein Objekt hinzu.
<i>Item</i> -Methode	Ermöglicht den Zugriff auf ein Objekt über dessen Index.
<i>Remove</i> -Methode	Entfernt ein Objekt.
<i>Count</i> -Methode	Ermittelt die Anzahl der Objekte.

Tabelle 9.1

Formulare als Objektklassen

Jedem Formular kann innerhalb von Access ein VBA-Modul zugeordnet werden, das nichts anderes als ein normales Klassenmodul ist. Dementsprechend können auch Formular- und Berichtsmodule mit Eigenschaften und Methoden ausgestattet werden. Auch das Anlegen mehrerer Objekte einer Formularklasse ist möglich. Damit öffnen Sie ein Formular mehrmals auf Ihrem Bildschirm, um so z. B. mehrere Datensätze gleichzeitig zu begutachten.

Alle vorgestellten Techniken lassen sich ohne Einschränkungen auch auf Berichte übertragen.

Methoden und Eigenschaften in Formularmodulen

Die Definition von Methoden in Formularmodulen bewerkstelligen Sie auf die gleiche Weise wie bei Klassenmodulen. Das Formular *Kunden* der Beispieldatenbank *Kapitel9* enthält beispielsweise eine öffentliche Prozedur mit dem Namen *Schließen*:

```
Public Sub Schließen()

    DoCmd.Close acForm, Me.Name

End Sub
```

Die Prozedur erledigt nichts anderes, als das aktuelle Formular – in diesem Fall also das Kundenformular – zu schließen. Von einem anderen Modul aus können Sie diese Prozedur nun so aufrufen, als wäre sie eine Methode des *Form*-Objekts *Kunden*:

```
Forms!Kunden.Schließen
```





Alle Methoden eines Formulars können Sie natürlich auch von dem Klassenmodul des Formulars aus nutzen. Wenn Sie von hier aus die *Schließen*-Methode ausführen möchten, ist die Angabe des Objektverweises allerdings nicht erforderlich. Es genügt also, wenn Sie hier lediglich

Schließen

oder

Me.Schließen

angeben. Gleiches gilt natürlich auch für "normale" Klassenmodule, die keinem Formular oder Bericht zugeordnet sind.

Property-Prozeduren

Auch die Definition von Eigenschaften ist durch den Einsatz von *Property*-Prozeduren möglich. Im Zusammenhang mit Formularmodulen müssen Sie bei der Namensvergabe jedoch beachten, dass keine Namenskonflikte mit Steuerelementnamen auftreten.

Die folgende *Property Get*-Prozedur fügt dem Formular *Kunden* eine neue Eigenschaft namens *Kundennummer* hinzu, über die Sie die Primärschlüsselnummer des momentan angezeigten Kunden ermitteln:

```
Public Property Get Kundennummer() As Variant
```

```
    Kundennummer = Me![KundenNr]
```

```
End Property
```

Die Prozedur liest die Kundennummer einfach aus dem Steuerelement *KundenNr* aus und gibt dieses als Ergebnis der Prozedur zurück.

Das Gegenstück, nämlich die passende *Property Let*-Prozedur, hat die Aufgabe, im Formular den gewünschten Kunden auszuwählen:

```
Public Property Let Kundennummer(ByVal vNewValue  
As Variant)
```

```
    Me.SetFocus
```

```
    Me.Controls![KundenNr].SetFocus
```

```
    DoCmd.FindRecord FindWhat:=vNewValue, _  
                    Match:=acEntire, _  
                    MatchCase:=True, _  
                    Search:=acSearchAll, _  
                    SearchAsFormatted:=False, _  
                    OnlyCurrentField:=True, _  
                    FindFirst:=True
```

```
End Property
```



Über die ersten beiden Anweisungen wird das *Kundenformular* sowie das Steuerelement *KundenNr* aktiviert. Anschließend erfolgt eine Suche nach der neuen Kundennummer mittels der *FindRecord*-Methode.

Die fertige Eigenschaft können Sie nun verwenden, um im Kundenformular komfortabel die aktuelle Kundennummer zu ermitteln oder zu verändern. Der erste Beispielausdruck ermittelt die Nummer des aktuellen Kunden; der zweite Ausdruck macht den Kunden mit der Nummer 11 zum aktuellen Datensatz:

```
Debug.Print Forms!Kunden.Kundennummer  
Forms!Kunden.Kundennummer = 11
```

Anlegen von Formularinstanzen

Wenn Sie mithilfe der *OpenForm*-Methode (Deutsch: *ÖffnenFormular*-Aktion) ein Formular öffnen, legt Access nach dem Vorbild der Formularobjektklasse – die Sie über den Formular-Editor bearbeiten – eine Objektinstanz des Formulars an. Diese Instanz wird auch als *Standardinstanz* bezeichnet.

Auf Basis der Objektklasse eines Formulars kann jedoch nicht nur eine, sondern mehrere Formularinstanzen geöffnet werden. Dies ist vergleichbar mit mehreren PKW-Objekten, die Sie auf der Grundlage der Klasse *clsPKW* anlegen.

Durch ein mehrfach geöffnetes Formular haben Sie beispielsweise die Möglichkeit, mehrere Datensätze gleichzeitig auf dem Bildschirm zu präsentieren.

Wie bereits erwähnt, ist die *OpenForm*-Methode nur dazu geeignet, die Standardinstanz eines Formulars zu öffnen. Wenn Sie die *OpenForm*-Methode ein zweites Mal anwenden, aktiviert die Methode lediglich die Standardinstanz. Das Formular wird aber nicht ein zweites Mal geöffnet.

Zum Anlegen einer Instanz müssen Sie auf die gleiche Weise vorgehen, wie beim Erzeugen einer neuen Objektinstanz einer Klasse:

- Deklaration einer Objektvariablen mit dem Klassennamen des Formulars
- Erzeugen der neuen Instanz
- Zuweisen der neuen Instanz an die Objektvariable

Der Klassenname eines Formulars setzt sich aus dem Text *Form_* und dem Namen des Formulars zusammen. Das Formular *Kunden* der Beispieldatenbank hat beispielsweise den Klassennamen *Form_Kunden*. Um eine Instanz dieses Formulars zu öffnen, verwenden Sie die folgende Anweisung, die alle drei oben aufgeführten Schritte in einem Durchgang erledigt:

```
Dim MeinKundenformular As New Form_Kunden
```

○ — Anlegen einer Instanz





Stattdessen können Sie auch zwei einzelne Anweisungen angeben:

```
Dim MeinKundenformular As Form_Kunden  
Set MeinKundenformular = New Form_Kunden
```

Wenn Sie nun zwei Instanzen des Kundenformulars öffnen möchten, benötigen Sie dementsprechend zwei Variablen:

```
Dim MeinKundenformular1 As New Form_Kunden  
Dim MeinKundenformular2 As New Form_Kunden
```

Instanzen sichtbar
machen

Wenn Sie auf diese Weise Formularinstanzen öffnen, sind diese nicht direkt auf dem Bildschirm sichtbar. Der Grund dafür ist, dass Access die Formulare im unsichtbaren Zustand öffnet. Um die Instanzen sichtbar zu machen, weisen Sie der *Visible*-Eigenschaft den Wert *True* zu. Ein weiteres Problem besteht darin, dass alle Instanzen exakt die gleiche Bildschirmposition und die gleiche Größe haben. Wenn Sie mehrere Formularinstanzen öffnen, ist daher immer nur das oberste Formular sichtbar.

Das folgende Beispiel öffnet zwei Instanzen des Formulars *Kunden* und macht diese anschließend mithilfe der *Visible*-Eigenschaft sichtbar. Damit sich die Formulare nicht gegenseitig verdecken, werden sie über den Menüpunkt *Fenster > Alle anordnen* nebeneinander auf dem Bildschirm angeordnet. Die Ausführung dieses Menüpunkts bewerkstelligt die *RunCommand*-Methode:

```
Sub ZweiKundenformulareÖffnen()  
  
Dim MeinKundenformular1 As New Form_Kunden  
Dim MeinKundenformular2 As New Form_Kunden  
  
    MeinKundenformular1.Visible = True  
    MeinKundenformular2.Visible = True  
  
    DoCmd.RunCommand acCmdTileVertically  
  
    MsgBox "Es wurden zwei Instanzen geöffnet"  
  
End Sub
```

Wenn Sie die Prozedur – die Sie in der Beispieldatenbank finden – ausführen, öffnet Access die Formularinstanzen und ordnet diese auf dem Bildschirm an. Anschließend wird eine Meldung angezeigt. Wenn Sie diese Meldung mit *OK* bestätigen, verschwinden die Formulare wieder vom Bildschirm. Der Grund: Mit dem Ende der Prozedur ist auch die Lebensdauer der beiden Objektvariablen beendet. Access entfernt hierbei automatisch alle Objekte aus dem Speicher. Die Folge ist, dass Access die Formulare schließt.



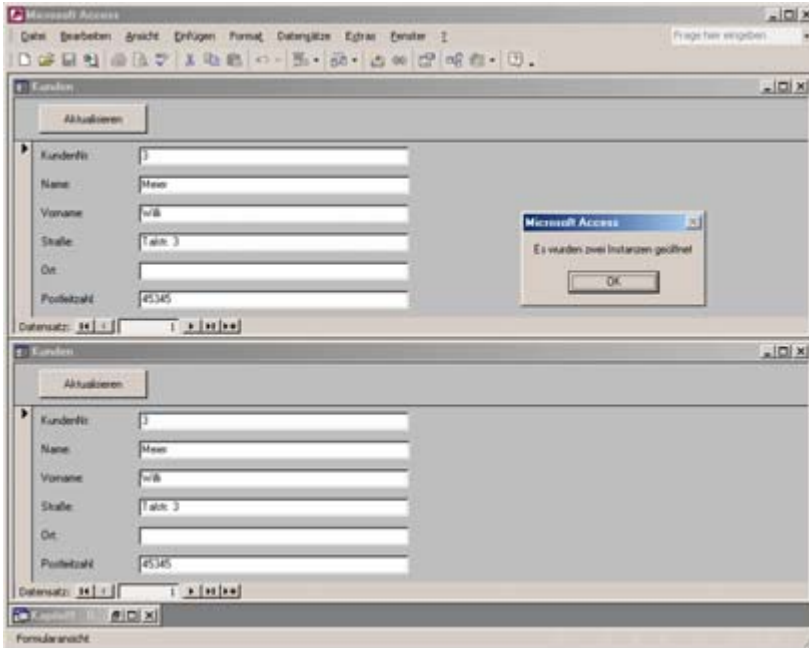


Abb. 9.2

Zwei Instanzen des Formulars *Kunden*

Dauerhaftes Öffnen von Formularinstanzen

Eine Möglichkeit zum dauerhaften Öffnen von Formularen besteht im Einsatz von globalen Variablen (also öffentlichen oder privaten Variablen) in Standardmodulen. Die Lebensdauer dieser Variablen beginnt mit dem Öffnen und endet mit dem Schließen der Datenbank.

Zum Speichern der einzelnen Objektverweise können Sie entweder Arrays (vorzugsweise dynamische Arrays) oder *Collection*-Objekte einsetzen. Letzteres ist in der Praxis unkomplizierter.

Das folgende Beispiel öffnet mehrere Instanzen des Formulars *Kunden*, zeigt eine Meldung an und blendet die einzelnen Formulare schließlich ein. Voraussetzung für das Funktionieren der Prozedur ist, dass Sie im Deklarationsbereich des Moduls folgende globale Variable deklariert haben:

```
Private Formularauflistung As New Collection
```

Es folgt die Prozedur:

```
Sub MehrereKundenformulareDauerhaftÖffnen()
```

```
Dim EineKundenformularinstanz As Form_Kunden
```

```
Dim i As Integer
```

```
Dim Anzahl As Integer
```





```
Anzahl = Val(InputBox("Wieviele Instanzen möchten _
                        Sie öffnen?"))

If Anzahl = 0 Then Exit Sub "Abbrechen"

For i = 1 To Anzahl
    Set EineKundenformularinstanz = New Form_Kunden
    Formularauflistung.Add EineKundenformularinstanz
Next i

MsgBox "Die Instanzen wurden geöffnet." & _
    "Die Formulare werden nun eingeblendet."

For Each EineKundenformularinstanz In _
    Formularauflistung
    EineKundenformularinstanz.Visible = True
Next EineKundenformularinstanz

DoCmd.RunCommand acCmdTileHorizontally

End Sub
```

Beim Start der Prozedur werden Sie aufgefordert, die Anzahl der zu öffnenden Instanzen anzugeben. Access legt schließlich die entsprechende Anzahl an Instanzen an und zeigt eine Meldung an. Um die einzelnen Formulare sichtbar zu machen, setzt die Prozedur die *Visible*-Eigenschaft auf *True* und ordnet die Formulare auf dem Bildschirm an.

Schließen der
Instanzen

Das Schließen der Instanzen bewerkstelligt folgende Prozedur:

```
Sub FormularinstanzenSchließen()

Dim i As Integer
Dim EinFormular As Form

    For i = Formularauflistung.Count To 1 Step -1

        Set EinFormular = Formularauflistung(i)
        Set EinFormular = Nothing
        Formularauflistung.Remove i

    Next i

End Sub
```

Beachten Sie, dass die *For...Next*-Schleife rückwärts zählt. Dies ist erforderlich, da beim Entfernen eines Elements aus der Auflistung alle nachfolgenden Elemente aufrücken. Wenn Sie direkt das Formular mit dem Index 1 löschen, würde das zweite Formular aufrücken und den Index 1 bekommen. Wenn Sie nun anschlie-



ßend das Objekt mit dem Index 2 löschen möchten, könnte – wenn Sie nur zwei Instanzen geöffnet hätten – unter diesem Index bereits kein Formular mehr vorhanden sein. Die Folge wäre ein Laufzeitfehler.

Wenn Sie eine geöffnete Formularinstanz schließen, sollten Sie dafür sorgen, dass diese auch aus der Auflistung entfernt wird. So stellen Sie sicher, dass hier keine Verweise auf nicht mehr vorhandene Objekte enthalten sind. Am einfachsten bewerkstelligen Sie dies, indem Sie es vom *Schließen*-Ereignis des Formulars aus erledigen. So ist sichergestellt, dass das Formular auch dann aus der Auflistung entfernt wird, wenn es vom Benutzer mithilfe der Maus – beispielsweise über die X-Schaltfläche – geschlossen wird. Das Problem ist hierbei allerdings, dass Sie ein bestimmtes Formular innerhalb der Auflistung identifizieren müssen, um es anschließend gezielt entfernen zu können. Da der Index innerhalb der benutzerdefinierten Auflistung willkürlich vergeben wird, ist er dafür nicht geeignet. Stattdessen müssen Sie jedem Element der Auflistung beim Hinzufügen einen eindeutigen *Key* zuweisen. Zusätzlich müssen Sie diesen Key auch zusammen mit dem Formular speichern, um später eine Zuordnung vornehmen zu können. Dazu eignet sich beispielsweise eine öffentliche Variable, die Sie innerhalb des Formularobjekts anlegen.

Aktualisieren der Auflistung

In der Beispieldatenbank finden Sie das Formular *Kunden*, das die folgende öffentliche Variable enthält:

```
Public Key As String
```

Das Öffnen der Formularinstanzen bewerkstelligt die folgende Prozedur:

```
Sub MehrereKundenformulareDauerhaftÖffnen2 ()

Dim EineKundenformularinstanz As Form_Kunden
Dim i As Integer
Dim Anzahl As Integer

Anzahl = Val(TextBox("Wie viele Instanzen möchten _
    Sie öffnen?"))

If Anzahl = 0 Then Exit Sub "Abbrechen"

For i = 1 To Anzahl
    Set EineKundenformularinstanz = New Form_Kunden

    EineKundenformularinstanz.Key = _
        EindeutigenKeyErmitteln(Formularauflistung)

    Formularauflistung.Add EineKundenformularinstanz, _
        Str(EineKundenformularinstanz.Key)

    EineKundenformularinstanz.Visible = True
```





```
Next i
```

```
DoCmd.RunCommand acCmdTileHorizontally
```

```
End Sub
```

Ermitteln eines
eindeutigen
Schlüssels

Innerhalb der Prozedur wird die Funktion *EindeutigenKeyErmitteln* verwendet, die einen Schlüssel erzeugt, der für die Auflistung eindeutig ist. Die Funktion gibt zu diesem Zweck eine Zeichenkette zurück, die einen eindeutigen Zahlenwert enthält. Dieser ist um eins größer, als der maximal vorhandene *Key*. Damit die Funktion möglichst universell eingesetzt werden kann, greift sie nicht direkt auf die Auflistung *Formularauflistung* zu, sondern erwartet die betreffende Auflistung als Parameter:

```
Function EindeutigenKeyErmitteln(C As Collection) As String
```

```
Dim MaxKey As Integer  
Dim EinObjekt As Object
```

```
For Each EinObjekt In C
```

```
    MaxKey = Val(EinObjekt.Key)
```

```
Next EinObjekt
```

```
EindeutigenKeyErmitteln = Str(MaxKey + 1)
```

```
End Function
```

Damit das betreffende *Form*-Objekt beim Schließen eines Formulars aus der Auflistung entfernt wird, ist das *Schließen*-Ereignis des Formulars an folgende Ereignisprozedur geknüpft:

```
Private Sub Form_Close()
```

```
    KundenformularEntfernen Key
```

```
End Sub
```

Die öffentliche Prozedur *KundenformularEntfernen* befindet sich wieder in dem Standardmodul der Datenbank. So ist sichergestellt, dass sie auf die privat deklarierte Auflistung *Formularauflistung* zugreifen kann. Die Prozedur hat den folgenden Aufbau:

```
Sub KundenformularEntfernen(Key As String)
```

```
On Error Resume Next
```

```
    Formularauflistung.Remove Key
```

```
End Sub
```



Mithilfe der *On Error Resume*-Anweisung werden alle Fehler abgefangen, die beim Entfernen des Objekts aus der Auflistung auftreten. Das Augenmerk liegt hierbei auf folgendem Fehler: Wenn Sie über das Datenbankfenster die Standardinstanz des Kundenformulars öffnen, wird diese nicht automatisch der Auflistung *Formularauflistung* hinzugefügt. Da beim Schließen dieses Formulars ebenfalls die Prozedur *KundenFormularEntfernen* ausgeführt wird, kommt es zu einem Laufzeitfehler, der allerdings dank der *On Error*-Anweisung ignoriert wird.

Wenn Sie Änderungen an der Variablen *Key* innerhalb des Kundenformulars kontrollieren möchten, können Sie diese als privat deklarieren und den Zugriff nur über zwei *Property*-Prozeduren erlauben. In diesem Zusammenhang geben Sie der privaten Variablen z.B. den Namen *m_Key* und der benutzerdefinierten Eigenschaft den Namen *Key*. Dies ist erforderlich, da Variablen und Eigenschaften eines Moduls nicht denselben Namen aufweisen dürfen.



Hinweis

Beim Öffnen mehrerer Instanzen eines Formulars werden diese automatisch der *Forms*-Auflistung hinzugefügt. Da sie hier allerdings alle unter dem gleichen Namen aufgeführt werden, gibt es keine Möglichkeit, die einzelnen Instanzen zu unterscheiden. Dies ist auch der Grund dafür, warum in dem obigen Beispiel zur Verwaltung der Instanzen eine benutzerdefinierte Auflistung zusammen mit der öffentlichen Variablen *Key* verwendet wurde.



Forms-Auflistung





Design von Datenbankobjekten



Datenbanken designen	471
Tabellen designen	473
Design von Abfragen	493
Formulare und Berichte designen	510
Zugriff auf Module	531



10



Alle Datenbankobjekte von Access, mit Ausnahme von Makros und Modulen, lassen sich auch mit VBA erzeugen. Darunter fällt das Erstellen von Tabellen mit Indizes und Beziehungen, das Definieren von Abfragen und der Entwurf von Formularen, Berichten und Modulen. Es gibt gewisse Schlüsselwörter, Eigenschaften und Methoden, mit denen neue Objekte geschaffen werden können. Alle Objekte werden dabei aus der aktiven Datenbank heraus erstellt, so dass Sie zum Beispiel Formulare während der Laufzeit verändern können.

Die hier beschriebenen Mechanismen und Vorgehensweisen basieren auf der DAO-Technologie. Da Access XP jedoch eine neu angelegte Datenbank standardmäßig mit den ADO-Objekten verbindet, müssen Sie zunächst den Verweis von ADO auf DAO umstellen. Wählen Sie hierzu im Visual-Basic-Editor den Menüeintrag *Extras > Verweise*, deaktivieren Sie in der Liste den Eintrag *Microsoft ActiveX Data Objects Library* und aktivieren Sie dafür den Eintrag *Microsoft DAO Object Library*.

Bei den Objekten kann zwischen solchen unterschieden werden, auf die über die Datenbank-Engine zugegriffen wird, und solchen, die angesprochen werden. Zu den Datenbank-Engine-Objekten gehören zum Beispiel die Datenbank-Engine selbst, Datenbanken, Benutzer, Tabellen, Abfragen usw. Auf Formulare und Berichte wird dagegen über das *Application*- oder *CurrentProject*-Objekt zugegriffen.

Das macht sich darin bemerkbar, dass der Inhalt von Tabellen oder Feldnamen jederzeit ermittelbar ist, auf Felder von Formularen aber nur über das *Application*-Objekt zugegriffen werden kann, wenn sie geöffnet sind. Die Collection *CurrentProject.AllForms* liefert dagegen auch die Formulare, die nicht geöffnet sind.

Auch die Anzahl der Objekte lässt sich bei Tabellen sofort bestimmen. Es wird die Anzahl der im Datenbankfenster vorhandenen Tabellen ermittelt. Versucht man auf die gleiche Weise die Anzahl der Formulare der Datenbank herauszubekommen, erscheint (wenn kein Formular geöffnet ist) eine Null als Ergebnis. Es werden nur geöffnete Formulare berücksichtigt. Sollen alle Formulare aufgelistet werden, also auch die geschlossenen, müssen Sie die Auflistung *AllForms* des Container-Objekts *CurrentProject* durchlaufen. Mithilfe der Eigenschaft *Name* können Sie sich dann die Namen aller Formulare anzeigen lassen.

Zum besseren Verständnis und leichteren Nachvollziehen der Beispiele ist in *Abbildung 10.1* die Objekthierarchie der DAO-Datenbank-Engine von Access abgebildet. Die Kenntnis über den Zugriff auf die jeweiligen Objekte wie Tabellen und Formulare wird in diesem Kapitel vorausgesetzt. Falls Sie damit nicht vertraut sein sollten, lesen Sie zuerst die entsprechenden *Kapitel 3 Die Programmiersprache VBA* und *Kapitel 4 Programmieren mit Objekten*.

Hinweis



Alle in diesem Kapitel vorkommenden Beispiele sind vollständig in der Datenbank *Kapiteho.mdb* enthalten. Dabei sind alle Prozeduren aus einem Abschnitt jeweils in einem Modul zusammengefasst. Die Datenbank finden Sie auf der dem Buch beiliegenden CD.



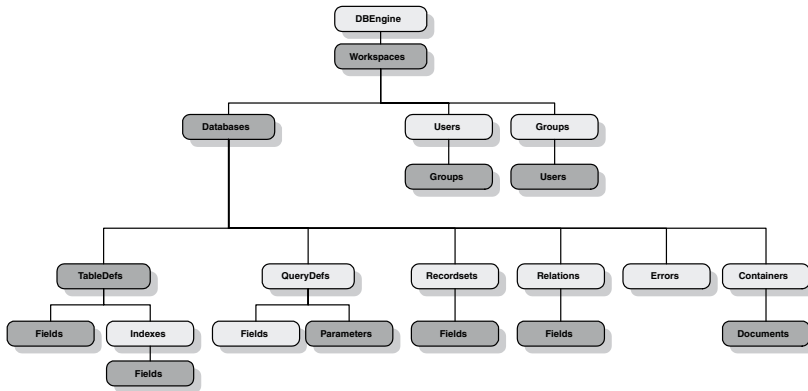


Abb. 10.1

Die Objekthierarchie von Access für den Zugriff auf die DAO-Datenbank-Engine

Datenbanken designen

Um eine neue Datenbank mit VBA zu erzeugen, stellen die *Workspace*-Objekte von Access die *CreateDatabase*-Methode zur Verfügung. Während der Ausführung einer Prozedur können dabei mehrere Datenbanken erzeugt werden und gleichzeitig geöffnet sein. Diese Datenbanken gehören dann alle zu dem aktuellen *Workspace*-Objekt. Die Syntax der *CreateDatabase*-Methode lautet:

```
CreateDatabase(<Name>, <Connect> [, <Option>])
```

Beim Argument *<Name>* handelt es sich um eine Zeichenkette, die den neuen Datenbanknamen samt Pfad angibt. Das Argument *<Connect>* bestimmt die Reihenfolge für Zeichenfolgenvergleiche von Texten und lässt sich über Konstanten festlegen. Die Konstanten finden Sie in *Tabelle 10.1*.

Konstanten für die Sortierreihenfolge

Konstante	Reihenfolge
<i>dbLangGeneral</i>	Englisch, Deutsch, Französisch, Portugiesisch, Italienisch und Spanisch (Modern)
<i>dbLangArabic</i>	Arabisch
<i>dbLangChineseSimplified</i>	Vereinfachtes Chinesisch
<i>dbLangChineseTraditional</i>	Traditionelles Chinesisch
<i>dbLangCzech</i>	Tschechisch
<i>dbLangCyrillic</i>	Russisch
<i>dbLangDutch</i>	Holländisch



Konstanten für die Sortierreihenfolge (Forts.)

Konstante	Reihenfolge
<i>dbLangGreek</i>	Griechisch
<i>dbLangHebrew</i>	Hebräisch
<i>dbLangHungarian</i>	Ungarisch
<i>dbLangIcelandic</i>	Isländisch
<i>dbLangJapanese</i>	Japanisch
<i>dbLangKorean</i>	Koreanisch
<i>dbLangNordic</i>	Skandinavische Sprachen (nur Microsoft JET-Datenbankmodul, Version 1.0)
<i>dbLangNorwdan</i>	Norwegisch und Dänisch
<i>dbLangPolish</i>	Polnisch
<i>dbLangSlovenian</i>	Slowenisch
<i>dbLangSpanish</i>	Spanisch
<i>dbLangSwedfin</i>	Schwedisch und Finnisch
<i>dbLangThai</i>	Thailändisch
<i>dbLangTurkish</i>	Türkisch

Tabelle 10.1

Für den Platzhalter <Option> kann eine der in Tabelle 10.2 aufgelisteten Konstanten verwendet werden. Die Datenbank kann damit verschlüsselt werden und es kann die Versionsnummer der JET-Datenbank-Engine, die verwendet werden soll, angegeben werden. Durch Addition von Konstanten lassen sich die Optionen auch kombinieren.

Konstanten für die verwendbaren Optionen

Konstante	Beschreibung
<i>dbEncrypt</i>	Verschlüsselt die Datenbank.
<i>dbVersion10</i>	Die Datenbank verwendet die JET-Datenbank-Engine der Version 1.0.
<i>dbVersion11</i>	Die Datenbank verwendet die JET-Datenbank-Engine der Version 1.1.



Konstanten für die verwendbaren Optionen (Forts.)

Konstante	Beschreibung
<i>dbVersion20</i>	Die Datenbank verwendet die JET-Datenbank-Engine der Version 2.0.
<i>dbVersion30</i>	Die Datenbank verwendet die JET-Datenbank-Engine der Version 3.0 (kompatibel mit der von Access 97 verwendeten Version 3.5).
<i>dbVersion40</i>	Die Datenbank verwendet die JET-Datenbank-Engine der Version 3.6 bzw. 4.0.

Tabelle 10.2

Datenbank erzeugen

Die folgenden zwei Zeilen erzeugen eine neue Datenbank. Zuerst muss eine Variable vom Datentyp *Database* definiert werden. Dieser wird dann mittels der *Set*-Anweisung der neuen Datenbank zugewiesen. Das Beispiel führt genau diese Schritte aus:

```
Dim DB As Database
Set DB = DBEngine.Workspaces(0).CreateDatabase _
    ("C:\Programme\Microsoft _
    Office\Office\Beispiele\NeueDB.MDB", _
    dbLangGeneral, dbEncrypt)
```

Die Datenbank wird automatisch der *Databases*-Objektgruppe des *Workspace*-Objekts zugefügt.

Falls Sie Access in einem anderen Pfad als *C:\Programme\Microsoft Office\Office* installiert haben, gleichen Sie den im Prozedurcode angegebenen Pfad den Gegebenheiten Ihres PCs an.



Hinweis

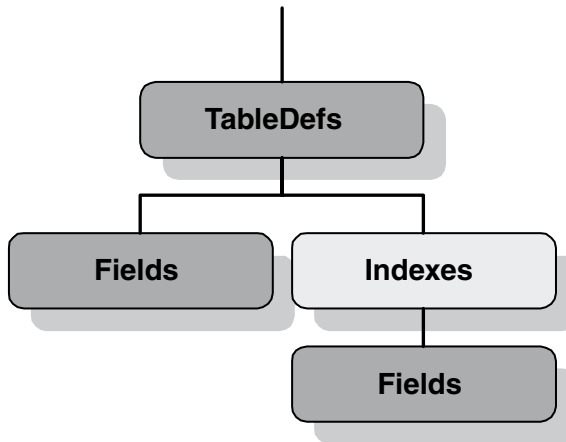
Tabellen designen

Zur Definition von Tabellen, Indizes und Beziehungen mit VBA werden die Objekte *Database*, *TableDef*, *Relation*, *Field* und *Index* verwendet. Jedes einzelne Objekt beinhaltet dazu die entsprechenden Methoden.



Abb. 10.2

Die *TableDef*-
Objekthierarchie



Tabellen erzeugen

Zum Erzeugen einer neuen Tabelle werden die *TableDef*-Objekte benutzt. Mithilfe des Schlüsselworts *New* wird ein neues, temporäres *TableDef*-Objekt angelegt. Die Tabelle kann der Datenbank aber erst hinzugefügt werden, wenn auch Felder definiert worden sind. Das Speichern einer leeren Tabelle wäre ja auch sinnlos.

Hinweis



Anders als bei Abfragen, die Sie mit der *CreateQueryDef*-Methode erstellen können, werden die mit *New* erzeugten Tabellen nicht automatisch an das *Database*-Objekt angefügt und gespeichert. Es wird lediglich eine temporäre Tabelle erzeugt, die der Datenbank mit der *Append*-Methode hinzugefügt werden muss.

Erstellen einer temporären Tabelle

Das folgende Beispiel erzeugt eine solche temporäre Tabelle. Die Tabelle erhält nur einen Namen. Andere Attribute werden nicht definiert.

```

Dim DB As Database
Dim T As TableDef

Set DB = CurrentDb
Set T = New TableDef
T.Name = "Kunden"
  
```

Um der Tabelle besondere Merkmale zuzuweisen oder die Eigenschaften anzeigen zu lassen, benutzen Sie die Eigenschaft *Attributes*. Alle Konstanten der Eigenschaft sind in *Tabelle 10.3* aufgeführt.



Attribute für TableDef-Objekte

Konstante	Wert	Beschreibung
<i>dbHiddenObject</i>	1	Die Tabelle wird nicht im Datenbankfenster angezeigt.
<i>dbAttachExclusive</i>	65536	Die Tabelle ist eine eingebundene Tabelle, die exklusiv geöffnet ist.
<i>dbAttachsSavePWD</i>	131072	Der Benutzername und das Kennwort der eingebundenen Tabelle werden zusammen mit den Informationen zur Verbindung gespeichert.
<i>dbAttachedODBC</i>	536870912	Die Tabelle ist eine eingebundene Tabelle aus einer ODBC-Datenbank, wie z. B. Microsoft SQL-Server oder ORACLE-Server.
<i>dbAttachedTable</i>	1073741824	Die Tabelle ist eine eingebundene Tabelle aus einer Datenbank, auf die nicht über ODBC zugegriffen wird, wie z. B. Microsoft Access oder Paradox.
<i>dbSystemObject</i>	-2147483646	Die Tabelle ist eine Systemtabelle.

Tabelle 10.3

Die Konstanten sind vom Datentyp *Long*. Durch einfaches Aufsummieren von verschiedenen Konstanten lassen sich beliebige Kombinationen erzeugen. Erhält eine Tabelle keine Attribute, liefert die *Attributes*-Eigenschaft den Wert 0 zurück. Die Zuweisung, um eine Tabelle im Datenbankfenster zu verstecken, lautet für das obige Beispiel wie folgt:

```
T.Attributes = dbHiddenObject
```

Es lassen sich auch Tabellen aus anderen Datenbanken einbinden. Dazu werden die Eigenschaften *Connect* und *SourceTableName* benötigt. Über die *Connect*-Eigenschaft wird die Verbindung zu der Herkunftsdatenbank hergestellt. Sie legt den Datenbanktyp der eingebundenen Tabelle und den Pfad fest, in dem sich die Datenbank der einzubindenden Tabelle befindet. Der Name der einzubindenden Tabelle wird mit der *SourceTableName*-Eigenschaft definiert.

Mit dem folgenden Beispiel wird die Tabelle *Personal* aus der Datenbank *Nord-Wind.mdb* in die aktuelle Datenbank eingebunden. Voraussetzung ist dabei, dass sich die Datenbank im Verzeichnis *C:\Programme\Microsoft Office\Office\Beispiele* befindet. Sollte dies nicht der Fall sein, geben Sie den Pfad an, in dem sich die Datenbank auf Ihrem PC befindet.

Tabellen einbinden





```
Sub TabelleEinbinden()

    Dim DB As Database
    Dim T As TableDef

    Rem Aktuelle Datenbank auswählen
    Set DB = CurrentDb

    Rem Temporäre Tabelle mit dem Namen
    Rem "tab_Eingebunden_Personal" erzeugen
    Set T = New TableDef
    T.Name = "tab_Eingebunden_Personal"

    Rem Datenbank der einzubindenden Tabelle definieren
    T.Connect = ";DATABASE=C:\Programme\Microsoft _
                Office\Office\Beispiele\Nordwind.MDB"

    Rem Namen der einzubindenden Tabelle festlegen
    T.SourceTableName = "Personal"

    Rem Tabelle einbinden
    DB.TableDefs.Append T

End Sub
```

Für jeden Datenbanktyp gibt es eine genau definierte Eingabevorschrift, sowohl für den verwendeten Typ als auch für die Angabe des Verzeichnisses. In *Tabelle 10.4* sind diese Angaben für verschiedene Datenbanken aufgelistet.

Eingabevorschriften für das Einbinden von Tabellen



Datenbanktyp	Bezeichner	Pfad
Microsoft	Access	[Datenbank];
Laufwerk:\Pfad\Dateiname.mdb	dBASE III	dBASE III;
Laufwerk:\Pfad	dBASE IV	dBASE IV;
Laufwerk:\Pfad	dBASE 5	dBASE 5;
Laufwerk:\Pfad	Paradox 3.x	Paradox 3.x;
Laufwerk:\Pfad	Paradox 4.x	Paradox 4.x;
Laufwerk:\Pfad	Paradox 5.x	Paradox 5.x;
Laufwerk:\Pfad	Btrieve	Btrieve;
Laufwerk:\Pfad\Dateiname.ddf	FoxPro 2.0	FoxPro 2.0;



Eingabevorschriften für das Einbinden von Tabellen (Forts.)

Datenbanktyp	Bezeichner	Pfad
Laufwerk:\Pfad	FoxPro 2.5	FoxPro 2.5;
Laufwerk:\Pfad	FoxPro 2.6	FoxPro 2.6;
Laufwerk:\Pfad	Excel 3.0	Excel 3.0;
Laufwerk:\Pfad\Dateiname.xls	Excel 4.0	Excel 4.0;
Laufwerk:\Pfad\Dateiname.xls	Excel 5.0 bis 97	Excel 5-7;
Laufwerk:\Pfad\Dateiname.xls	Excel 97 bis XP	Excel 8-10;
Laufwerk:\Pfad\Dateiname.xls	HTML-Import	HTML Import;
Laufwerk:\Pfad\Dateiname	HTML-Export	HTML Export;
Laufwerk:\Pfad	Text	Text;
Laufwerk:\Pfad	ODBC	ODBC;
Keine Pfadangabe		

Tabelle 10.4

Die Zuweisung über die *Connect*-Anweisung erfolgt dabei durch eine Zeichenkette, die sich wie folgt zusammensetzt:

```
"<Bezeichner>DATABASE=<Pfad>"
```

Wenn Sie zum Beispiel eine FoxPro 2.6-Datenbank mit dem Namen *Beispiel.dbf* vorliegen haben, die sich im Verzeichnis *C:\FoxPro* befindet, und aus dieser Datenbank eine Tabelle in Ihre Datenbank einbinden wollen, lautet die Anweisung:

```
T.Connect = "FoxPro 2.6;DATABASE=C:\FoxPro\Beispiel.Dbf"
```

Beim Zugriff auf eine ODBC-Datenbank ist noch die Angabe eines Benutzers, eines Kennworts und des Namens der Datenquelle möglich.

Werden außer *ODBC*; keine weiteren Angaben gemacht, kann der Benutzer über ein automatisch erscheinendes Dialogfenster die richtige ODBC-Datenquelle aus allen vorhandenen auswählen. Gleiches gilt auch für das Anmeldekennwort: Wenn beim Einbinden einer Tabelle mit der *Connect*-Methode kein Kennwort eingetragen wird, für die Tabelle aber eine Zugriffsbeschränkung besteht, wird bei jedem Zugriff auf die Tabelle der *Anmelden*-Dialog angezeigt.



Hinweis





Felder erzeugen

Beim Erzeugen eines neuen *TableDef*-Objekts wird zunächst eine temporäre Tabelle angelegt. Damit die Tabelle der Datenbank hinzugefügt werden kann, müssen die Felder der Tabelle definiert werden. Dazu dient die *Fields*-Objektgruppe, mit der über die *Set*-Anweisung und das Schlüsselwort *New* neue Datenfelder angelegt werden können. Der Name, der Datentyp und weitere Eigenschaften eines Feldes werden anschließend definiert.

Zunächst wird ein temporäres, leeres Feld erstellt. Anschließend wird mit den Attributen *Name*, *Type* und *Size* das Feld genauer bestimmt. Die Felder *Nachname* und *Geburtstag* der Tabelle *Kunden* werden beispielsweise folgendermaßen definiert:

```
Set F = New Field
  With F
    .Name = "Nachname"
    .Type = dbText
    .Size = 30
  End With

Set F = New Field
  F.Name = "Geburtstag"
  F.Type = dbDate
```

Die Eigenschaften eines Feldes lassen sich wahlweise über die *With*-Anweisung oder direkt über den Variablennamen angeben, wie beim Datenfeld *Geburtstag*. Es empfiehlt sich aber, die *With*-Anweisung zu verwenden. Einerseits muss bei langen Feldnamen der Name nicht in jeder Zeile aufgeführt werden. Andererseits müssen bei Änderung des Namens der Feldvariablen nicht alle Eigenschaftenzuweisungen aktualisiert werden. Im Folgenden wird deshalb bei mehr als einer Eigenschaft die *With*-Anweisung verwendet.

Felddatentypen



Jeder Felddatentyp wird durch eine Konstante repräsentiert. Die Konstante *dbDate* steht zum Beispiel für den Datentyp *Datum*. Beim Datentyp *Text* kann zusätzlich eine Größe (*Size*-Eigenschaft) zwischen 1 und 255 angegeben werden. Alle Datentypen, die verwendet werden können, sind mit ihren zugehörigen Konstanten in *Tabelle 10.5* aufgelistet.

Konstanten zur Definition von Datentypen



Konstante	Datentyp
<i>dbDate</i>	Datum/Uhrzeit
<i>dbTime</i>	Uhrzeit
<i>dbTimeStamp</i>	Zeit
<i>dbText</i>	Text



Konstanten zur Definition von Datentypen (Forts.)

Konstante	Datentyp
<i>dbChar</i>	Text fester Länge
<i>dbMemo</i>	Memo (oder Hyperlink)
<i>dbBoolean</i>	Ja/Nein
<i>dbInteger</i>	Zahl (Integer)
<i>dbBigInt</i>	Zahl (Long)
<i>dbLong</i>	Zahl (Long Integer) oder AutoWert
<i>dbCurrency</i>	Währung
<i>dbSingle</i>	Zahl (Single)
<i>dbDouble</i>	Zahl (Double)
<i>dbDecimal</i>	Zahl (Dezimal)
<i>dbFloat</i>	Zahl (Fließkomma)
<i>dbNumeric</i>	Zahl
<i>dbByte</i>	Zahl (Byte)
<i>dbGUID</i>	GUID-Daten
<i>dbBinary</i>	Binärdaten fester Länge
<i>dbVarBinary</i>	Binärdaten
<i>dbLongBinary</i>	OLE-Objekt

Tabelle 10.5

Wie auch bei den *TableDef*-Objekten wird bei Feldern über die *SET*-Anweisung und das Schlüsselwort *New* nur ein temporäres Objekt angelegt. Diesem können dann zusätzliche Eigenschaften zugewiesen werden. Um das Feld aber der Tabelle hinzuzufügen, muss auf die *Fields*-Auflistung die *Append*-Methode angewendet werden.

Um einer Tabelle mehrere Felder hinzuzufügen, reicht es aus, wenn nur eine Variable vom Typ *Fields* vorhanden ist. Ihr kann nämlich nach Ausführen der *Append*-Methode ein neues Feld zugewiesen werden, da die Objektvariable nur einen Zeiger auf das wirkliche *Fields*-Objekt darstellt, das im Speicher des Rechners erzeugt wurde. Das folgende Beispiel erzeugt die Tabelle *Kunden* mit einigen Datenfeldern.

Felder hinzufügen





```
Sub NeueTabelle()  
  
    Dim DB As Database  
    Dim T As TableDef  
    Dim F As Field  
  
    Rem aktuelle Datenbank auswählen  
    Set DB = CurrentDb  
  
    Rem Temporäre Tabelle mit dem Namen "Kunden" anlegen  
    Set T = New TableDef  
    T.Name = "Kunden"  
  
    Rem Feld "KundeNr" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "KundeNr"  
        .Type = dbLong  
        .Attributes = dbAutoIncrField  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Nachname" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "Nachname"  
        .Type = dbText  
        .Size = 30  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Vorname" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "Vorname"  
        .Type = dbText  
        .Size = 25  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Geburtstag" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "Geburtstag"  
        .Type = dbDate  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Bemerkung" erstellen und speichern  
    Set F = New Field
```



```

With F
    .Name = "Bemerkung"
    .Type = dbMemo
End With
T.Fields.Append F

Rem Tabelle hinzufügen
DB.TableDefs.Append T
    
```

End Sub

Wenn die neu erstellte Tabelle nicht direkt im Datenbankfenster sichtbar ist, wechseln Sie in eine andere Kategorie und wieder zurück in die Kategorie *Tabellen*. Das Datenbankfenster wird dadurch aktualisiert, wodurch auch die neue Tabelle angezeigt wird. Das Gleiche gilt auch für neu erzeugte Abfragen.



Hinweis

Das Feld *KundeNr* ist im Beispiel das AutoWert-Feld. Damit es auch als solches geführt wird, können neuen Feldern über die *Attributes*-Eigenschaft besondere Merkmale zugewiesen werden. Im Fall eines AutoWert-Feldes geschieht das durch die Konstante *dbAutoincrField*. Alle anderen Möglichkeiten zeigt *Tabelle 10.6*.

Konstanten der *Attributes*-Eigenschaft

Konstante	Wert	Beschreibung
<i>dbFixedField</i>	1	Die Feldgröße ist fest (Standardeinstellung für numerische Felder).
<i>dbVariableField</i>	2	Die Feldgröße ist variabel (nur für Textfelder).
<i>dbAutoincrField</i>	16	Für neue Datensätze wird der Wert des Feldes automatisch auf einen eindeutigen Wert vom Datentyp <i>Long Integer</i> erhöht, der nicht verändert werden kann.
<i>dbUpdatableField</i>	32	Der Wert des Feldes kann verändert werden.
<i>dbDescending</i>	1	Das Feld wird in absteigender Reihenfolge sortiert (dieses Attribut kann nur für <i>Field</i> -Objekte eines <i>Index</i> -Objekts eingestellt werden).
<i>dbSystemField</i>	8192	Das Feld ist ein Replikationsfeld eines <i>TableDef</i> -Objekts, das auf replizierbaren Datenbanken verwendet wird und kann nicht gelöscht werden.
<i>dbHyperlinkField</i>	32768	Das Feld enthält Information zu einem Hyperlink (nur für Memo-Felder).

Tabelle 10.6





Außer der *Attributes*-Eigenschaft lassen sich für Felder auch noch andere Eigenschaften festlegen. Mit *DefaultValue* kann beim Erzeugen eines Feldes direkt ein Standardwert zugewiesen werden. Die Eigenschaft *OrdinalPosition* gibt die Reihenfolge der Auflistung der Felder in der Tabelle an. So erscheinen zum Beispiel die Felder *Text1* und *Text2* in vertauschter Reihenfolge in der Tabelle, wenn sie mit den Zeilen

```
Set F = New Field
With F
    .Name = "Text1"
    .Type = dbText
    .Size = 10
    .OrdinalPosition = 2
End With
T.Fields.Append F
```

```
Set F = New Field
WITH F
    .Name = "Text2"
    .Type = dbText
    .Size = 10
    .OrdinalPosition = 1
End With
T.Fields.Append F
```

hinzugefügt wurden.

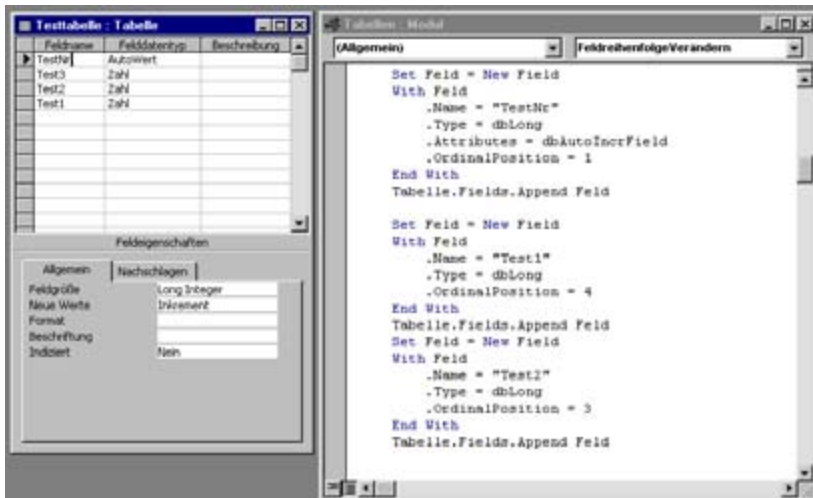


Abb. 10.3

Felder mit neuer
Reihenfolge



Mit VBA können Sie auch auf fremde, im Augenblick nicht geöffnete Datenbanken zugreifen und deren Inhalt verändern oder Objekte hinzufügen. So lässt sich zum Beispiel je nach Bedarf nachträglich noch eine Tabelle mit Feldern anlegen, das passende Formular erzeugen und die entsprechende Beziehung definieren. Im folgenden Beispiel wird der Datenbank *Nordwind.mdb* die Tabelle *NeueTabelle* mit einem Feld hinzugefügt. Auch hier ist das Verzeichnis der Datenbank gegebenenfalls anzupassen.

Fremde Datenbanken verändern

```
Sub TabelleEinfügenInExterneDatenbank()  
  
    Dim DB As Database  
    Dim T As TableDef  
    Dim F As Field  
  
    Rem Datenbank Nordwind.mdb öffnen  
    Set DB = _  
        DBEngine(0).OpenDatabase("C:\Programme\Microsoft _  
        Office\Office\Beispiele\Nordwind.mdb")  
  
    Rem Neue Tabelle erzeugen  
    Set T = New TableDef  
    T.Name = "NeueTabelle"  
  
    Rem Neues Feld erzeugen  
    Set F = New Field  
    With F  
        .Name = "NeuesFeld"  
        .Type = dbText  
        .Size = 10  
    End With  
  
    Rem Neues Feld und neue Tabelle hinzufügen  
    T.Fields.Append F  
    DB.TableDefs.Append T  
  
    Rem Datenbank Nordwind.mdb schließen  
    DB.Close  
  
End Sub
```

In der Praxis kann das Hinzufügen von Feldern zu einer Tabelle der aktuellen oder einer fremden Datenbank mit VBA für eine flexible Datenbankgestaltung genutzt werden. So kann der Anwender bei einer Adress- oder Kundenverwaltung selbst entscheiden, ob die Datenbank beispielsweise die Felder *Zweite Telefonnummer*, *Compuservefeld* und *Internetadresse* beinhalten soll oder nicht. Die Felder können über ein Dialogfenster vom Anwender aktiviert werden, so dass sie der Tabelle automatisch hinzugefügt werden und auch im Formular erscheinen. Auf das Verändern von Formularen wird im Abschnitt *Formulare und Berichte designen* noch genauer eingegangen.





Tabellen und Felder löschen

Wenn Sie Tabellen oder Felder löschen wollen, entfernen Sie diese mithilfe der *Delete*-Methode der *TableDefs*- bzw. *Fields*-Objekte. Beim Löschen ist es sinnvoll, erst einmal abzufragen, ob das zu löschende Objekt überhaupt vorhanden ist. Danach sollte noch einmal eine Bestätigung folgen, um ungewolltes Löschen zu vermeiden. Das folgende Programm entfernt eine vom Benutzer einzugebende Tabelle:

```
Sub TabelleLöschen()

    Dim DB As Database
    Dim T As TableDef
    Dim Ergebnis As Integer
    Dim tabname As String

    On Error GoTo Fehlerbehandlung

    tabname = InputBox("Bitte gebene Sie den" & _
        " Namen der zu löschenden Tabelle ein!", _
        "Tabelle löschen")

    Set DB = CurrentDb
    Set T = DB.TableDefs(tabname)

    Ergebnis = MsgBox("Tabelle " & tabname & _
        " wirklich löschen?", vbYesNo+ _
        vbQuestion, "Tabelle Löschen")

    If Ergebnis = vbYes Then
        DB.TableDefs.Delete T.Name
        MsgBox "Tabelle " & tabname & " gelöscht", _
            vbExclamation
    End If

    Exit Sub

Fehlerbehandlung:

    If Err.Number = 3265 Then
        MsgBox "Tabelle ist nicht vorhanden", _
            vbExclamation
    End If
    Exit Sub

End Sub
```

Sollen Felder der aktuellen Tabelle gelöscht werden, geschieht dies mit der Zeile:

```
T.Fields.Delete <Feldname>
```



Indizes erzeugen

Um eine Tabelle vollständig anzulegen, fehlt noch die Definition von Indizes. Der wichtigste Index in einer Tabelle ist der *Primärschlüssel*. Der Primärschlüssel ist ein eindeutiger Index, der jedem Datensatz genau einen Wert zuordnet.

Zur Definition von Indizes stellt Access die *Index*-Objekte zur Verfügung. Dabei wird wieder über die *Set*-Anweisung und das Schlüsselwort *New* ein temporärer Index erzeugt. Eigenschaften wie der Name des Index werden nachträglich definiert.

Im Beispiel der Kundentabelle ist das Feld *KundeNr* als AutoWert-Feld definiert worden. Dieses Feld soll den Primärschlüssel der Kundentabelle darstellen. Ein Primärschlüssel wird zunächst einmal wie ein normaler Index definiert. Er sollte den Namen *PrimaryKey* erhalten:

```
Dim I As Index
Set I = New Index
I.Name = "PrimaryKey"
```

Mit dieser Anweisung wird ein temporärer Index mit dem Namen *PrimaryKey* erzeugt. Um den Index als Primärschlüssel zu definieren, muss die *Primary*-Eigenschaft des *Index*-Objekts auf *True* gestellt werden:

```
I.Primary = True
```

Die *Index*-Objekte erlauben auch noch andere, zusätzliche Einstellungen. Darunter fällt die Eigenschaft *Unique*, mit der ein normales Indexfeld als eindeutiger Index deklariert werden kann. In der Entwurfsansicht der Tabelle entspricht das dem Eintrag *Ja (ohne Duplikate)* im Eingabefeld *Indiziert*.

Die Eigenschaft *Required* legt fest, ob in das Feld ein Wert eingetragen werden muss. Steht die Eigenschaft auf *True*, sind keine Nullwerte erlaubt; steht die Eigenschaft auf *False* kann das Feld ruhig leer bleiben.

Aus der Objekthierarchie wird deutlich, dass die *Index*-Objekte eine *Fields*-Auflistung beinhalten. Über diese *Fields*-Objekte wird bestimmt, welchem Datenfeld der Tabelle der temporäre Index zugewiesen werden soll. Dazu muss wieder erst ein temporäres Indexfeld angelegt werden, das dann mit der *Append*-Methode der *Fields*-Auflistung des *Index*-Objekts hinzugefügt wird:

```
Set IndexF = New Field
IndexF.Name = "KundeNr"
I.Fields.Append IndexF
```

Damit ist das Indexfeld gespeichert und dem Index zugeordnet. Zum Schluss muss noch der Index selbst an die Tabelle gebunden werden:

```
T.Indexes.Append I
```

Primärschlüssel
definieren

Unique-Eigenschaft

Required-
Eigenschaft

 Hinweis

Index hinzufügen





Dem *Fields*-Objekt einer *Index*-Objektgruppe kann noch eine Sortierreihenfolge zugewiesen werden. Der Standardwert ist die aufsteigende Sortierung. Mit der *Attributes*-Eigenschaft kann das Indexfeld über die Konstante *dbDescending* eine absteigende Sortierreihenfolge erhalten. Die Anweisung dazu lautet:

```
IndexF.Attributes = dbDescending
```

Der folgende Quelltext stellt die komplette Prozedur dar, welche die Tabelle *Kunden* anlegt, die einzelnen Felder hinzufügt, das Feld *KundeNr* als Primärschlüssel definiert und das Feld *Nachname* mit einem zusätzlichen Index versieht:

```
Sub NeueTabelle()  
  
    Dim DB As Database  
    Dim T As TableDef  
    Dim F As Field  
    Dim I As Index  
    Dim IndexF As Field  
  
    Rem Aktuelle Datenbank auswählen  
    Set DB = DBEngine.Workspaces(0).Databases(0)  
  
    Rem Temporäre Tabelle mit dem Namen "Kunden" anlegen  
    Set T = New TableDef  
    T.Name = "Kunden"  
  
    Rem Feld "KundeNr" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "KundeNr"  
        .Type = dbLong  
        .Attributes = dbAutoIncrField  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Nachname" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "Nachname"  
        .Type = dbText  
        .Size = 30  
    End With  
    T.Fields.Append F  
  
    Rem Feld "Vorname" erstellen und speichern  
    Set F = New Field  
    With F  
        .Name = "Vorname"  
        .Type = dbText  
        .Size = 25
```





```
End With
T.Fields.Append F

Rem Feld "Geburtstag" erstellen und speichern
Set F = New Field
With F
    .Name = "Geburtstag"
    .Type = dbDate
End With
T.Fields.Append F

Rem Feld "Bemerkung" erstellen und speichern
Set F = New Field
With F
    .Name = "Bemerkung"
    .Type = dbMemo
End With
T.Fields.Append F

Rem Temporäres Indexfeld anlegen
Set I = New Index
I.Name = "PrimaryKey"

Rem Index als Primärschlüssel deklarieren
I.Primary = True

Rem Temporäres Indexfeld erstellen
Set IndexF = New Field
IndexF.Name = "KundeNr"

Rem Indexfeld hinzufügen
I.Fields.Append IndexF

Rem Index hinzufügen
T.Indexes.Append I

Rem Index für das Feld "Nachname" definieren
Set I = New Index
I.Name = "Nachname"
Set IndexF = New Field
IndexF.Name = "Nachname"
I.Fields.Append IndexF
T.Indexes.Append I

Rem Tabelle hinzufügen
DB.TableDefs.Append T

End Sub
```





Clustered-
Eigenschaft

Mehrfelderindex definieren

Mit dem *Index*-Objekt lassen sich auch Mehrfelderindizes definieren. Zu diesem Zweck muss bei einem Index die *Clustered*-Eigenschaft auf *True* gesetzt werden. Anschließend werden alle Felder, die zu dem Index gehören sollen, nacheinander definiert und mit der *Append*-Methode hinzugefügt.

Das folgende Beispiel definiert für die schon vorhandene Tabelle *Personal* einen Mehrfelderindex mit dem Namen *PlzOrt*, der aus den Feldern *Plz* und *Ort* zusammengesetzt ist.

```
Sub MehrfelderindexDefinieren()

    Dim DB As Database
    Dim T As TableDef
    Dim F As Field
    Dim I As Index
    Dim IndexF As Field

    Set DB = CurrentDb
    Set T = DB.TableDefs("Personal")

    Rem Temporäres Indexfeld mit dem Namen PLZORT
    erstellen
    Set I = New Index
    I.Name = "PLZORT"

    Rem Index als Mehrfelderindex definieren
    I.Clustered = True

    Rem Felder des Index hinzufügen
    Set IndexF = New Field
    IndexF.Name = "PLZ"
    I.Fields.Append IndexF

    Set IndexF = New Field
    IndexF.Name = "Ort"
    I.Fields.Append IndexF

    T.Indexes.Append I

End Sub
```

Indizes löschen

Das Löschen von Indizes geschieht auf die gleiche Weise wie das Löschen von Feldern und Tabellen mit der *Delete*-Methode. Bei Indizes wird die *Delete*-Methode auf das *TableDef*-Objekt angewendet.



Das folgende Beispiel löscht den zuvor mit der Prozedur *MehrfelderindexDefinieren* erstellten Mehrfelderindex *PlzOrt*:

```
Sub IndexLöschen ()

    Dim DB As Database
    Dim T As TableDef
    Dim I As Index

    Set DB = CurrentDb

    Rem Tabelle auswählen, in der der Index
    Rem gelöscht werden soll
    Set T = DB.TableDefs("Personal")

    Rem Index über den Namen auswählen
    Set I = T.Indexes("PLZORT")

    Rem Index Löschen
    T.Indexes.Delete I.Name

End Sub
```

Beim Löschen eines Index spielt es keine Rolle, ob es sich um einen einfachen Index, einen Mehrfelderindex oder einen Primärschlüssel handelt. Es muss keine Differenzierung erfolgen. Der Index wird lediglich über seinen Namen identifiziert und gelöscht.



Hinweis

Beziehungen definieren

Wenn Sie mehrere Tabellen erstellt haben, werden für gewöhnlich auch Beziehungen zwischen diesen Tabellen definiert. Access bietet dazu die Standardbeziehungen 1:1 und 1:n an, die mit und ohne referentielle Integrität definiert werden können. Eine 1:n-Beziehung wird normalerweise über den Menüpunkt *Extras > Beziehungen* erstellt.

VBA stellt für die Definition von Beziehungen einen eigenen Objekttyp, den *Relation*-Objekttyp, zur Verfügung. Ein *Relation*-Objekt ist so ähnlich aufgebaut wie ein *Index*-Objekt und beinhaltet eine *Fields*-Gruppe. Wie gewohnt wird zuerst ein temporäres Objekt, hier eine Beziehung, erzeugt. Die Beziehung muss dabei einen für die aktuelle Datenbank eindeutigen Namen erhalten. Der Name ist nicht weiter von Bedeutung. Sie benötigen den Namen nur, wenn Sie die Beziehung wieder löschen möchten.

```
Dim DB As Database
Dim R As Relation

Set DB = CurrentDb
Set R = New Relation
R.Name = "Beziehung1"
```

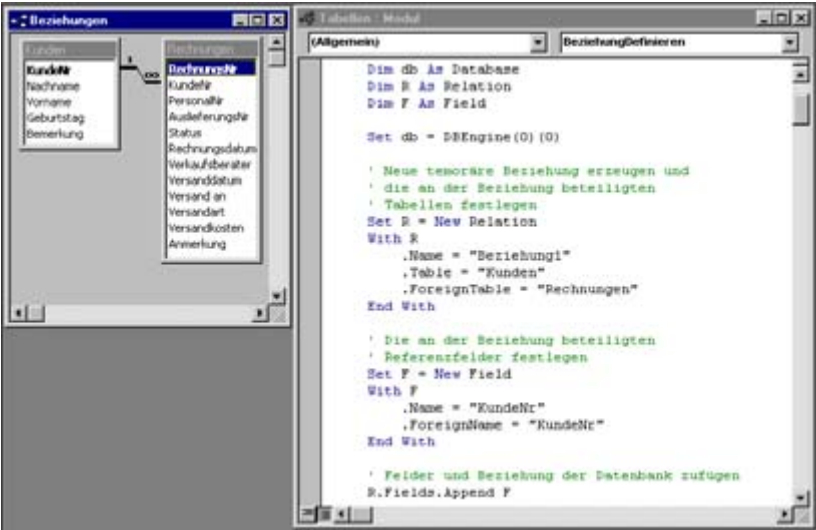


Beziehung erstellen



Abb. 10.4

Die Relations-Objekthierarchie



Verknüpfungs-
tabellen festlegen

An dieser Beziehung sind zwei Tabellen beteiligt, die über die *Table*- und *ForeignTable*-Eigenschaft festgelegt werden. Im vorliegenden Fall soll eine 1:n-Verknüpfung hergestellt werden, in der die Kundentabelle die 1-Seite und die Rechnungstabelle die n-Seite repräsentiert. Jedem Kunden sind also mehrere Rechnungen zugeordnet. Mit der *Table*-Eigenschaft wird die Tabelle der 1-Seite definiert und mit der *ForeignTable*-Eigenschaft die der n-Seite.

```
R.Table = "Kunden"
R.ForeignTable = "Rechnungen"
```

Über die *Attributes*-Eigenschaft können der Beziehung über Konstanten wieder besondere Merkmale zugewiesen bzw. Standardeinstellungen ausgeschaltet werden. Dazu gehört zum Beispiel die referentielle Integrität, die von einem neuen *Relation*-Objekt automatisch aktiviert wird. Auch die Definition einer 1:n-Beziehung ist standardmäßig eingestellt. Mit den in *Tabelle 10.7* aufgeführten Konstanten können die Werte verändert werden.

Attribute für *Relation*-Objekte

Konstante	Beschreibung
<i>dbRelationUnique</i>	Die Beziehung ist eine 1:1-Beziehung.
<i>dbRelationDontEnforce</i>	In der Beziehung wird referentielle Integrität nicht durchgesetzt.
<i>dbRelationInherited</i>	Die Beziehung besteht in einer Datenbank, die die beiden eingebundenen Tabellen enthält und die nicht die aktuelle Datenbank ist.

Attribute für Relation-Objekte (Forts.)

Konstante	Beschreibung
<i>dbRelationLeft</i>	Beziehungsart <i>Left Join</i>
<i>dbRelationRight</i>	Beziehungsart <i>Right Join</i>
<i>dbRelationUpdateCascade</i>	Aktualisierungen werden an Detailtabellen weitergegeben.
<i>dbRelationDeleteCascade</i>	Löschungen werden an Detailtabellen weitergegeben.

Tabelle 10.7

Die an der Beziehung beteiligten Tabellen stehen nun fest. Es folgt die Definition der Felder, über die die Verknüpfung hergestellt werden soll. Dazu wird zuerst auf der 1-Seite über eine *Set*-Anweisung die *Name*-Eigenschaft definiert. In vorliegendem Fall ist das das Primärschlüsselfeld *KundeNr* aus der Tabelle *Kunden*. Anschließend muss diesem Feld mit der Eigenschaft *ForeignName* das Verbindungsfeld aus der Tabelle *Rechnungen* zugewiesen werden. Es hat ebenfalls den Namen *KundeNr*.

```
Set F = New Field
  With F
    .Name = "KundeNr"
    .ForeignName = "KundeNr"
  End With
```

Zu guter Letzt müssen alle temporären Objekte wie gewohnt mit der *Append*-Methode der Datenbank hinzugefügt werden. Die folgende Prozedur erstellt die komplette Beziehung:

```
Sub BeziehungDefinieren()

  Dim DB As Database
  Dim R As Relation
  Dim F As Field

  Set DB = CurrentDb

  Rem Neue temporäre Beziehung erzeugen und
  Rem die an der Beziehung beteiligten Tabellen
festlegen
  Set R = New Relation
  With R
    .Name = "Beziehung1"
    .Table = "Kunden"
    .ForeignTable = "Rechnungen"
  End With
```

Verknüpfungsfelder festlegen



```

Rem Die an der Beziehung beteiligten Referenzfelder
Rem festlegen
Set F = New Field
With F
    .Name = "KundeNr"
    .ForeignName = "KundeNr"
End With

Rem Felder und Beziehung der Datenbank zufügen
R.Fields.Append F
DB.Relations.Append R

End Sub
    
```

Mit der obenstehenden Prozedur wird die 1:n-Beziehung zwischen den Tabellen *Kunden* und *Rechnungen* mit referentieller Integrität hergestellt. In *Abbildung 10.5* sieht man im *Beziehungen*-Fenster das Ergebnis. Es ist völlig identisch zur Vorgehensweise über den Menüpunkt *Extras > Beziehungen*.



○ Abb. 10.5
Die definierte
Beziehung im
Beziehungen-Fenster

Beziehung löschen

Das Löschen von Beziehungen geschieht auf die gleiche Weise wie das Löschen von Tabellen mit der *Delete*-Methode. Das folgende Beispiel löscht die mit der Prozedur *BeziehungDefinieren* hergestellte Beziehung, jedoch ohne Sicherheitsabfrage.



```
Sub BeziehungLöschen ()

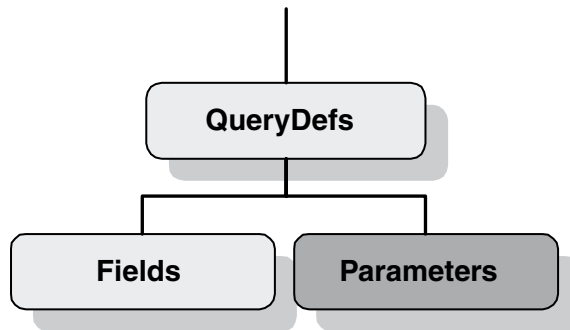
    Dim DB As Database
    Dim R As Relation

    Set DB = CurrentDb
    Set R = DB.Relations("Beziehung1")
    DB.Relations.Delete R.name

End Sub
```

Design von Abfragen

Zum Erstellen von Abfragen mit VBA stellt das *Database*-Objekt die *CreateQueryDef*-Methode zur Verfügung. Mit ihr wird eine neue Abfrage, d.h. ein *QueryDef*-Objekt erzeugt. Dieses *QueryDef*-Objekt besitzt einen Namen und mehrere Eigenschaften. Die wichtigste Eigenschaft ist dabei *SQL*, mit der der Abfragetext definiert wird.



○ Abb. 10.6

Die *QueryDef*-Objekthierarchie

Im folgenden Beispiel wird zunächst eine neue Abfrage erstellt. Anschließend wird der SQL-Text definiert.

○ Abfrage erzeugen

```
Sub NeueAbfrage ()

    Dim DB As Database
    Dim Q As QueryDef

    Set DB = CurrentDb

    Rem Erzeugen der neuen Abfrage
    Set Q = DB.CreateQueryDef("Personal nach Nachname _
                             sortiert")

End Sub
```





```
Rem Definition des SQL-Textes
Q.SQL = "Select * From Personal Order By Nachname"
```

```
End Sub
```

Wichtig ist hierbei, dass das *QueryDef*-Objekt keine Datensätze oder das Ergebnis einer Abfrage speichert. Es wird lediglich die Definition der neuen Abfrage gespeichert, die im Datenbankfenster in der Kategorie *Abfragen* erscheint. Um das Ergebnis der Abfrage einzusehen, muss ein *Recordset*-Objekt erstellt werden, durch das die Datensätze aufgelistet werden können.

Hinweis



Der Zugriff auf Datensätze einer Abfrage mittels des *Recordset*-Objekts und auch der Umgang mit SQL-Auswahlabfragen werden in diesem Kapitel vorausgesetzt. Falls Sie nicht genau wissen, wie die Befehle aufgebaut sind, können Sie die Syntax im *Kapitel 5 Datenzugriff mit VBA* nachlesen.

Bei der Definition der Abfrage kann der SQL-Text auch direkt mit angegeben werden. Für diesen Fall lautet das obige Beispiel wie folgt:

```
Sub NeueAbfrage ()

    Dim DB As Database
    Dim Q As QueryDef

    Set DB = CurrentDb
    Set Q = DB.CreateQueryDef("Personal nach Nachname _
        sortiert", _
        "Select * From Personal Order By Nachname")

End Sub
```

Hinweis



Im Gegensatz zu den *TableDef*- und *Fields*-Objekten müssen Abfragen nicht mit der *Append*-Methode der Datenbank hinzugefügt werden. Die Abfragen werden automatisch in der Datenbank gespeichert.

Mit der *CreateQueryDef*-Methode lässt sich neben der Auswahlabfrage mit dem entsprechenden SQL-Text auch jede andere Art von Abfrage erzeugen. Dazu gehören zum Beispiel Löschabfragen, Aktualisierungsabfragen oder Kreuztabellenabfragen. Die Struktur der SQL-Anweisungen dieser Abfragetypen wurde bereits im *Kapitel 5 Datenzugriff mit VBA* behandelt.

Felder einer Abfrage ermitteln

Ähnlich wie bei Tabellen lassen sich die Felder einer gerade erzeugten oder schon bestehenden Abfrage auf komfortable Weise mit dem *Fields*-Objekt ermitteln. Der Zugriff auf die Felder ist dabei identisch zu dem bei Tabellen, nur das Basisobjekt ist diesmal eine Abfrage. Mit einer kleinen Prozedur lassen sich die Felder der vorher erstellten Abfrage *Personal nach Nachname sortiert* im Direktfenster des Moduls auflisten.



```

Sub FelderEinerAbfrageAuflisten()

    Dim DB As Database
    Dim Q As QueryDef
    Dim I As Integer
    Dim Feld As Field

    Set DB = CurrentDb

    Rem Abfrage öffnen
    Set Q = DB.QueryDefs("Personal nach Nachname
sortiert")

    Rem Die Namen aller Felder der Abfrage ausgeben
    For Each Feld In Q.Fields
        Debug.Print Feld.Name
    Next Feld

End Sub

```

Die Schleife gibt die Namen der einzelnen Felder untereinander aus. Sie können aber auch alle anderen verfügbaren Eigenschaften auflisten bzw. abfragen. Durch Einfügen der Zeile

```
Debug.Print Feld.Type
```

in die Schleife können Sie zum Beispiel den Datentyp des jeweiligen Feldes mit anzeigen lassen.

Parameterabfragen handhaben

Bei der Verwendung von Parameterabfragen wird die Abfrage durch einen oder mehrere Parameter flexibel gehalten. Diesen Parametern müssen allerdings Werte zugewiesen werden, bevor die Abfrage ausgeführt werden kann. Die Zuweisung erfolgt über die Auflistung *Parameters* des *QueryDef*-Objekts. Als Beispiel dient die Abfrage *Artikel - Parameterabfrage*, die den Parameter *Preis* enthält.

```

PARAMETERS Preis Currency;
SELECT DISTINCTROW [Artikel (für
Parameterabfrage)].Artikelbezeichnung, [Artikel (für
Parameterabfrage)].Artikelpreis
FROM [Artikel (für Parameterabfrage)]
WHERE (([Artikel (für
Parameterabfrage)].Artikelpreis<[Preis]));

```

Die Abfrage gibt alle Artikel aus, deren Preis kleiner ist, als der mit dem Parameter übergebene Wert. Die Zuweisung eines Wertes erfolgt in einer Prozedur direkt über den Parameternamen.





```

Sub ParameterZuweisen ()

    Dim DB As Database
    Dim Q As QueryDef
    Dim R As Recordset

    Set DB = CurrentDb

    Rem Abfrage öffnen
    Set Q = DB.QueryDefs("Artikel - Parameterabfrage")

    Rem Dem Parameter Preis den Wert 10 zuordnen
    Q![Preis] = 10

    Rem Ergebnis der Abfrage im Direktfenster ausgeben
    Set R = Q.OpenRecordset()
    R.MoveFirst
    Do While R.eof = False
        Debug.Print "Preis: "; _
            Format(R.fields![Artikelpreis], _
                "0.00 DM"); " Artikel: "; _
            R.Fields![Artikelbezeichnung]
        R.MoveNext
    Loop
    R.Close
    DB.Close

End Sub

```

In der Beispielprozedur wird zunächst die Abfrage *Artikel - Parameterabfrage* geöffnet und anschließend dem Parameter *Preis* der Wert 10 zugewiesen. Durch die darauf folgende Schleife werden dann alle Artikel, deren Preis kleiner als DM 10,- ist, im Direktfenster ausgegeben.

Parameter ermitteln

Um herauszubekommen, ob eine bereits bestehende Abfrage Parameter benutzt und, wenn ja, welche verwendet werden, kann mit der Eigenschaft *Name* auf die Parameter einer Abfrage zugegriffen werden. Das folgende Beispiel listet die Parameter der Abfrage *Artikel - Parameterabfrage* im Direktfenster auf. Es erscheint als einziger Parameter der Preis.

```

Sub ParameterEinerAbfrageAuflisten()

    Dim DB As Database
    Dim Q As QueryDef
    Dim P As Parameter

    Set DB = CurrentDb

```



```

Rem Abfrage öffnen
Set Q = DB.QueryDefs("Artikel - Parameterabfrage")

Rem Alle Parameter der Abfrage ausgeben
For Each P In Q.Parameters
    Debug.Print P.Name
Next P

End Sub

```

Tabellen und Indizes erzeugen

Mithilfe von Abfragen können auch Tabellen mit Indizes definiert werden. Es lassen sich dabei fast alle Operationen ausführen, die auch mit den *TableDef*-Objekten gemacht werden können. *Tabelle 10.8* stellt die für die Tabellendefinition möglichen Abfragetypen und ihre Schlüsselwörter zusammen.

Die Tabellenerstellungsbefehle von Access-SQL

Anweisung	Syntax	Abfragetyp
<i>ALTER TABLE</i>	<i>ALTER TABLE <Tabellenname> {ADD DROP}</i>	Tabellenstruktur erstellen > abändern
<i>CREATE INDEX</i>	<i>CREATE [<UNIQUE-Klausel>] INDEX <Indexname> ON <Tabellenname> (<Feldliste>) [WITH {PRIMARY DISALLOW NULL IGNORE NULL}]</i>	Index erzeugen
<i>CREATE TABLE</i>	<i>CREATE TABLE <Tabellenname> (<Feldnamenliste>)</i>	Tabelle erzeugen
<i>DROP</i>	<i>DROP {TABLE <Tabellenname> INDEX <Indexname> ON <Tabellenname>}</i>	Tabelle löschen > Index löschen

Tabelle 10.8

Zum Erstellen einer neuen Tabelle mittels VBA wird ein *QueryDef*-Objekt definiert, das als *SQL*-Eigenschaft die *Create Table*-Anweisung erhält. Dabei lautet die Syntax der *Create Table*-Anweisung folgendermaßen:

```
CREATE TABLE <Tabellenname> (<Feldliste>);
```

Tabellenname bezeichnet den Namen der neu anzulegenden Tabelle. Die einzelnen Datenfelder der Tabelle werden in der *Feldliste* angegeben. *Feldliste* hat den folgenden Aufbau:

CreateTable-
Anweisung





```
<Feldname1> <Typ> [( <Größe> ) ] [ <Index1> ] [ , <Feldname2>
<Typ> [( <Größe> ) ] [ <Index2> ] . . . ] [ , <Mehrfelderindex> ] )
```

Felder definieren

Die einzelnen Felder werden durch Kommas getrennt. Wie an den Namen der Bezeichner zu erkennen ist, erhält jedes Feld einen Namen (*Feldname1*, *Feldname2*...) und einen Datentyp (*Typ*). Die möglichen Datentypen sind in *Tabelle 10.9* aufgelistet. Die optionale Größenangabe (*Größe*) bezieht sich nur auf Textfelder und kann Werte von 1 bis 255 annehmen. Wenn Sie keine Größe angeben, wird als Standardwert 255 eingetragen. Der Aufbau gestaltet sich also analog zu dem der *TableDef*-Objekte.

Datentypen in Access-SQL

Access-SQL Datentypen	Datentyp
<i>BINARY</i>	Nur für eingebundene Tabellen. Access verwendet den Datentyp nicht.
<i>BIT</i>	Ja/Nein (Boolean)
<i>BYTE</i>	Zahl (Byte)
<i>COUNTER</i>	Zahl (Long Integer) oder AutoWert
<i>CURRENCY</i>	Währung
<i>DATETIME</i>	Datum/Zeit
<i>DOUBLE</i>	Zahl (Double)
<i>LONG</i>	Zahl (Long Integer)
<i>LONGBINARY</i>	OLE-Objekt
<i>LONGTEXT</i>	String (VBA)
<i>SHORT</i>	Zahl (Short Integer)
<i>SINGLE</i>	Zahl (Single)
<i>TEXT</i>	Text
<i>VALUE</i>	Variant (VBA)

Tabelle 10.9

Indizes definieren

Anders als bei den *TableDef*-Objekten können durch die *Create Table*-Anweisung auch direkt die Indizes sowie der Primärschlüssel definiert werden. Dies geschieht durch die Platzhalter *Index1*, *Index2*..., für die eine *Constraint*-Klausel eingetragen werden kann. Am Ende der Feldliste kann ebenfalls mithilfe einer *Constraint*-Klausel ein Mehrfelderindex definiert werden.



Das folgende Beispiel definiert die neue Tabelle *Artikel* mit vier Datenfeldern, jedoch zunächst ohne Indizes:

```
Sub NeueTabellePerAbfrage ()

    Dim DB As Database
    Dim Q As QueryDef
    Dim SQL As String

    Set DB = CurrentDb

    Rem SQL-Text festlegen
    SQL = "Create Table Artikel (ArtikelNr Counter, "
    & _
    "Artikelbezeichnung Text (200), Artikelpreis _
    :Currency," & "Einheit Text (20), _
    Mehrwertsteuersatz Single);"

    Rem Abfrage erzeugen
    Set Q = DB.CreateQueryDef("qry_Artikel", SQL)

    Rem Abfrage ausführen
    Q.Execute

    Rem Abfrage löschen
    DB.QueryDefs.Delete Q.Name

End Sub
```

In der letzten Zeile wird die neu erzeugte Abfrage direkt wieder gelöscht, da sie in der Datenbank nicht weiter verwendet werden soll. Sie dient alleine zum Erzeugen der Tabelle.

Auch in allen folgenden Beispielen in diesem Abschnitt werden die erzeugten Abfragen mittels der *Delete*-Methode gelöscht, da es sich um Datendefinitions-abfragen handelt. Wird zum Beispiel eine Auswahlabfrage erzeugt, die durch die *SELECT*-Anweisung definiert wird, muss diese nicht gelöscht werden. Zum einen kann sie mehrfach aufgerufen werden, zum anderen ist so durch Öffnen der Abfrage im Datenbankfenster das Ergebnis der Abfrage direkt einsehbar.



Tipp

Achten Sie darauf, dass der Name der Abfrage, den Sie in der *CreateQueryDef*-Methode angeben, nicht derselbe ist wie der Name der Tabelle selbst. Das würde zu einem Fehler führen, da schon ein Objekt mit dem angegebenen Namen vorhanden ist. Folglich könnte die Tabelle nicht erstellt werden. Die Zeile



Hinweis

```
Set Q = DB.CreateQueryDef("Artikel", SQL)
```

ist also für das obige Beispiel nicht zulässig.





Bei der Prozedur zum Erzeugen einer neuen Tabelle wurden keine Indizes und kein Primärschlüssel definiert. Die Definition von Indizes wird über die *Constraint*-Klausel gehandhabt.

Constraint-Klausel

Mithilfe der *Constraint*-Klausel lassen sich zwei verschiedene Arten von Indizes erstellen. Die häufigste Art ist der Index bestehend aus einem Feld. Die Syntax für diesen Index lautet:

```
CONSTRAINT <Indexname> {PRIMARY KEY|UNIQUE|REFERENCES  
<Fremdtabellenname> [<Fremdfeld>]}
```

Die zweite Möglichkeit besteht in der Definition eines Index aus mehreren Feldern. Dieser Mehrfelderindex hat die Syntax:

```
CONSTRAINT <Indexname> {PRIMARY KEY (<Primärschlüssel-  
Feldliste>)|UNIQUE (<Eindeutige Schlüssel-  
Feldliste>)|FOREIGN KEY(<Referenzliste>) REFERENCES  
<Fremdtabellenname> [<Fremdfeldliste>]}
```

Die Prädikate *Primary Key*, *Unique* und *References* bestimmen die Art des Schlüssels.

Primary Key

Mit dem reservierten Wort *Primary Key* ordnen Sie einem Feld oder mehreren Feldern einen Primärschlüssel zu. Jede Tabelle darf nur einen Primärschlüssel haben, da sonst ein Fehler auftritt.

Unique

Das Prädikat *Unique* ist analog zu der *Index*-Objekteigenschaft *Unique* und entspricht somit der Feldeigenschaft *Indiziert > Ja (ohne Duplikate)* in der Entwurfsansicht einer Tabelle. Dadurch lässt sich jedes Feld oder jede Feldliste auf einen eindeutigen Schlüssel beschränken. Bei Mehrfelderindizes muss die Kombination aller zu diesem Index gehörenden Felder eindeutig sein.

Foreign Key und References

Falls ein Feld als Fremdschlüssel fungieren soll, d. h., wenn es auf ein Primärschlüsselfeld einer anderen Tabelle Bezug nimmt, zum Beispiel um eine relationale Beziehung zwischen zwei Tabellen über dieses Feld herzustellen, wenden Sie den *Foreign Key*-Ausdruck an. Ist der Primärschlüssel der Fremdtabelle aus mehreren Feldern zusammengesetzt, muss der Fremdschlüssel über einen Mehrfelderindex definiert werden, in dem die verweisenden Felder aufgeführt sind. Über *References* wird die Tabelle angegeben, auf die mit dem *Foreign Key*-Ausdruck Bezug genommen werden soll. Zusätzlich müssen die Namen der Felder, auf die verwiesen werden soll, explizit angegeben werden. Bei der Definition des Mehrfelderindex ist darauf zu achten, dass sich die Feldnamen, die Anzahl der Felder und die Reihenfolge der Felder in beiden Definitionen entsprechen.

Es folgt die abgeänderte Prozedur *NeueTabellePerAbfrage*, die das Feld *ArtikelNr* über die *Constraint*-Klausel als Primärschlüsselfeld definiert:

```
Sub NeueTabellePerAbfrage ()
```

```
Dim DB As Database  
Dim Q As QueryDef
```



```

Dim SQL As String

Set DB = CurrentDb

Rem SQL-Text festlegen
SQL = "Create Table Artikel (ArtikelNr Counter
      Constraint" & _
      "PrimaryKey Primary Key, Artikelbezeichnung
Text _
      (200), " & _
      "Artikelpreis Currency, Einheit Text (20), "
& _
      "Mehrwertsteuersatz Single);"
Rem Abfrage erzeugen
Set Q = DB.CreateQueryDef("qry_Artikel", SQL)

Rem Abfrage ausführen
Q.Execute

Rem Abfrage löschen
DB.QueryDefs.Delete Q.Name

End Sub

```

Das Ergebnis lässt sich am besten in der Entwurfsansicht der neuen Tabelle betrachten. Die folgende Abbildung zeigt die einzelnen Felder und ihre Datentypen, wie sie durch die Abfrage erstellt wurden.



Abb. 10.7

Entwurfsansicht der durch die Abfrage erzeugten Tabelle

Wird ein Feld als Primärschlüssel definiert, brauchen Sie das Prädikat *Unique* nicht zu verwenden, da Primärschlüssel immer automatisch eindeutig definiert sind.



Hinweis





Tabellen verändern

Alter Table-Anweisung

Wenn Sie keine neue Tabelle erstellen, sondern nur Felder zu einer schon bestehenden hinzufügen oder löschen wollen, verwenden Sie die *Alter Table*-Anweisung. Pro Anweisung lässt sich jeweils ein Objekt hinzufügen oder löschen. Die Syntax der Anweisung lautet:

```
ALTER TABLE <Tabellenname> {ADD {[COLUMN] <Feldname>
<Typ> [( <Größe>)] [CONSTRAINT
<Index>]|CONSTRAINT <Mehrfelderindex>} | DROP {[COLUMN]
<Feldname>|CONSTRAINT <Indexname>}}
```

Add

Mit dem Prädikat *Add* fügen Sie einer Tabelle ein Feld oder einen Index hinzu. Die Bezeichner, die auf *Add* folgen, entsprechen exakt denen bei der *Create Table*-Anweisung. Das optionale Feld *Größe* bezieht sich wieder nur auf den Datentyp *Text*.

Drop

Mit *Drop* können Sie ein Feld oder einen Index löschen. Sie müssen lediglich den Namen des Feldes bzw. den Namen des Index angeben.

Durch die folgende Prozedur werden der Tabelle *Kunden* die Felder *Straße*, *PLZ* und *Ort* hinzugefügt. Da mit jeder *Alter Table*-Anweisung jeweils nur ein Feld hinzugefügt werden kann, werden die Anweisungen in einem Array gespeichert.

```
Sub FelderHinzufügen ()

    Dim DB As Database
    Dim Q As QueryDef
    Dim SQL() As String
    Dim i As Integer

    ReDim SQL(2)

    Set DB = CurrentDbCurrentDb

    Rem SQL-Texte für die drei Felder festlegen
    SQL(0) = "Alter Table Kunden Add Column Straße _
        Text (60);"
    SQL(1) = "Alter Table Kunden Add Column PLZ Long;"
    SQL(2) = "Alter Table Kunden Add Column Ort Text
(50);"

    Rem SQL-Abfrage erzeugen, ausführen und löschen
    For i = 0 To 2
        Set Q = DB.CreateQueryDef("qry_Kunden", SQL(i))
        Q.Execute
        DB.QueryDefs.Delete Q.Name
    Next i

End Sub
```



Verwendet man in der *For...Next*-Schleife die *SQL*-Eigenschaft des *QueryDef*-Objekts, braucht die Abfrage nur einmal erzeugt und gelöscht zu werden. Bei der bestehenden Abfrage wird nur noch der *SQL*-Text verändert. Anstelle der *For...Next*-Schleife würde folgender Programmcode in der Prozedur stehen:

```
Rem SQL-Abfrage erzeugen und ausführen
Set Q = DB.CreateQueryDef("qry_Kunden", SQL(0))
Q.Execute
For i = 1 To 2
    Q.SQL = SQL(i)
    Q.Execute
Next i
```

```
Rem Abfrage löschen
DB.QueryDefs.Delete Q.Name
```

Diese Variante finden Sie in der Prozedur *FelderHinzufügen2* des Moduls *Abfragen*.

Die im Beispiel verwendete *Execute*-Methode des *QueryDef*-Objekts führt eine bestehende Abfrage direkt aus. Ein weiteres Beispiel dazu finden Sie in dem noch folgenden Abschnitt *Abfragen ausführen*.



Hinweis

Wie man aus dem Ergebnis der Abfrage ersehen kann, lässt sich mit der *Alter Table*-Anweisung nicht die Position festlegen, an der die Felder eingefügt werden sollen. Bei der *CreateField*-Methode konnte dies durch die Eigenschaft *OrdinalPosition* geregelt werden.



Abb. 10.8

Ergebnis der Abfrage
FelderHinzufügen





Indizes erzeugen

○ *Create Index-Anweisung*

Die *Create Index*-Anweisung bietet Ihnen die Möglichkeit, einer schon bestehenden Tabelle nachträglich einen einfachen Index oder einen Mehrfelderindex zuzuweisen. Die Syntax der SQL-Anweisung ist die folgende:

```
CREATE [UNIQUE] INDEX <Indexname> ON <Tabellenname>
(<Feldliste>) [WITH {PRIMARY|DISALLOW NULL|IGNORE NULL}]
```

○ *Unique*

Mit dem optionalen Prädikat *Unique* erstellen Sie einen eindeutigen Index. Der Bezeichner *Indexname* definiert den Namen des neu zu erstellenden Index, und mit *Tabellenname* geben Sie die Tabelle an, in der der Index erstellt werden soll. *Feldliste* beinhaltet bei einem einfachen Index nur einen Feldnamen, bei einem Mehrfelderindex viele Felder. Die einzelnen Felder werden dabei durch Kommas getrennt.

○ *With-Klausel*

○ *Primary*

Die *With*-Klausel ermöglicht es Ihnen, dem Indexfeld zusätzliche Merkmale zuzuordnen. Sie haben dazu drei Schlüsselwörter zur Auswahl. Mit dem reservierten Wort *Primary* können Sie das Indexfeld als Primärschlüsselfeld definieren. In diesem Fall brauchen Sie die Option *Unique* nicht mehr mit anzugeben, da Primärschlüssel automatisch eindeutig sind.

Hinweis



Besitzt eine Tabelle schon einen Primärschlüssel, dann führt das erneute Erstellen eines Primärschlüssels zu einer Fehlermeldung.

○ *Disallow Null und Ignore Null*

Mit dem *Disallow Null*-Prädikat werden in den Indexfeldern bei der Anlage eines neuen Datensatzes keine *Null*-Werte zugelassen. Mit *Ignore Null* werden Felder, die *Null*-Werte beinhalten, erst gar nicht in den Index eingefügt. Dadurch wird Platz gespart und der Index kann schneller bearbeitet werden.

○ *Mehrfelderindex erzeugen*

Das folgende Beispiel erzeugt den Mehrfelderindex *PlzOrt*, der sich aus den Feldern *Plz* und *Ort* der Tabelle *Kunden* zusammensetzt. Durch *Ignore Null* werden keine Datensätze hinzugefügt, die in einem der beiden Felder *Null*-Werte enthalten.

```
Sub MehrfelderIndex ()

    Dim DB As Database
    Dim Q As QueryDef
    Dim SQL As String

    Set DB = CurrentDb

    Rem SQL-Text definieren
    SQL = "Create Index PlzOrt On Kunden" & _
        " (Plz, Ort) With Ignore Null;"

    Rem Abfrage erzeugen
    Set Q = DB.CreateQueryDef ("qry_MehrfelderIndex",
    SQL)

    Rem Abfrage ausführen
    Q.Execute
```



```
Rem Abfrage löschen
DB.QueryDefs.Delete Q.Name
```

End Sub

Die folgende Abbildung zeigt im *Indizes*-Fenster das Ergebnis der Abfrage. Es sind der Indexname und die zugehörigen Felder aufgeführt. Auch das Feld *Nullwerte ignorieren* ist mit *Ja* richtig gesetzt worden.

Abb. 10.9

Ergebnis der Abfrage
Mehrfelderindex



Tabellen und Indizes löschen

Wenn Sie eine Tabelle oder einen Index mit einer Abfrage löschen wollen, verwenden Sie die *DROP*-Anweisung. Für das Löschen einer Tabelle ist die *DROP TABLE*-Anweisung zuständig. Der SQL-Text der Anweisung lautet:

```
DROP TABLE <Tabellenname>
```

Sie hat dieselbe Auswirkung wie die *Delete*-Methode der *TableDefs*-Auflistung. Die entsprechende Access-Basic-Funktion würde also so aussehen:

```
DB.TableDefs.Delete <Tabellenname>
```

Im Gegensatz zu der *Delete*-Methode der *TableDefs*-Auflistung hat die SQL-Anweisung *Delete* eine andere Bedeutung. Mit ihr wird nicht die Tabelle, d. h. die Struktur der Tabelle gelöscht, sondern die Datensätze der Tabelle. Die SQL-*Delete*-Anweisung realisiert also eine Löschabfrage.



Hinweis

Das folgende Beispiel entspricht der Prozedur *TabelleLöschen* aus dem Abschnitt *Tabellen designen*. Sie löscht ebenfalls eine vom Benutzer eingegebene Tabelle, allerdings mithilfe einer Abfrage.





```

Sub TabelleLöschenPerAbfrage()

    Dim DB As Database
    Dim Q As QueryDef
    Dim SQL As String
    Dim Ergebnis As Integer
    Dim Tabname As String

    On Error GoTo Fehlerbehandlung

    Set DB = CurrentDb

    Tabname = InputBox("Bitte gebene Sie den Namen  
der" & _  
        " zu  
löschenden Tabelle ein!", "Tabelle löschen")

    SQL = "Drop Table " & Tabname
    Ergebnis = MsgBox("Tabelle " & Tabname & _  
        " wirklich löschen?", vbYesNo + vbQuestion, _  
        "Tabelle löschen")

    If Ergebnis = vbYes Then
        Set Q = DB.CreateQueryDef("qry_  
TabelleLöschen", _  
            SQL)
        Q.Execute
        MsgBox "Tabelle " & Tabname & " gelöscht", _  
            vbExclamation
    End If
    DB.QueryDefs.Delete "qry_TabelleLöschen"

    Exit Sub

Fehlerbehandlung:

    If Err.Number = 3265 Then
        MsgBox "Tabelle ist nicht vorhanden",  
vbExclamation
    End If
    Exit Sub

End Sub

```

```
DROP INDEX <Indexname> ON <Tabellenname>
```



Die vorangehenden Abschnitte haben gezeigt, dass sich Tabellen, Felder und Indizes durch Abfragen ebenso gut erstellen, verändern und löschen lassen wie mit den *Database*-Objekten. Welche Form letztendlich zum Erzeugen von Tabellen gewählt wird, hängt vom vorliegenden Problem und von den Vorlieben des Entwicklers ab. Ein eingefleischter Access-Basic-Programmierer wird vermutlich nicht die SQL-Anweisung *Create Table* benutzen, um eine Tabelle zu erzeugen, sondern eher die *Set*-Anweisung und das Schlüsselwort *New*, zumal sich mit SQL auch keine Beziehungen definieren lassen. Für einen Anfänger dagegen ist es sicherlich leichter, die entsprechende Abfrage im QBE-Entwurfsbereich zu entwerfen und dann den SQL-Text in die Prozedur zu kopieren.

Ermitteln eines Abfragetyps

Über die *Type*-Eigenschaft eines *QueryDef*-Objekts ist es möglich, den Typ einer bestehenden Abfrage nachträglich zu ermitteln. In *Tabelle 10.10* sind dazu die Konstanten und deren Werte für alle unter Access möglichen Abfragetypen aufgelistet. Die Abfragen zum Erzeugen und Löschen von Tabellen, Indizes und Feldern zählen zu den Datendefinitionsabfragen, denen die Konstante *dbQDDL* zugeordnet ist.

Konstanten der *Type*-Eigenschaft von *QueryDef*-Objekten

Konstante	Wert	Abfragetyp
<i>dbQSelect</i>	0	Auswahlabfrage
<i>dbQCrosstab</i>	16	Kreuztabellenabfrage
<i>dbQDelete</i>	32	Löschabfrage
<i>dbQUpdate</i>	48	Aktualisierungsabfrage
<i>dbQAppend</i>	64	Anfügeabfrage
<i>dbQMakeTable</i>	80	Tabellenerstellungsabfrage
<i>dbQDDL</i>	96	Datendefinitionsabfrage
<i>dbQSQLPassThrough</i>	112	Pass-Through-Abfrage
<i>dbQSetOperation</i>	128	Union-Abfrage
<i>dbQSPTBulk</i>	144	Gibt zusammen mit <i>dbQSqlPassThrough</i> eine Aktualisierungsabfrage an.
<i>dbQCompound</i>	160	Verbundabfrage
<i>dbQProcedure</i>	224	Prozedur
<i>dbQAction</i>	240	Aktionsabfrage

Tabelle 10.10





Die folgende Prozedur macht die Verwendung der *Type*-Eigenschaft deutlich. Sie können über einen Dialog den Namen einer Abfrage eingeben und erhalten als Ergebnis den Wert der Konstanten des Abfragetyps.

```
Sub AbfragetypErmitteln ()

Dim DB As Database
Dim Q As QueryDef
Dim Ergebnis As String

On Error GoTo fehler

Set DB = CurrentDb

Ergebnis = InputBox("Bitte geben Sie den Namen " & _
    "einer Abfrage ein.", "Abfragetyp ermitteln")
If Ergebnis <> "" Then
Set Q = DB.QueryDefs(Ergebnis)
MsgBox Q.Type
End If

fehler:
If Err = 3011 Then
MsgBox "Abfrage nicht vorhanden!", vbExclamation
End If
Exit Sub

End Sub
```

Abfragen ausführen

In allen Beispielen dieses Abschnitts, bei denen es sich um Datendefinitionsabfragen handelte, wurden zuerst die Abfragen mit der *CreateQueryDef*-Methode erzeugt und dann mit der *Delete*-Methode gelöscht. Dies sollte die Arbeit mit dem *QueryDef*-Objekt verdeutlichen.

Die Abfragen lassen sich aber auch auf eine einfachere Art und Weise ausführen. Dazu muss weder eine neue Abfrage erzeugt noch anschließend wieder gelöscht werden. Hierzu bietet sich zum einen die *RunSQL*-Methode des *DoCmd*-Objekts an, zum anderen die *Execute*-Methode eines *QueryDef*-Objekts.

RunSql-Anweisung

Die saubere Methode, ohne auf irgendwelche Datenbankobjekte zugreifen zu müssen, ist der Befehl *AusführenSQL*, der auch als Makroaktion zur Verfügung steht. In VBA wird die englische Version (*RunSQL*) in Verbindung mit der *DoCmd*-Anweisung verwendet.

Die folgende Prozedur könnte auch nur aus einer Zeile bestehen, wenn der SQL-Text direkt hinter die *DoCmd*-Anweisung geschrieben würde. Sie ist somit die kompakteste Form, um eine Abfrage auszuführen.



```

Sub NeueTabellePerRunSQL ()

    Dim SQL As String

    Rem SQL-Text festlegen
    SQL = "Create Table Artikel (ArtikelNr Counter" & _
        " Constraint PrimaryKey Primary Key, " & _
        "Artikelbezeichnung text (200), " & _
        "Artikelpreis Currency, Einheit Text (20), " & _
        "Mehrwertsteuersatz single);"
    Rem SQL-Anweisung ausführen
    DoCmd.RunSQL SQL

End Sub

```

Die zweite Möglichkeit besteht darin, die *Execute*-Methode des *Database*-Objekts zu benutzen. Auch hier könnte die Abfrage wieder in einer Zeile geschrieben werden. Es muss aber zusätzlich noch eine Objektvariable vom Typ *Database* definiert und die Datenbank ausgewählt werden. Damit bietet sich aber die Möglichkeit, die Abfrage auch in anderen Datenbanken ausführen zu lassen, indem Sie über die *Set*-Anweisung der Variablen *DB* nicht die aktuelle Datenbank zuweisen, sondern mit der *OpenDatabase*-Methode eine externe Datenbank öffnen.

 *Execute-Methode*

```

Sub NeueTabellePerDBExecute ()

    Dim DB As Database
    Dim SQL As String

    Set DB = CurrentDb

    Rem SQL-Text festlegen
    SQL = "Create Table Artikel (ArtikelNr Counter" & _
        " Constraint PrimaryKey Primary Key, " & _
        "Artikelbezeichnung Text (200), Artikelpreis"
    & _
        " Currency, Einheit Text (20), " & _
        "Mehrwertsteuersatz single);"
    Rem SQL-Anweisung ausführen
    DB.Execute SQL

End Sub

```





Formulare und Berichte designen

Formulare und Berichte lassen sich wie Tabellen und Abfragen mit VBA erzeugen und auch verändern. Der Unterschied besteht allerdings darin, dass Formulare und Berichte geöffnet sein müssen, um Manipulationen an ihnen vornehmen zu können, und das jeweils in der Entwurfsansicht. Einige Eigenschaften lassen sich auch in der Formularansicht ändern.

Formulare erzeugen

CreateForm-Funktion

Mit der Funktion *CreateForm* lässt sich in der aktuellen Datenbank ein neues Formular in der Entwurfsansicht erzeugen. Es erscheint als minimiertes Fenster in der linken unteren Ecke des Bildschirms. Sie kennen dies von Access, wenn Sie zum Beispiel mit den Assistenten ein Autoformular auf Basis einer Tabelle erstellen lassen. Auch in diesem Fall wird zunächst das Formular links unten als minimiertes Fenster dargestellt. Danach fügt Access die einzelnen Felder der Tabelle im Formular ein und öffnet es in der Formularansicht.

Mit den folgenden Programmzeilen definieren Sie zunächst eine Variable vom Typ *Form*. Anschließend wird ein neues Formular erzeugt. Das Formular erhält dabei automatisch den Namen *Formular1*, wenn noch kein Formular mit diesem Namen vorhanden ist.

```
Dim F As Form
Set F = CreateForm()
```

Formularvorlage

Bei dem Funktionsaufruf können zwei Parameter übergeben werden. Mit dem ersten Parameter können Sie eine andere Datenbank angeben, in der das Formular erzeugt werden soll. Mit dem zweiten Parameter kann eine Formularvorlage ausgewählt werden. Mit Formularvorlagen lassen sich die Standardeigenschaften für das Formular festlegen bzw. aus anderen Formularen übernehmen. Wenn wie in dem Beispiel nichts eingetragen wird, bezieht sich die Funktion auf die aktuelle Datenbank. Als Formularvorlage wird die gewählt, die im Dialog *Optionen* in der Kategorie *Formulare > Berichte* als Formularvorlage eingetragen ist. Es kann aber auch jedes beliebige Formular als Vorlage benutzt werden. Dazu brauchen Sie nur den Namen des Formulars anzugeben. Mit der folgenden Zeile erhält das neue Formular die Standardeigenschaften des Formulars *Adressen*.

```
Set F = CreateForm(, "Adressen")
```

Es lassen sich auch alle anderen Formulareigenschaften mit VBA verändern. Dem Formular kann beispielsweise mit der *Caption*-Eigenschaft ein Titel gegeben werden oder als Rahmenart kann *Dünn* eingestellt werden. Die Prozedur *NeuesFormular* definiert die Rahmenart, die Beschriftung und die Standardansicht des Formulars neu. Alle anderen Eigenschaften bleiben unverändert.



```

Sub NeuesFormular ()

    Dim F As Form

    Rem neues Formular erzeugen
    Set F = CreateForm()

    Rem Beschriftung definieren
    F.Caption = "TestFormular"

    Rem Rahmenart auf Dünn einstellen
    F.BorderStyle = 1

    Rem Standardansicht auf Einzelnes Formular
    einstellen
    F.DefaultView = 0

End Sub

```

Anders als bei Tabellen oder Abfragen müssen Formulare nicht mit einer *Append*-Methode der Datenbank hinzugefügt werden. Das Hinzufügen zu der Datenbank geschieht über das Speichern des Formulars. Es müssen auch keine Felder erstellt werden, so dass auch leere Formulare erzeugt werden können. Wenn ein Formular Felder erhält, werden diese ebenfalls automatisch beim Speichern hinzugefügt.



Hinweis

Bei jedem Formular können verschiedene Formularbereiche aktiviert werden, für die Sie die Eigenschaften getrennt einstellen können. Auch Steuerelemente lassen sich in den verschiedenen Bereichen platzieren. Der Zugriff erfolgt über das Schlüsselwort *Section*, mit dem eine Bereichskennnummer übergeben wird. *Tabelle 10.11* listet die Bereichskennnummern für den Zugriff auf jeden Bereich auf.



Formularbereiche

Bereichskennnummern für Formulare

Bereich	Wert	Konstante
Detailbereich	0	<i>acDetail</i>
Formularkopf	1	<i>acHeader</i>
Formularfuß	2	<i>acFooter</i>
Seitenkopf	3	<i>acPageHeader</i>
Seitenfuß	4	<i>acPageFooter</i>

Tabelle 10.11



Hinweis



Wenn Sie auf einen bestimmten Bereich zugreifen wollen, muss er zuvor aktiviert, d.h. im Formular sichtbar sein. Ist dies nicht der Fall, tritt ein Laufzeitfehler auf. Die Bereiche können über die Menübefehle *Ansicht > Seitenkopf/-fuß* bzw. *Ansicht > Formularkopf/-fuß* eingeblendet werden.

Das folgende Beispiel soll für die Bereiche Formularkopf und -fuß und für den Detailbereich die Eigenschaft *Spezialeffekt* auf *Erhöht* setzen und die Hintergrundfarbe auf *Rot*.

```
Sub BereichseigenschaftenÄndern()  
  
    Dim F As Form  
    Dim i As Integer  
    Set F = CreateForm()  
  
    Rem Formularkopf und -fuß einblenden  
    DoCmd.DoMenuItem 3, 2, 11, Version:=acMenuVer70  
  
    Rem Eigenschaften ändern  
    For i = 0 To 2  
        F.Section(i).SpecialEffect = 1  
        F.Section(i).BackColor = 255      "Farbe rot  
    Next i  
  
End Sub
```

Steuerelemente erstellen

CreateControl-Funktion



Mit VBA lassen sich auch alle Steuerelemente in ein in der Entwurfsansicht geöffnetes Formular einfügen. Zu diesem Zweck verwenden Sie die *CreateControl*-Funktion. Die Syntax der Funktion lautet:

```
CreateControl(<Formularname>, <Steuerelementtyp>  
[<Bereich>, <Hauptobjekt> , <Feldname> , <Links>,  
<Oben>, <Breite>, <Höhe>])
```

Der Bezeichner *Formularname* bestimmt den Namen des Formulars, in welches das Steuerelement eingefügt werden soll. Das Formular muss in der Entwurfsansicht geöffnet sein. Über den *Steuerelementtyp* kann mit der entsprechenden Typnummer bzw. Konstanten die Art des Steuerelements ausgewählt werden. Die verschiedenen Steuerelementtypen mit ihren entsprechenden Konstanten sind in *Tabelle 10.12* aufgelistet.



Steuerelementtypen

Konstante	Wert	Steuerelement
<i>acLabel</i>	100	Bezeichnungsfeld
<i>acRectangle</i>	101	Rechteck
<i>acLine</i>	102	Linie
<i>acImage</i>	103	Bild
<i>acCommandButton</i>	104	Befehlsschaltfläche
<i>acOptionButton</i>	105	Optionsfeld
<i>acCheckBox</i>	106	Kontrollkästchen
<i>acOptionGroup</i>	107	Optionsgruppe
<i>acBoundObjectFrame</i>	108	Gebundenes Objektfeld
<i>acTextBox</i>	109	Textfeld
<i>acListBox</i>	110	Listenfeld
<i>acComboBox</i>	111	Kombinationsfeld
<i>acSubform</i>	112	Unterformular
<i>acObjectFrame</i>	114	Objektfeld
<i>acPageBreak</i>	118	Seitenumbruch
<i>acCustomControl</i>	119	Zusatzsteuerelement
<i>acToggleButton</i>	122	Umschaltfläche
<i>acTabCtl</i>	123	Register
<i>acPage</i>	124	Seite

Tabelle 10.12

Das Argument *Bereich* kennzeichnet den Formularbereich, in den das Steuerelement platziert werden soll. Tragen Sie hier die entsprechende Konstante oder Bereichskennnummer ein. Es werden dieselben Nummern und Konstanten verwendet, die in *Tabelle 10.11* für die *Section*-Methode aufgeführt sind. Der Bezeichner *Feldname* entspricht der Eigenschaft *Name* eines Steuerelements. Mit *Left*, *Top*, *Width* und *Height* kann die Position und die Größe des Elements definiert werden.

Die Angaben für Position und Größe des Steuerelements werden in der Microsoft Access-eigenen Maßeinheit Twips angegeben. Ein Twip entspricht dabei 1/20 Punkt bzw. 1/1.440 eines Zolls. Ein Zentimeter beinhaltet somit 567 Twips.



Hinweis



Der folgende Quelltext erzeugt ein neues Formular mit einer Befehlsschaltfläche.

```
Sub NeuesFormularMitSchaltfläche()

    Dim f As Form
    Dim C As Control

    Set f = CreateForm()

    Rem Befehlsschaltfläche erzeugen
    Set C = CreateControl(f.Name, acCommandButton, 0,
" ", _
        "btn_Drucken", 500, 500, 1500, 500)
    Rem Beschriftung der Schaltfläche ändern
    C.Caption = "&Drucken"
    Rem Fenster mit neuem Formular wiederherstellen
    DoCmd.Restore
End Sub
```

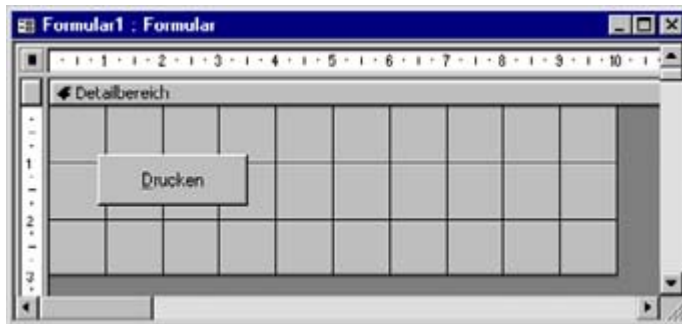
Die Parameter können auch über die Eigenschaften des Steuerelements definiert werden. Die folgenden Zeilen bewirken das Gleiche wie die *CreateControl*-Anweisung im vorangehenden Beispiel:

```
Set C = CreateControl(f.Name, 104, 0)
With C
    .Caption = "&Drucken"
    .Name = "btn_Drucken"
    .Left = 500
    .Top = 500
    .Width = 1500
    .Height = 500
End With
```

Abbildung 10.10 zeigt das neue Formular mit der definierten Befehlsschaltfläche in der Entwurfsansicht:

Abb. 10.10

Neues Formular mit einer Befehlsschaltfläche



In der *CreateControl*-Anweisung kann noch optional eine Hauptgruppe angegeben werden. Damit lassen sich zum Beispiel zusätzliche Optionsfelder einer Optionsgruppe oder Bezeichnungsfelder zuordnen. Das kann auch während der Laufzeit durch den Benutzer geschehen. Ein Beispiel dazu wird im folgenden Abschnitt erläutert.

Optionsgruppen manipulieren

Stellen Sie sich vor, Sie haben in Ihrer Datenbankanwendung ein Formular mit einer oder mehreren Optionsgruppen. Zum Beispiel könnten Sie in einer Videodatenbank zu jeder Kassette die Kassettenlänge und den Hersteller über Umschaltflächen zuordnen. Das folgende Formular realisiert die Zuordnung.

Abb. 10.11

Formular *Video* zur Auswahl von Hersteller und Länge über Umschaltflächen

Nun ist es sehr wahrscheinlich, dass Sie der Videodatenbank während der Laufzeit Hersteller oder Kassettenlängen hinzufügen wollen. Dazu müssen Sie dem Formular aber neue Umschaltflächen hinzufügen und diesen die richtigen Werte zuweisen.

Damit Sie das Beispiel besser nachvollziehen können, sind die Namen und Werte der schon vorhandenen Steuerelemente in *Tabelle 10.13* zusammengestellt.

Elemente des Formulars *Video*

Element	Name	Optionswert
Formularname	<i>Video</i>	-
Optionsgruppe <i>Hersteller</i>	<i>ogr_Hersteller</i>	-
Optionsgruppe <i>Länge</i>	<i>ogr_Länge</i>	-



Elemente des Formulars *Video* (Forts.)

Element	Name	Optionswert
Umschaltfläche <i>Agfa</i>	<i>tgl_HAgfa</i>	1
Umschaltfläche <i>Scotch</i>	<i>tgl_HScotch</i>	2
Umschaltfläche <i>Maxell</i>	<i>tgl_HMaxell</i>	3
Umschaltfläche <i>BASF</i>	<i>tgl_HBASF</i>	4
Umschaltfläche <i>go</i>	<i>tgl_Lgo</i>	1
Umschaltfläche <i>120</i>	<i>tgl_L120</i>	2
Umschaltfläche <i>240</i>	<i>tgl_L240</i>	3
Textfeld <i>Nummer</i>	<i>KassettenNr</i>	-
Textfeld <i>Länge</i>	<i>KassettenLängeNr</i>	-
Textfeld <i>Hersteller</i>	<i>KassettenHerstellerNr</i>	-

Tabelle 10.13

Parameter abfragen

Um einer Optionsgruppe einen neuen Wert hinzuzufügen, muss vom Anwender zunächst der neue Wert eingegeben werden. Hier soll ein neuer Hersteller hinzukommen. Der Wert wird in der Variablen *NeuerHersteller* gespeichert.

```
Dim NeuerHersteller As String
NeuerHersteller = InputBox("Geben Sie den Namen des _
neuen" & " Herstellers ein!")
```

Anschließend muss das Formular *Video* in der Entwurfsansicht geöffnet werden.

```
DoCmd.OpenForm "Video", acDesign
```

Steuerelement- daten berechnen

Die folgende Schleife berechnet sowohl die Anzahl der Optionsfelder in der Gruppe *ogr_Hersteller* als auch die letzte Position in der Gruppe. Dazu wird bei allen Steuerelementen des aktiven Formulars geprüft, ob sie aus der Gruppe *ogr_Hersteller* sind. Dies ist durch die spezielle Namensgebung für die Elemente der Optionsgruppe möglich. Die Felder sind alle durch ein großes *H* (für Hersteller) gekennzeichnet. Die letzte Position und die Anzahl der Gruppenelemente werden zur Weiterverarbeitung zwischengespeichert.

```
Anzahl = 0
y_pos = 0

For Each C In Forms!video.Controls
    If Left(C.Name, 5) = "tgl_H" Then
        Anzahl = Anzahl + 1
```





```

        If C.Top > y_pos Then
            y_pos = C.Top
        End If
    End If
Next C

```

Nun muss mit den berechneten Daten das neue Steuerelement erzeugt werden. Die Umschaltfläche wird dabei über die Hauptgruppe *ogr_Hersteller* auf die Optionsgruppe bezogen.

```

Rem Die neue Umschaltfläche erzeugen
Set C = CreateControl("Video", acToggleButton, 0, _
    "org_Hersteller", "tgl_H" &
    NeuerHersteller, _
    113, y_pos + 453, 913, 343)

```

```

Rem Eigenschaften der Umschaltfläche anpassen
With C
    .FontName = "MS Sans Serif"
    .FontSize = 8
    .FontWeight = 200
    .Name = "tgl_H" & NeuerHersteller
    .Caption = NeuerHersteller
    .OptionValue = Anzahl + 1
End With

```

Zum Schluss werden noch Eigenschaften wie der Name, die Beschriftung, Schriftart und -größe und der Optionswert für die Umschaltfläche festgelegt. Wie in *Abbildung 10.12* zu sehen ist, wurden alle Eintragungen für die neu hinzugefügte Umschaltfläche *TDK* richtig vorgenommen.

Für die fertige Anwendung fehlt noch eine Kleinigkeit. In der Regel soll der Anwender nicht merken, dass das Formular in der Entwurfsansicht geöffnet wird. Er soll nur auf eine Befehlsschaltfläche klicken, den Namen der neuen Umschaltfläche eingeben und direkt mit dem neuen Formular weiterarbeiten können. Im oben angegebenen Beispiel liegt das Formular nach der Bearbeitung aber in der Entwurfsansicht vor.

Um das zu ändern, muss die Prozedur das Formular speichern und wieder in die Formularansicht zurückwechseln. Zu diesem Zweck wird die *SendKeys*-Anweisung verwendet. Mit ihr lassen sich per Programm Tastenanschläge senden, als würden sie auf der Tastatur eingegeben. In *Tabelle 10.14* sind einige wichtige Tastencodes aufgeführt.

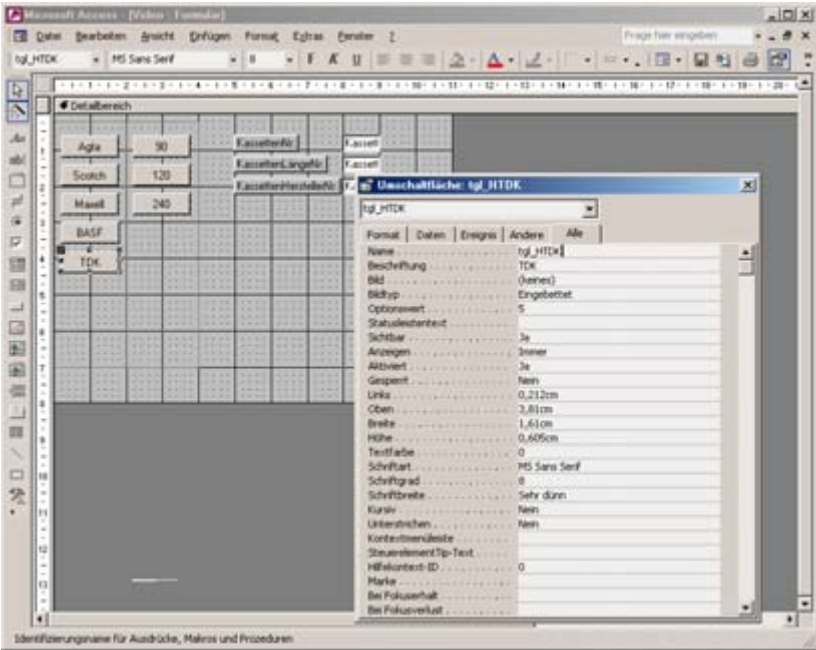
Anlegen des
Steuerelements

SendKeys-
Anweisung



Abb. 10.12

Hinzugefügte
Umschaltflächen



Tastaturcodes für die *SendKeys*-Anweisung

Taste	Code
UMSCHALTASTE	+
STRG-TASTE	^
ALT-TASTE	%
EINGABETASTE	{ENTER} oder ~
ESC	{ESC}
EINFG	{INSERT}
ENTF	{DELETE} oder {DEL}
TABULATORASTE	{TAB}
UNTBR	{BREAK}
F1 bis F16	{F1} bis {F16}

Tabelle 10.14



Die *SendKeys*-Anweisung für das Speichern des Formulars und das Wechseln in die Formularansicht lautet demnach:

```
SendKeys "%DS%AF"
```

Es werden einfach die entsprechenden Menüpunkte aktiviert, so wie Sie sie auch über Shortcuts erreichen würden. Wenn Sie ein neues Formular erzeugen und es speichern wollen, können Sie direkt den Formularnamen mit übergeben. Die einzelnen Tastenanschläge werden nacheinander aufgeführt.

```
SendKeys "%DSTestFormular{Enter}% (DC) "
```

Zuerst wird das Formular unter dem Namen *TestFormular* gespeichert und dann geschlossen.

In *Abbildung 10.13* sehen Sie eine Möglichkeit, wie dem Formular *Video* über ein Zusatzformular (*Video - Hinzufügen*) neue Hersteller hinzugefügt werden können. Nach Bestätigung der Namenseingabe kann der Anwender sofort mit dem neuen Formular weiterarbeiten.

Abb. 10.13 ○

Hinzufügen eines
Herstellers während
der Laufzeit



Das Hinzufügen einer neuen Umschaltfläche wurde hier über ein zweites Formular mit den entsprechenden Schaltflächen realisiert. Sie können zwar aus einem Formular heraus über eine Befehlsschaltfläche in die Entwurfsansicht desselben Formulars wechseln, das Hinzufügen einer Schaltfläche ist aber nicht möglich.



Hinweis

Es folgt die komplette Prozedur für das Hinzufügen einer Umschaltfläche. Es wird noch zusätzlich abgefragt, ob ein Hersteller eingegeben oder ob der Eingabedialog abgebrochen wurde.





```
Sub NeueUmschaltfläche()

    Dim NeuerHersteller As String
    Dim f As Form
    Dim C As Control
    Dim I As Integer
    Dim y_pos As Integer
    Dim Anzahl As Integer

    Rem Neuen Herstellernamen eingeben
    NeuerHersteller = InputBox("Geben Sie den Namen "
& _
        "des neuen Herstellers ein!", _
        "Hersteller hinzufügen")
    If NeuerHersteller <> "" Then

        Rem In die Entwurfsansicht wechseln
        DoCmd.OpenForm "Video", acDesign

        Rem Anzahl der Optionsgruppenelemente berechnen
        Rem Position des letzten Optionsgruppenelements
        Rem speichern
        Anzahl = 0
        y_pos = 0

        For Each C In Forms!video.Controls
            If Left(C.Name, 5) = "tgl_H" Then
                Anzahl = Anzahl + 1
                If C.Top > y_pos Then
                    y_pos = C.Top
                End If
            End If
        Next C

        Rem Die neue Umschaltfläche erzeugen
        Set C = CreateControl("Video", 122, 0, _
            "org_Hersteller", "tgl_H" &
            NeuerHersteller, _
            113, y_pos + 453, 913, 343)

        Rem Eigenschaften der Umschaltfläche anpassen
        With C
            .FontName = "MS Sans Serif"
            .FontSize = 8
            .FontWeight = 200
            .Name = "tgl_H" & NeuerHersteller
            .Caption = NeuerHersteller
            .OptionValue = Anzahl + 1
        End With
    End If
End Sub
```





```

        Rem Formular speichern und in die
        Formularansicht
        Rem wechseln
        SendKeys "%DS%AF"

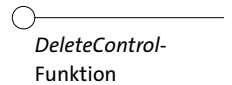
    End If

End Sub

```

Steuerelemente löschen

Selbstverständlich können auch Steuerelemente aus einem Formular gelöscht werden. Benutzen Sie dazu die Funktion *DeleteControl*. Sie hat die folgende Syntax:



```
DeleteControl <Formularname>, <Steuerelementname>
```

Beim Löschen eines Steuerelements ist es unerheblich, in welchem Bereich es sich befindet. Der Name des Elements reicht zum Löschen aus. Die folgende Prozedur löscht die Umschaltfläche *BASF* aus dem Formular.

```

Sub UmschaltflächeLöschen ()

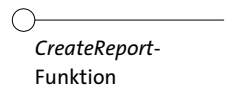
    DoCmd OpenForm "Video", acDesign
    DeleteControl "Video", "tgl_HBASF"

End Sub

```

Berichte erzeugen

Berichte lassen sich auf die gleiche Weise wie Formulare erstellen und handhaben. An die Stelle der *CreateForm*-Funktion rückt die *CreateReport*-Funktion, die denselben Aufbau hat. Es muss eine Variable vom Typ *Report* definiert werden und dann über diese Variable der neue Bericht.



```

Dim R As Report
Set R = CreateReport()

```

Der Bericht wird wieder in der Entwurfsansicht geöffnet und als minimiertes Fenster in der linken unteren Ecke des Bildschirms dargestellt. Zusätzlich zum Detailbereich beinhaltet der Bericht bereits einen Seitenkopf und einen Seitenfuß. Werden der Berichtskopf und Berichtsfuß verwendet, müssen die Bereiche wieder mit dem entsprechenden Menübefehl aktiviert werden. Der Menübefehl lautet in diesem Fall *Ansicht > Berichtskopf -fuß* und kann mit der *DoMenuItem*-Aktion im Programm ausgeführt werden:

```
DoCmd DoMenuItem 7, 2, 11, Version := acMenuVer70
```





Gruppierungen aktivieren

Gruppierungsebenen

Der Unterschied zu Formularen besteht darin, dass Berichte Gruppierungsebenen enthalten können. So können zum Beispiel alle Kunden nach Orten gruppiert werden. Jede Gruppierungsebene kann einen Gruppierungskopf und einen Gruppierungsfuß erhalten.

Diese Gruppierungsebenen werden normalerweise in der Entwurfsansicht eines Berichts über den Menüpunkt *Ansicht > Sortieren und Gruppieren* erstellt. Sowohl das Erstellen von Gruppierungsebenen als auch alle Eigenschaften, die im Dialog *Sortieren und Gruppieren* eingestellt werden können, lassen sich auch mit VBA realisieren.

CreateGroupLevel-Funktion

Mit der Funktion *CreateGroupLevel* erzeugen Sie eine neue Ebene. Die Syntax des Befehls lautet folgendermaßen:

```
CreateGroupLevel(<Berichtsname>, <Ausdruck>,
<Gruppenkopf>, <Gruppenfuß>)
```

<Berichtsname> ist ein Platzhalter für den Namen des Berichts, in dem die Gruppierungsebene erstellt werden soll. Für *<Ausdruck>* ist der Feldname einzusetzen, auf dem die Gruppierung basiert. Für *<Gruppenkopf>* und *<Gruppenfuß>* sind boolesche Werte einzusetzen. Sie entscheiden darüber, ob der entsprechende Bereich angezeigt wird oder nicht.

Die Funktion *CreateGroupLevel* gibt die Anzahl der Gruppierungsebenen zurück, die vor dem Erzeugen der neuen Ebene vorhanden waren. Neue Gruppierungsebenen werden direkt um den Detailbereich geschachtelt. Damit ergibt sich die folgende Schachtelung der Ebenen:

```
Kopfbereich der ersten Gruppierungsebene
  Kopfbereich der zweiten Gruppierungsebene
    \u
    Kopfbereich der letzten Gruppierungsebene
      Detailbereich
      Fußbereich der letzten Gruppierungsebene
        \u
        Fußbereich der zweiten Gruppierungsebene
Fußbereich der ersten Gruppierungsebene
```

Mit dem folgenden Beispiel erstellen Sie einen neuen Bericht und fügen eine neue Gruppierungsebene hinzu.

```
Sub Gruppierung ()

    Dim R As Report
    Dim Anzahl As Integer

    Rem Neuen Bericht erzeugen
    Set R = CreateReport()
```



```
Rem Neue Gruppierungsebene erzeugen
Anzahl = CreateGroupLevel(R.Name, "Ort", True,
False)
```

End Sub

Die Eigenschaften der einzelnen Bereiche eines Berichts lassen sich wie bei Formularen mithilfe der *Section*-Eigenschaft verändern. Die Bereichskennnummern für Berichte sind in *Tabelle 10.15* aufgeführt.

Bereichskennnummern für Berichte

Bereich	Wert	Konstante
Detaillbereich	0	<i>acDetail</i>
Berichtskopf	1	<i>acHeader</i>
Berichtsfuß	2	<i>acFooter</i>
Seitenkopf	3	<i>acPageheader</i>
Seitenfuß	4	<i>acPageFooter</i>
Gruppierungsebene 1, Kopfbereich	5	<i>acGroupLevel1Header</i>
Gruppierungsebene 1, Fußbereich	6	<i>acGroupLevel1Footer</i>
Gruppierungsebene 2, Kopfbereich	7	<i>acGroupLevel2Header</i>
Gruppierungsebene 2, Fußbereich	8	<i>acGroupLevel2Footer</i>
usw.	9, 10, ...	

Tabelle 10.15

Für die Gruppierungsebenen gibt es noch zusätzliche Eigenschaften. Es können alle Eigenschaften, die auch mit dem Dialog *Sortieren und Gruppieren* eingestellt werden können, verändert werden.

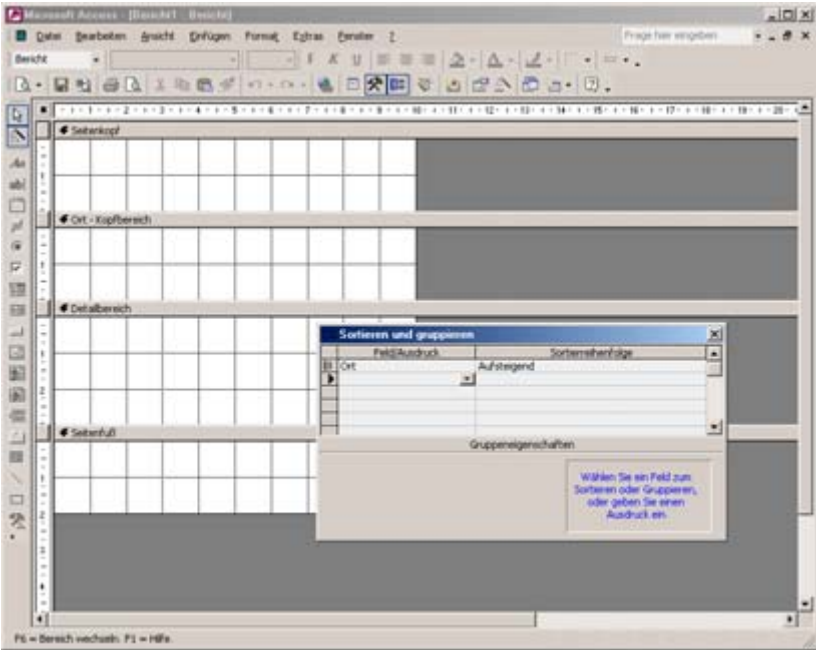
Die *Gruppieren nach*-Eigenschaft wird mit der Eigenschaft *GroupOn* definiert. Die möglichen Einstellungen, die der *GroupOn*-Eigenschaft zugewiesen werden können, finden Sie in der *Tabelle 10.16*.

GroupOn



Abb. 10.14

Der Dialog Sortieren und Gruppieren



Werte der GroupOn-Eigenschaft

Einstellung	Wert	Zulässige Datentypen
Jeder Wert	0	Text, Datum/Zeit, AutoWert, Währung, Zahl
Anfangszeichen	1	Text
Jahr	2	Datum/Zeit
Quartal	3	Datum/Zeit
Monat	4	Datum/Zeit
Woche	5	Datum/Zeit
Tag	6	Datum/Zeit
Stunde	7	Datum/Zeit
Minute	8	Datum/Zeit
Intervall	9	AutoWert, Währung, Zahl

Tabelle 10.16



Mit der Eigenschaft *GroupInterval* wird das Gruppierungsintervall einer Gruppierungsebene festgelegt. Der einzugebende Wert ist vom Datentyp *Integer*.

GroupInterval

Die Eigenschaft *KeepTogether* gibt an, welche Daten einer Gruppierungsebene zusammengehalten werden sollen. Hier gibt es drei verschiedene Einstellungen:

KeepTogether

Werte der *KeepTogether*-Eigenschaft

Einstellung	Wert
<i>Nein</i>	0
<i>Ganze Gruppe</i>	1
<i>Mit erstem Detaildatensatz</i>	2

Tabelle 10.17

Die Sortierreihenfolge kann mit der *SortOrder*-Eigenschaft eingestellt werden. Dazu braucht aber keine Gruppierung vorliegen. Es kann entweder aufsteigend oder absteigend sortiert werden.

SortOrder

Werte für die *SortOrder*-Eigenschaft

Einstellung	Wert	
<i>Aufsteigend</i>	0	(False)
<i>Absteigend</i>	-1	(True)

Tabelle 10.18

Alle vier Eigenschaften lassen sich mit der *GroupLevel*-Eigenschaft eines Berichts setzen. Mit der *GroupLevel*-Eigenschaft wird auf eine bestimmte Gruppierungsebene Bezug genommen. Die folgenden Zeilen setzen die Werte einer Gruppierungsebene:

GroupLevel

```
Reports![TestBericht].GroupLevel(0).SortOrder = 1
Reports![TestBericht].GroupLevel(0).GroupOn = 0
Reports![TestBericht].GroupLevel(0).GroupIntervall = 1
Reports![TestBericht].GroupLevel(0).KeepTogether = 2
```





*CreateReport-
Control-Funktion*

Berichtssteuerelement erzeugen

Die oben beschriebenen Eigenschaften beziehen sich alle auf die Bereiche eines Berichts. Für einen Bericht lassen sich aber ebenso wie für Formulare auch Steuerelemente erzeugen. Der Aufbau und die Syntax sind bei beiden völlig analog. Aus der *CreateControl*-Anweisung für Formulare wird die *CreateReportControl*-Anweisung für Berichte. Die Werte für die einzelnen Steuerelemente sind identisch mit denen in *Tabelle 10.12*. Das folgende Programm erzeugt einen Bericht und ein Steuerelement:

```
Sub NeuesBerichtSteuerelement()

    Dim R As Report
    Dim C As Control

    Rem Neuen Bericht erzeugen
    Set R = CreateReport()

    Rem Neues Steuerelement erzeugen
    Set C = CreateReportControl(R.Name, acTextBox, 0)

    Rem Eigenschaften des Steuerelements definieren
    With C
        .Name = "Neues Textfeld"
        .Left = 500
        .Top = 500
    End With

End Sub
```

*DeleteReport-
Control-Funktion*

Auch das Löschen von Steuerelementen vollzieht sich genauso wie bei Formularen. Die Anweisung lautet für Berichte *DeleteReportControl* und hat die folgende Syntax:

```
DeleteReportControl <Berichtname>, <Steuerelementname>
```

Entwicklungstool

Zum Abschluss des Kapitels wird nun noch ein Beispiel vorgestellt, das sowohl den Zugriff auf Tabellen über die Datenbank-Engine als auch den Eingriff in Formulare verwendet. In diesem Beispiel sollen nach Wunsch des Anwenders Felder aus vorhandenen Tabellen über einen Dialog ausgewählt und in den entsprechenden Formularen ergänzt werden.

Als Basis für das Beispiel dient die Tabelle *Adressen*. Bei den Feldern, die vom Benutzer hinzugefügt werden können, handelt es sich um eine CompuServe-ID-Nummer und eine Internetadresse. Der Anwender kann während der Laufzeit nachträglich entscheiden, ob er diese Felder in seiner Datenbankanwendung nutzen will oder nicht.



Aus Platzgründen und der Übersichtlichkeit halber beschränkt sich das Beispiel auf Formulare. Die Vorgehensweise für Berichte ist aber identisch zu der für Formulare. Außerdem werden nur Text- und Bezeichnungsfelder berücksichtigt. Auch hier lässt sich das Programm auf alle möglichen Feldtypen erweitern.



Hinweis

Zunächst wird das Tool für den Entwickler vorgestellt, mit dem automatisch alle Felder, die der Anwender später nachträglich hinzufügen kann, in einer Tabelle gespeichert werden. Anschließend wird auf den für den Anwender zur Verfügung gestellten Dialog eingegangen, mit dem der Benutzer Felder auswählen und hinzufügen kann.

Damit die Prozedur für das Entwicklertool einwandfrei läuft, sind einige Vereinbarungen einzuhalten, die die Vorgehensweise beim Erstellen der Formulare und die Namenskonventionen betreffen.



Entwicklertool

Zuerst werden alle Tabellen mit den benötigten Feldern erstellt und ggf. Abfragen definiert. Auf Basis der Tabellen und Abfragen werden dann die Formulare entworfen. Es werden alle Felder aufgenommen, auch die, die erst nachträglich vom Anwender hinzugefügt werden sollen. Bei diesen Feldern ist eine besondere Namenskonvention für dieses Beispiel zu beachten (Sie können sich auch jedes beliebige andere Erkennungszeichen überlegen und benutzen).

Entsprechend den allgemeinen Namenskonventionen, auf die im *Kapitel 11 Vorgehensweise bei der Anwendungsentwicklung* im Abschnitt *Richtlinien für Namenskonventionen* noch genau eingegangen wird, beginnen Textfelder mit `txt_` und Bezeichnungsfelder mit `lbl_`. Um die variablen Felder von den statischen zu unterscheiden, erhalten sie noch einen weiteren Unterstrich. Die Namen der variablen Felder beginnen also mit `txt__` und `lbl__`.

Beachten Sie auch, dass das Textfeld und das Bezeichnungsfeld von einer Feldvariablen bis auf die vorangestellte Kennung für den Steuerelementtyp den gleichen Namen hat. Für das Feld *CompuServe-Adresse* zum Beispiel hat das Textfeld die Bezeichnung `txt__CompuServe` und das Bezeichnungsfeld den Namen `lbl__CompuServe`.



Hinweis

Wenn alle Formulare erstellt worden sind und die Felder die richtige Kennung haben, öffnen Sie das Formular *SchlußBeispiel - Entwickler Start*. Dieses Formular wie auch das Formular *SchlußBeispiel - Felddauswahl* befindet sich in der Datenbank *Kapitel9.mdb* auf der dem Buch beiliegenden CD.



Abb. 10.15

Startformular für den Entwickler





Wenn Sie auf die Schaltfläche klicken, wird zuerst die Prozedur *TabelleErzeugen* aufgerufen. Diese Prozedur erstellt die Tabelle *SchlußBeispiel - Felder*, in der die Daten der Felder abgelegt werden, die nachträglich eingefügt werden sollen. Anschließend wird die Prozedur *DatenZuFeldernSpeichern* aufgerufen.

```
Sub DatenZuFeldernSpeichern()  
  
    Dim DB As Database  
    Dim Cont As Container  
    Dim D As Document  
    Dim C As Control  
    Dim R As Recordset  
  
    Set DB = CurrentDb  
    Set R = DB.OpenRecordset("SchlußBeispiel - Felder")  
    Set Cont = DB.Containers("Forms")  
  
    For Each D In Cont.Documents  
        If Left(D.Name, 6) <> "Schluß" Then  
            DoCmd.OpenForm D.Name, acDesign, , , ,  
                acHidden  
            For Each C In Forms(D.Name).Controls  
                Rem Speichern der Daten von  
                Rem markierten Feldern  
                If Mid(C.Name, 4, 2) = "__" Then  
                    R.AddNew  
                    R!Formularname = D.Name  
                    R!Name = C.Name  
                    R!Oben = C.Top  
                    R!Links = C.Left  
                    R!Breite = C.Width  
                    R!Höhe = C.Height  
                    R!Spezialeffekt = C.SpecialEffect  
                    R!Farbe = C.BackColor  
                    If Left(C.Name, 3) = "txt" Then  
                        R!inhalt = C.ControlSource  
                    End If  
                    If Left(C.Name, 3) = "lbl" Then  
                        R!inhalt = C.Caption  
                    End If  
                    R.Update  
                End If  
            Next C  
            DoCmd.Close acForm, D.Name  
        End If  
    Next D  
  
    Rem Löschen der Felder in den entsprechenden  
    Formularen
```





```
Rem Abfrage einfügen (Anzahl der Datensätze <> =)
If R.RecordCount <> 0 Then
    R.MoveFirst
    Do Until R.EOF
        DoCmd.OpenForm R!Formularname, acDesign, _
            , , , acHidden
        If Left(R!Name, 3) = "txt" Then
            DeleteControl R!Formularname, R!Name
        End If
        Forms(R!Formularname).SetFocus
        DoCmd.Save acForm, R!Formularname
        DoCmd.Close acForm, R!Formularname
        R.MoveNext
    Loop
End If
End Sub
```

Das Programm holt sich die Namen aller Formulare der Datenbank. Dazu wird die *Documents*-Auflistung des Containers *Forms* durchlaufen. Danach werden alle Formulare nach Feldern durchsucht, die im Namen einen doppelten Unterstrich haben. Die Eigenschaften dieser Felder werden in der zuvor erstellten Tabelle gespeichert. Im letzten Schritt werden alle Felder, die in der Tabelle abgelegt wurden, aus den entsprechenden Formularen gelöscht.

Es werden im Beispiel nur die Textfelder gelöscht, da die Bezeichnungsfelder automatisch mitgelöscht werden, wenn die Textfelder dem Formular über die Feldliste zugefügt wurden. Sollen einzelne Felder gelöscht werden, muss die Abfrage entsprechend geändert werden.



Hinweis

Damit wäre die Entwicklung abgeschlossen. Das Formular *Schlußbeispiel - Entwickler Start* und die Prozeduren *TabelleErzeugen* und *DatenZuFeldernSpeichern* können nun ruhig aus der Anwendung entfernt werden.

Dem Anwender sollte jetzt ein Formular zur Verfügung gestellt werden, mit dem er nach Wunsch Felder in einzelnen Formularen hinzufügen kann. Eine Möglichkeit zur Gestaltung ist die Verwendung von zwei Listenfeldern. Im ersten Listenfeld werden die Formulare aufgelistet, denen Felder hinzugefügt werden können. Im zweiten Listenfeld werden die zum Formular gehörenden Felder aufgeführt. Die folgende Abbildung zeigt ein solches Formular, das in der Beispieldatenbank den Namen *Schlußbeispiel - Feldauswahl* trägt.

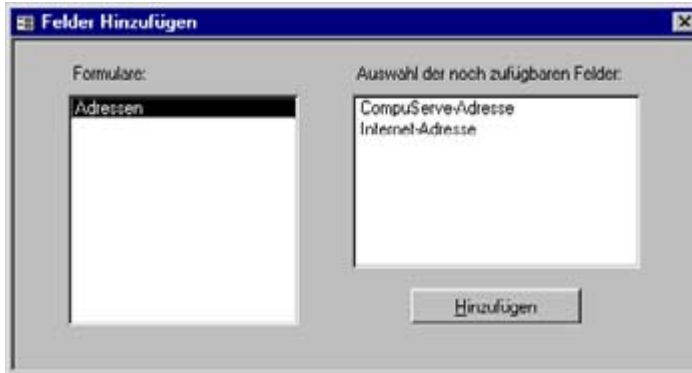


Anwenderdialog



Abb. 10.16

Formular für den
Anwender



Wird in diesem Formular ein Doppelklick auf einen Feldeintrag ausgeführt oder ein Eintrag markiert und die Schaltfläche *Hinzufügen* gedrückt, dann wird die Prozedur *FelderAnlegen* aufgerufen:

```
Sub FelderAnlegen(Inhalt As String)
```

```
    Dim DB As Database
    Dim R As Recordset
    Dim C As Control
    Dim Formname As String
```

```
    Set DB = CurrentDb
    Set R = DB.OpenRecordset("SchlußBeispiel " & _
        "- Felder", dbOpenDynaset)
    If R.RecordCount <> 0 Then
        Rem Ersten Datensatz finden
```

```
    R.FindFirst "[Inhalt] = " & Inhalt & _
        " " And left([Name],3) = "txt"
```

```
    Formname = R!Formularname
    DoCmd.OpenForm Formname, acDesign, , , , acHidden
```

```
    Rem Textfeld erzeugen
    Set C = CreateControl(Formname, 109, 0)
    With C
```

```
        .Left = R!Links
        .Top = R!Oben
        .Width = R!Breite
        .Height = R!Höhe
        .Name = R!Name
        .ControlSource = Inhalt
```

```
    End With
    R.Delete
```





```

Rem Zugehöriges Bezeichnungsfeld finden
R.FindFirst "[Inhalt] = " & Inhalt & _
    " And left([Name],3) = "lbl"

Rem Bezeichnungsfeld erzeugen
Set C = CreateControl(Formname, 100, 0)
With C
    .Left = R!Links
    .Top = R!Oben
    .Width = R!Breite
    .Height = R!Höhe
    .Name = R!Name
    .Caption = Inhalt
End With
R.Delete

Rem Formular speichern
Forms(Formname).SetFocus
DoCmd.Save acForm, Formname
DoCmd.Close acForm, Formname

Rem Listenfelder im Formular Felder Hinzufügen
Rem aktualisieren
Forms![SchlußBeispiel - Feldauswahl] _
    .lst_Formulare.Requery
Forms![SchlußBeispiel - Feldauswahl].
lst_Felder.Requery
End If
End Sub

```

Die Prozedur öffnet die Tabelle *SchlußBeispiel - Felder* und sucht in ihr nach den Feldern, die zu dem im Listenfeld gewählten Eintrag gehören. Die Felder werden nacheinander dem richtigen Formular hinzugefügt. Zum Schluss wird der zu dem Feld gehörende Datensatz aus der Tabelle gelöscht und die Listenfelder im Formular *Felder Hinzufügen* werden aktualisiert.

Zugriff auf Module

Neu seit Access 97 ist die Möglichkeit, die Module einer Datenbank via VBA zu manipulieren. Zu diesem Zweck wurde eine ganze Reihe von neuen Methoden und Eigenschaften hinzugefügt. Ähnlich wie bei Formularen und Berichten müssen auch Modulobjekte geöffnet sein, um darauf zugreifen zu können. Ob die Module schon kompiliert sind oder nicht, spielt hier keine Rolle. Zu den Standardmodulen und Formular- bzw. Berichtsmodulen kommt auch ein neuer Typ von Modul hinzu, nämlich das *Klassenmodul*, das Sie bereits im vorhergehenden Kapitel kennen gelernt haben.



Module auflisten

Geöffnete Module
im Testfenster
ausgeben

Mit der *Modules*-Auflistung haben Sie Zugriff auf alle Module einer Datenbank. Jedes Modul wird hierbei durch ein Objekt des Datentyps *Module* repräsentiert. Mit der folgenden Prozedur können Sie beispielsweise die momentan geöffneten Module zusammen mit dem Typ des Moduls im Testfenster ausgeben:

```
Sub ModuleAuflisten()

Dim strTyp As String
Dim i As Integer

For i% = 0 To Modules.Count - 1

    Select Case Modules(i).Type
        Case acStandardModule:
            strTyp = "Standardmodul"
        Case acClassModule:
            strTyp = "Klassenmodul"
        End Select

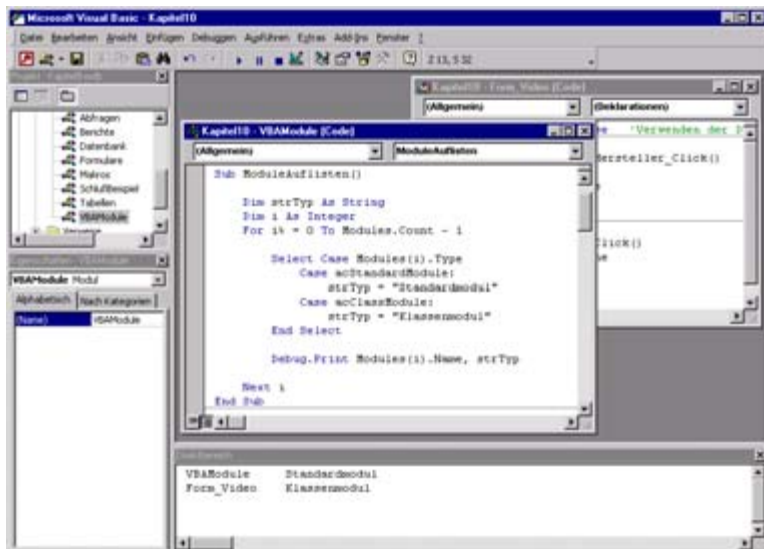
    Debug.Print Modules(i).Name, strTyp

Next i
End Sub
```

Die Module von Formularen und Berichten werden hierbei ebenfalls als *Klassenmodule* bezeichnet.

Abb. 10.17

Alle geöffneten Module
im Testfenster



Neben der *Type*-Eigenschaft stellen *Module*-Objekte noch weitere Eigenschaften bzw. Methoden zur Verfügung, die größtenteils für die Manipulation des Programmcodes zuständig sind. Die folgende Tabelle gibt Ihnen eine Übersicht:

Moduleigenschaften, die den Zugriff auf den Programmcode erlauben

Eigenschaft	Beschreibung
<i>CountOfLines</i>	Anzahl der Programmcodezeilen
<i>CountOfDeclarationLines</i>	Anzahl der Zeilen im Deklarationsbereich
<i>ProcOfLine</i>	Name der Prozedur, Funktion bzw. Eigenschaft an einer beliebigen Zeile
<i>ProcCountLines</i>	Anzahl der Zeilen der Definition einer Prozedur
<i>ProcStartLine</i>	Zeilennummer, an der eine Prozedur beginnt
<i>ProcBodyLine</i>	Zeilennummer, an der die Definition einer Prozedur beginnt
<i>Lines</i>	Eine beliebige Programmcodezeile an einer bestimmten Zeilennummer

Tabelle 10.19

Durch den Einsatz dieser Eigenschaften können Sie praktisch den gesamten Inhalt eines Moduls abfragen und ggf. manipulieren.

Die Beispieldatenbank *Kapitel10.mdb* enthält ein Formular namens *Modul-Information*, das Informationen zu einem beliebigen Modul ausgibt:

Modulinhalt
ermitteln

Abb. 10.18

Das Formular *Modul-Information* der Beispieldatenbank





Die *Ermitteln*-Schaltfläche ist an die folgende Ereignisprozedur geknüpft, die die Daten aus dem gewünschten Modul abruft:

```
Private Sub btnErmitteln_Click()

    Dim mdl As Module
    Dim lZeile As Long
    Dim strZeilentext As String
    Dim lNumProzedure As Long
    Dim lNumProzedurZeilen As Long

    Rem Modul öffnen und minimieren
    On Error Resume Next
    DoCmd.OpenModule txtModulname
    If Err <> 0 Then Exit Sub
    DoCmd.Minimize

    Set mdl = Modules(txtModulname)
    txtModultyp = ""
    txtDeclZeilen = ""
    txtGesamtZeilen = ""
    lstProzedure.RowSource = "Name;Zeile;"
    txtNumProzedure = ""
    txtAvgProzedurZeilen = ""

    " Modultyp ermitteln
    Select Case mdl.Type
        Case acStandardModule
            txtModultyp = "Standardmodul"
        Case acClassModule
            txtModultyp = "Klassenmodul"
    End Select

    txtGesamtZeilen = mdl.CountOfLines
    txtDeclZeilen = mdl.CountOfDeclarationLines

    Rem erste Prozedur ermitteln
    lZeile = mdl.CountOfDeclarationLines + 1
    strZeilentext = mdl.ProcOfLine(lZeile,
vbext_pk_Proc)
    If Len(strZeilentext) <> 0 Then
        lstProzedure.RowSource =
lstProzedure.RowSource _
            & strZeilentext & ";" & lZeile & ";"
        lNumProzedure = 1
        lNumProzedurZeilen = _
            mdl.ProcCountLines(strZeilentext, _
vbext_pk_Proc)
    End If
```





```

Rem restliche Prozeduren ermitteln
Do While lZeile <= mdl.CountOfLines
    lZeile = lZeile + 1
    If strZeilentext <> mdl.ProcOfLine(lZeile, _
        vbext_pk_Proc) Then
        strZeilentext = mdl.ProcOfLine(lZeile, _
            vbext_pk_Proc)
        lstProzedure.RowSource = _
            lstProzedure.RowSource & strZeilentext _
            & ";" & lZeile & ";"
        lNumProzedure = lNumProzedure + 1
        lNumProzedurZeilen = lNumProzedurZeilen + _
            mdl.ProcCountLines(strZeilentext, _
                vbext_pk_Proc)
    End If
Loop

txtNumProzedure = lNumProzedure
txtAvgProzedurZeilen = lNumProzedurZeilen \ _
    lNumProzedure

Rem Modul schließen
DoCmd.Close acModule, txtModulname

End Sub

```

Zuerst wird das angegebene Modul geöffnet. Dies ist notwendig, da der Zugriff nur bei geöffneten Modulen möglich ist. Danach werden die Anzahl der Zeilen im Deklarationsbereich sowie im ganzen Modul abgefragt.

Der letzte Teil der Prozedur ermittelt schließlich die Namen der enthaltenen Prozeduren sowie deren Zeilenanzahl. Aus diesen Angaben wird eine Werteliste generiert, die schließlich das Listenfeld füllt.

Module erzeugen

Den zwei verschiedenen Modultypen entsprechend gibt es zwei Möglichkeiten, ein Modul zu erzeugen. Um Standard- und Klassenmodule zu erstellen, setzen Sie die *RunCommand*-Methode des *DoCmd*-Objekts ein:

```

RunCommand acCmdNewObjectModule
\6 erzeugt ein Standardmodul
RunCommand acCmdNewObjectClassModule
\6 erzeugt ein Klassenmodul

```

Access legt das neue Modul an und öffnet es automatisch. Sie können also direkt danach mit der Manipulation des Programmcodes beginnen.

Standard- und Klassenmodule erzeugen





Ein Problem ist hierbei, dass das Modul in diesem Stadium einen willkürlich von Access zugewiesenen Namen besitzt. Zum Zugriff müssen Sie daher die *CurrentObjectName*-Eigenschaft des *Application*-Objekts einsetzen, die den Namen des momentan aktivierten Objekts zum Ergebnis hat. Dies funktioniert natürlich nur dann, wenn Sie nach der Anlage des neuen Moduls kein anderes Fenster aktivieren. Das folgende Beispiel zeigt die Vorgehensweise:

```
Sub StandardModulErzeugen()  
  
Dim mdl As Module  
  
    RunCommand acCmdNewObjectModule  
  
    Rem Neues Modul ist geöffnet und aktiv  
    Set mdl = Modules(Application.CurrentObjectName)  
  
    Rem Anzahl der Zeilen ausgeben  
    Debug.Print mdl.CountOfLines  
End Sub
```

Formular- und
Berichtsmodule
erzeugen

Neben Standardmodulen können Sie auch Klassenmodule erzeugen, die einem Formular und Bericht zugeordnet sind. Dies ist beispielsweise dann erforderlich, wenn Sie ein neues Formular mittels der *CreateForm*-Funktion erzeugen. Access legt hierbei nämlich nicht automatisch ein Formularmodul an, so dass Sie dieses bei Bedarf via VBA erledigen müssen.

Die folgende Prozedur erzeugt ein neues Formular und setzt anschließend die *HasModule*-Eigenschaft auf *True*. Hierdurch legt Access das Formularmodul an. Die Bezugnahme auf dieses Modul erfolgt schließlich mittels der *Module*-Eigenschaft des Formulars:

```
Sub FormularMitModulErzeugen(strFormularName As String)  
  
Dim frm As Form  
Dim mdl As Module  
  
    " neues Formular erstellen  
    Set frm = CreateForm(, strFormularName)  
    frm.HasModule = True      " erzeugt Modul automatisch  
    Set mdl = frm.Module  
    Debug.Print mdl.CountOfLines  
  
End Sub
```

Hinweis



Auf ähnliche Art und Weise können Sie ein Formularmodul löschen, indem Sie die *HasModule*-Eigenschaft auf *False* setzen. Diese Vorgehensweise ist allerdings mit Vorsicht zu genießen, da das Modul ohne vorherige Bestätigung gelöscht wird.



Programmcode in ein Modul einfügen

Sobald Sie ein neues Modul erzeugt haben, etwa als Standardmodul oder Formularmodul, können Sie hier neuen Programmcode einfügen. Hierzu stehen Ihnen unterschiedliche Möglichkeiten in Form verschiedener Methoden von *Module*-Objekten zur Verfügung. Die folgende Tabelle gibt Ihnen eine Übersicht:

Text einem Modul hinzufügen

Methode des *Module*-Objekts zum Einfügen von Text

Methode	Beschreibung
<i>AddFromFile</i>	Fügt einem Modul den Inhalt einer Textdatei hinzu. Der neue Text erscheint nach dem Deklarationsbereich.
<i>AddFromString</i>	Fügt einem Modul eine Zeichenkette hinzu. Der neue Text erscheint nach dem Deklarationsbereich.
<i>CreateEventProc</i>	Fügt einem Klassenmodul (z. B. eines Formulars) eine Ereignisprozedur für ein Steuerelement des Formulars hinzu. Der neue Text erscheint nach der letzten Programmzeile.
<i>InsertLines</i>	Fügt einem Modul eine Zeichenkette an der angegebenen Stelle hinzu.
<i>InsertText</i>	Fügt einem Modul eine Zeichenkette hinzu. Der neue Text erscheint nach der letzten Zeile des Moduls.

Tabelle 10.20

Die Methode *CreateEventProc* können Sie beispielsweise dazu einsetzen, die Ereignisprozedur einer Schaltfläche via VBA anzulegen. Die folgende Prozedur erwartet als Parameter einen Formularnamen und den Namen eines Steuerelements auf dem Formular. Anschließend legt sie für das Steuerelement eine *Beim Klicken*-Ereignisprozedur an:

Ereignisprozedur hinzufügen

```
Sub ClickEreignisErzeugen(strFormularName As String, _
    strControlName As String)
```

```
On Error GoTo ErrClickErzeugen
```

```
Dim frm As Form
```

```
Dim mdl As Module
```

```
Set frm = Forms(strFormularName)
```

```
Rem Sicherstellen, dass das Modul existiert
```



```
If Not frm.HasModule Then frm.HasModule = True
Set mdl = frm.Module
```

```
mdl.CreateEventProc "Click", strControlName
```

```
ExitClickErzeugen:
Exit Sub
```

```
ErrClickErzeugen:
MsgBox "Die Beim-Klicken-Ereignisprozedur konnte " _
    & "nicht erzeugt werden.", vbCritical
Resume ExitClickErzeugen

End Sub
```

Die Prozedur weist das Formular zuerst einer Variablen vom Typ *Form* zu. Über diese *Form*-Variable erfolgt der Zugriff auf das Formularmodul. Vorher muss jedoch sichergestellt werden, dass bereits ein Formularmodul existiert:

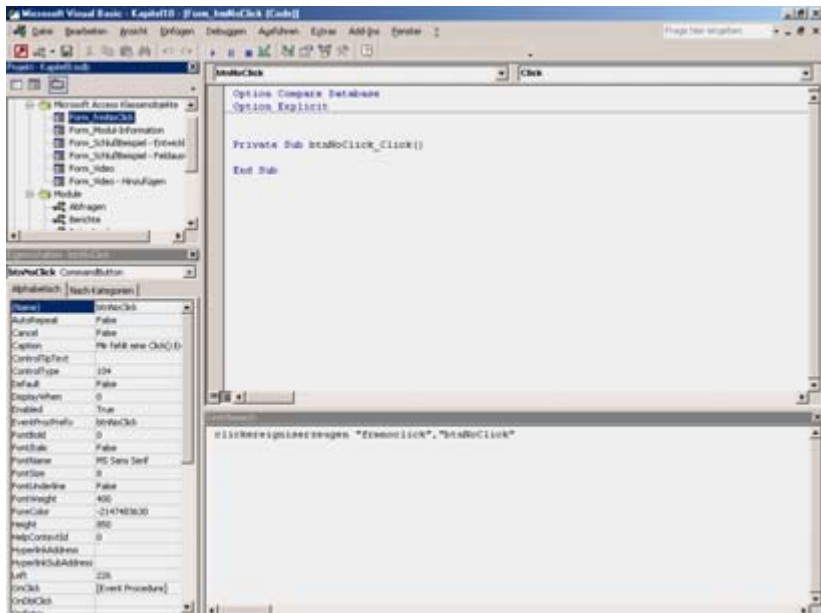
```
If Not frm.HasModule Then frm.HasModule = True
Set mdl = frm.Module
```

Die *CreateEventProc*-Methode legt dann schließlich die Ereignisprozedur an.

```
mdl.CreateEventProc "Click", strControlName
```

Abb. 10.19

Die erzeugte
Ereignisprozedur



Programmcodes verändern

Module-Objekte stellen insgesamt drei Methoden zur Verfügung, mit denen Sie Programmcodes suchen, ersetzen und löschen können:

Methoden zum Suchen, Ersetzen und Löschen von Programmcodes

Methode	Beschreibung
<i>Find</i>	Sucht nach einem beliebigen Text in einem Modul.
<i>ReplaceLine</i>	Ersetzt eine beliebige Zeile durch eine Zeichenfolge.
<i>DeleteLines</i>	Löscht beliebige Zeilen vom Modulinhalt.

Tabelle 10.21

Die folgende Beispielprozedur entfernt alle Zeilen aus einem Modul, die mit dem Kommentarzeichen beginnen. Als Parameter erwartet sie den Namen eines Moduls:

Modultext suchen
und ersetzen

```
Sub KommentareLöschen(strModulname As String)

Dim mdl As Module
Dim lAnfangszeile As Long

    Rem Modul öffnen und minimieren
    On Error Resume Next
    DoCmd.OpenModule strModulname
    If Err <> 0 Then Exit Sub
    DoCmd.Minimize

    Set mdl = Modules(strModulname)

    lAnfangszeile = 1
    Rem nach Kommentarzeichen (") suchen
    Do While mdl.Find("", lAnfangszeile, 1, _
        mdl.CountOfLines, 1)
        Rem eine Zeile löschen
        mdl.DeleteLines lAnfangszeile, 1
    Loop

    Rem Modul schließen
    DoCmd.Close acModule, strModulname, acSavePrompt

End Sub
```

In der Prozedur wird die aktuelle Zeile als Startzeile der *Find*-Methode in einer Variablen gespeichert. Wenn die *Find*-Methode eine Kommentarzeile findet, wird diese mittels der *DeleteLines*-Methode entfernt.





Vorgehensweise bei der Anwendungsentwicklung



Der Entwicklungszyklus	542
Relationale Datenbanksysteme	545
Besonderheiten beim Entwurf der Struktur	552
Replikation von Datenbanken	562
Formulare und Berichte	570
Einrichten der Benutzeroberfläche	572
Datenbanken verwalten	592
Benutzerdefinierte Hilfe	599
Das Sicherheitssystem	602





Das richtige konzeptionelle Vorgehen und eine genaue Planung spielen bei der Datenbankentwicklung eine wichtige Rolle. Dabei ist es egal, ob es sich um die Entwicklung einer kommerziellen, technischen oder ganz privaten Datenbank handelt.

In diesem Kapitel erfahren Sie nach und nach, angefangen beim Entwurf der Struktur bis zum Einrichten des Sicherheitssystems, wie Sie bei der Anwendungsentwicklung am besten vorgehen. Sie lernen, wie Sie verschiedene Tabellenstrukturen definieren, Formulare ansprechend entwerfen und eine leicht zu handhabende Benutzeroberfläche einrichten. Dazu gehören eigene Menüzeilen und Symbolleisten, aber auch die Implementierung eines Hilfesystem und das Reparieren und Komprimieren von Datenbanken. Hierbei wird für die VBA-Beispiele das Datenzugriffsmodell DAO verwendet.

Die wichtigsten Punkte bei der Anwendungsentwicklung sind das verwendete Datenbankmodell und der damit verbundene korrekte Entwurf der Struktur der Datenbank. Diese beiden Punkte beziehen Überlegungen zu Namenskonventionen, den Tabellen, in denen die Daten gespeichert werden, und den Verknüpfungen zwischen den Tabellen mit ein.

Da dies die elementaren Punkte bei der Datenbankentwicklung sind und sie den Grundstein der Datenbank bilden, stehen sie am Anfang jeder Anwendungsentwicklung und benötigen einen großen Zeitraum innerhalb des gesamten Entwicklungszyklus.

Eine sorgfältig durchdachte und geplante Datenstruktur erspart im Nachhinein viel Zeit. Sollten nämlich zum Beispiel nachträglich Feldnamen geändert werden oder neue Felder in Tabellen hinzukommen, wird es besonders bei großen Strukturen und/oder großen Datenmengen schnell unübersichtlich. Geänderte Namen müssen in jedem Formular, jeder Abfrage und jedem Modul geändert werden, und das Nachtragen der Daten eines neu hinzugekommenen Datenfeldes wird schon bei 1.000 Datensätzen nicht gerade wenig Zeit in Anspruch nehmen. Ein gut entwickeltes Konzept ist folglich der Garant für ein schnelles und fehlerfreies Gelingen der restlichen Entwicklungsschritte.

Der Entwicklungszyklus

Im Folgenden wird der Entwicklungszyklus für ein Datenbanksystem allgemein und anhand von Beispielen dargestellt und erläutert. Die meisten Beispiele aus diesem Kapitel beziehen sich auf die Datenbank *Kapitel1.mdb*. Bei allen anderen Beispielen ist die Datenbank, auf die sich das jeweilige Beispiel bezieht, mit angegeben. Sie finden alle Beispieldatenbanken auf der dem Buch beiliegenden CD.

Das nachstehende Flussdiagramm verdeutlicht das allgemeine Vorgehen bei der Entwicklung.



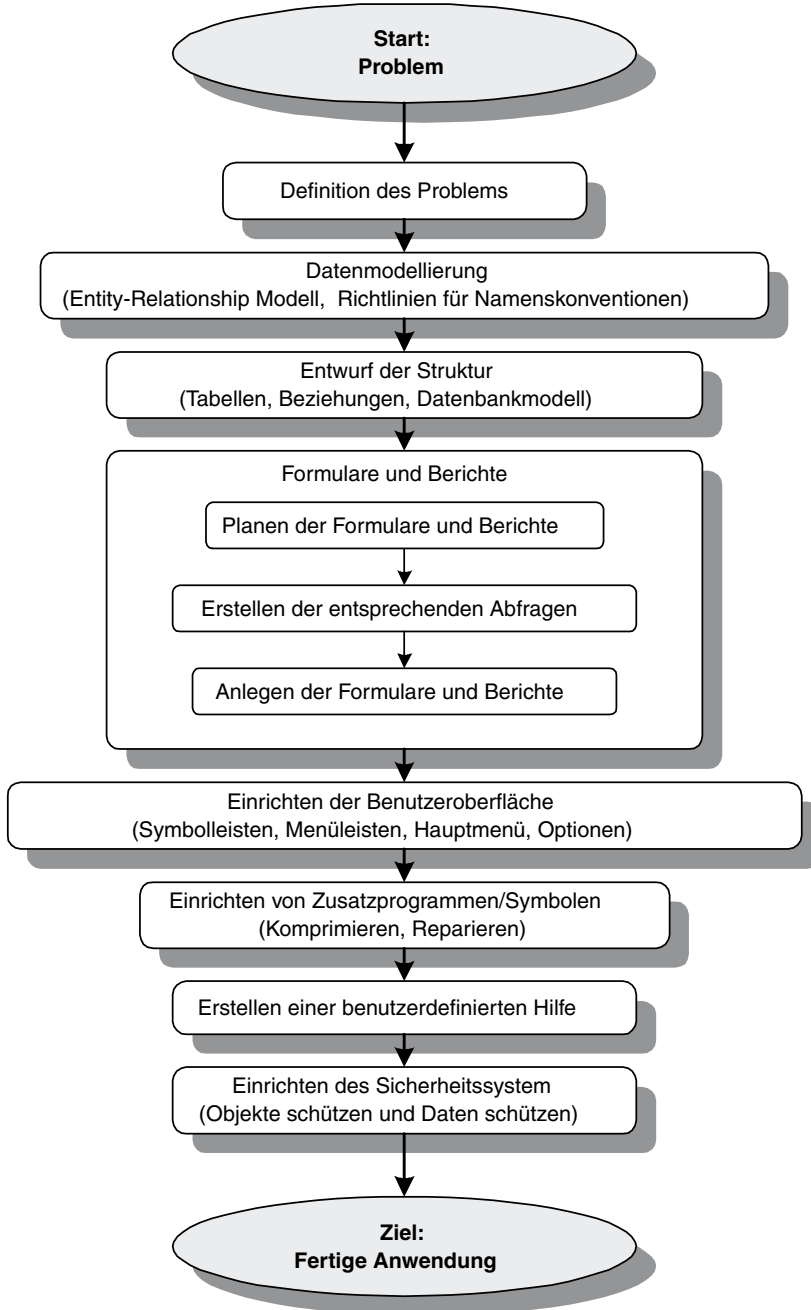


Abb. 11.1

Der Entwicklungszyklus einer Datenbank unter Access





Definition des Problems

Die Grundlage für jede Datenbank ist die genaue Definition des vorliegenden Problems. Dieses Problem ist so exakt wie möglich darzustellen; gleichzeitig muss die Darstellung aber so kurz sein, dass keine Parameter berücksichtigt werden, die für die spätere Anwendung bzw. die Problemlösung nicht relevant sind.

Abbildung der Realität

Die Lösung des Problems liegt demnach darin, die Realität im Computer abzubilden. Das Wesentliche dieser Abbildung ist, dass sie die Realität unvollständig darstellt. Diese Unvollständigkeit gehört zu den typischen Eigenschaften einer Abbildung. Sie beschränkt sich nur auf das Wesentliche, weil die Realität zu umfangreich ist. Dieses Wesentliche ist gleichzeitig aber auch das Nötige und Ausreichende zur Problemlösung. Ziel ist es also, ein Abbild der Realität zu schaffen, das alle notwendigen Informationen enthält und alle überflüssigen Daten vernachlässigt.

Um ein vorliegendes Problem in ein vernünftiges Datenmodell umzuformen, erfolgt die Vorgehensweise in mehreren Teilschritten. Im ersten Schritt wird der für das Problem vorliegende relevante Ausschnitt der Realität abgegrenzt. Für den Fall der Fakturierungsdatenbank *Kapitel11.mdb* sind die interessierenden Daten die Kunden, Lieferanten, Artikel und Rechnungen. Diese Daten bilden für die Datenbank *Kapitel11.mdb* den gewünschten Ausschnitt der Realität.

Objekte und Beziehungen definieren

In einem zweiten Schritt sind die Objekte des Realitätsausschnitts und ihre Beziehungen zu definieren. So liegen hier zum Beispiel die Objektklassen *Kunde* und *Artikel* vor. Die Beziehung zwischen den beiden Objekten ist eindeutig, da ein Kunde einen oder mehrere Artikel kauft. Zwischen den Objekten besteht deshalb die Beziehung "kauft". In der Datenbank *Kapitel11.mdb* sind die Artikel über die Rechnungsnummer und die Rechnungsposition mit den Kunden verknüpft. Ein Sonderfall liegt bei den Objekten *Artikel* und *Lieferanten* vor. Sie sind über zwei Beziehungen miteinander verbunden. Der Grund dafür ist, dass ein Artikel von mehreren Lieferanten geliefert werden und gleichzeitig ein Lieferant mehrere Artikel senden kann.

Eigenschaften der Objekte festlegen

Jedes Objekt besitzt viele verschiedene Eigenschaften. Im dritten Schritt kommt es darauf an, die wichtigen Eigenschaften der Objekte herauszufinden und den Objekten zuzuordnen. So sind für das Objekt *Kunde* zwar der Name, die Adresse und die Telefonnummer von Bedeutung, die Schuhgröße (es sei denn, es handelt sich um ein Schuhgeschäft) allerdings interessiert nicht. Für jeden Anwendungsfall sind natürlich andere Eigenschaften von Bedeutung, auch wenn es sich um dasselbe Objekt (z. B. Kunden) handelt. Für das Datenbankmodell sind ausschließlich die relevanten Eigenschaften zu speichern, die für die konkrete Lösung des Problems nützlich sind.

Datenmodellierung

Die oben beschriebenen drei Schritte werden *Datenmodellierung* genannt. Als Ergebnis erhält man ein konzeptionelles Datenschema, das alle Objektklassen und ihre relevanten Eigenschaften beinhaltet. Darüber hinaus beschreibt das konzeptionelle Datenschema die Beziehungen zwischen den Objektklassen. Wie schon in der Einleitung erwähnt, sollten Sie sich für die Modellierung sehr viel Zeit nehmen, um wirklich alle Objektklassen mit allen relevanten Eigenschaften und Beziehungen zu erfassen. Die Umsetzung des Schemas in eine Datenbank ist mithilfe des relationalen Datenbanksystems Access nur ein kleiner Schritt und in kurzer Zeit erledigt.



Relationale Datenbanksysteme

Access gehört zu der Klasse der relationalen Datenbank-Managementsysteme, auch kurz *relationale Datenbanksysteme* genannt. Das relationale Datenbankmodell wurde 1970 von E.F. Codd entwickelt und besticht im Vergleich zu seinen Vorgängern durch hohe Flexibilität und einfache Handhabung. Das damalige Manko, bei großen Datenmengen hohe Geschwindigkeitsverluste zu haben, fällt aufgrund der heutigen Prozessorleistungen nicht mehr so stark ins Gewicht.

Das wichtigste Merkmal einer relationalen Datenbank ist das Speichern der Daten in Tabellenform. Geht man vom konzeptionellen Datenschema aus, entspricht jede Objektklasse einer Tabelle. Die Eigenschaften für jedes Objekt dieser Objektklasse werden in den Feldern der Tabelle gespeichert. Ein weiteres wichtiges Merkmal von relationalen Datenbanksystemen ist die Existenz eines Primärschlüssels. Jede Tabelle kann solch einen Primärschlüssel zur eindeutigen Identifikation jedes Datensatzes enthalten. Sekundärschlüssel wie Indizes sind ebenfalls Bestandteil einer relationalen Datenbank.

Des Weiteren werden relationale Datenbanksysteme durch ihre relationalen Verknüpfungen bestimmt, d.h. durch die Beziehungen zwischen den einzelnen Tabellen. Eine Verknüpfung erfolgt über Referenzfelder, die in beiden Tabellen enthalten sein müssen. Stimmen die Werte in den beiden Feldern überein, verbindet Access die entsprechenden Datensätze. Die Tabellen *Kunden* und *Rechnungen* sind beispielsweise über das Referenzfeld *KundenNr* miteinander verknüpft. Liegt eine bestimmte Rechnung vor, wird dieser mittels der Beziehung über das Feld *KundenNr* die Adresse des entsprechenden Kunden zugeordnet.

Es handelt sich dabei um eine *1:n-Beziehung*: Jedem Kunden (1) sind mehrere Rechnungen (n) zugeordnet. Die Referenzfelder haben allerdings in den beiden Tabellen jeweils eine andere Bedeutung. In der Tabelle *Kunden* ist das Feld *KundenNr* als Primärschlüssel definiert, in der Tabelle *Rechnungen* als Fremdschlüssel. Der Grund dafür ist, dass zu jeder Rechnung genau ein Kunde vorhanden sein muss. Das Feld *KundenNr* in der Tabelle *Kunden* muss folglich eindeutig definiert sein. In der Tabelle *Rechnungen* treten dagegen Duplikate auf.

○ 1:n-Beziehung

Ziel der referentiellen Integrität ist es, zu verhindern, dass Datensätze erstellt werden, zu denen keine Referenz vorhanden ist, und keine Referenzen gelöscht werden, zu denen noch Datensätze vorhanden sind. Es macht zum Beispiel wenig Sinn, eine Rechnung zu erstellen, die keinem Kunden zugeordnet ist, da so ein überflüssiger Datensatz entstehen würde. Folglich muss also ein Kunde vorhanden sein, dem die Rechnung zugeordnet werden kann. Ebenso wenig ist es wünschenswert, dass ein Kunde gelöscht wird, zu dem noch Rechnungen vorhanden sind. Demnach müssen erst alle Rechnungen des entsprechenden Kunden gelöscht sein, damit der Kunde entfernt werden kann. Die Überprüfung und Einhaltung dieser Regeln sichert die referentielle Integrität. Die Definition einer relationalen Verknüpfung mit referentieller Integrität unter Access beeinflusst somit die Eingabe und das Löschen von Datensätzen positiv und vermeidet Fehler.

○ Referentielle Integrität





Bei der Umsetzung des konzeptionellen Datenbankmodells in eine Datenbank unter Access sind noch zwei weitere Schritte zu beachten. Zum einen sollten Sie ein **Entity-Relationship-Diagramm** (*ER-Diagramm*) entwerfen, in dem das Datenbankmodell grafisch dargestellt wird. Zusätzlich ist es ratsam, bestimmte Namenskonventionen bei der Vergabe von Tabellen-, Formular-, Steuerelementnamen usw. zu beachten und einzuhalten. Dies vereinfacht die nachträgliche Bearbeitung zum Beispiel durch andere Personen oder bei späteren Änderungen erheblich. In den folgenden beiden Abschnitten wird auf diese Schritte genauer eingegangen.

Das Entity-Relationship-Modell

Große Datenbankprojekte bestehen aus einer Vielzahl von Tabellen. Diese Tabellen können sehr komplex miteinander verbunden sein. Wird die Anzahl der Tabellen zweistellig, ist das Arbeiten ohne eine schriftliche Dokumentation kaum mehr möglich. Bevor Sie die praktische Umsetzung mit Access beginnen, sollten Sie daher ein konzeptionelles Datenschema in grafischer Form auf dem Papier dokumentieren.

Entity-Relationship-Diagramme für die grafische Darstellung

Für die Darstellung des konzeptionellen Datenschemas gibt es verschiedene Datenmodelle. Das **Entity-Relationship-Modell** (*ERM*) gehört zu den bekanntesten Datenmodellen. Falls Sie häufig mit großen Datenbanken arbeiten, können Sie für den Entwurf des konzeptionellen Datenschemas statt Bleistift und Papier auch Software einsetzen. Für den Entwurf großer **Entity-Relationship-Diagramme** (*ERD*) bieten zahlreiche Hersteller so genannte *CASE-Tools* für die Datenmodellierung an. Die Abkürzung *CASE* steht für **C**omputer **A**ided **S**oftware **E**ngineering, was übersetzt computergestützte Softwareentwicklung heißt.

Gegenstände und Beziehungen

Das Entity-Relationship-Modell wurde 1976 von P. Chen vorgestellt. Mit ihm kann das konzeptionelle Datenschema grafisch in Form eines ER-Diagramms dargestellt werden. Es beinhaltet die Entitäten (*Entity* = Gegenstand) und die Beziehungen (*Relationship*). Die Begriffe Entität und Datensatz sind synonym. Bei Entitäten handelt es sich wie bei Datensätzen um Objekte, die in einer Zeile der Datenbanktabelle durch ihre Eigenschaften beschrieben werden, zum Beispiel ein Kunde mit Name, Wohnort, Telefon und so weiter.

Entitätstypen

Um festzulegen, durch welche Eigenschaften die Objekte beschrieben werden, definieren Sie für die verschiedenen Objektklassen so genannte *Entitätstypen*. Jeder Entitätstyp kennzeichnet die Eigenschaften gleichartiger Entitäten und ist damit das Äquivalent zu den Tabellen einer Datenbank. *Abbildung 11.2* zeigt den Zusammenhang zwischen Entitätstypen, Entitäten und Feldern.

Beziehungstypen

Zusätzlich werden mit dem ER-Modell die Beziehungen zwischen den verschiedenen Entitäten beschrieben. Man unterscheidet dabei die konkrete Beziehung zwischen zwei Objekten und den Beziehungstyp. Der Beziehungstyp legt die Referenzfelder fest, über die die beiden Objekte miteinander verknüpft sind. Die konkrete Beziehung entsteht aber erst dann, wenn in die Referenzfelder Werte eingetragen werden.



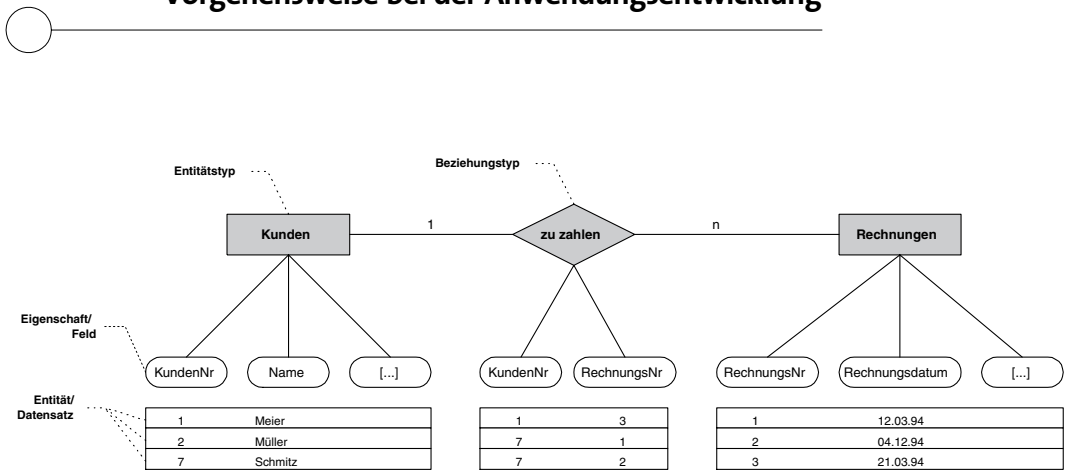


Abb. 11.2

In Abbildung 11.2 sehen Sie beispielsweise den Beziehungstyp *zu zahlen*. Über die Referenzfelder *KundenNr* und *RechnungsNr* verknüpft dieser Beziehungstyp die Entitätstypen *Kunden* und *Rechnungen*. Konkrete Beziehungen bestehen zwischen Kunde Nr. 1 und Rechnung Nr. 3 sowie zwischen Kunde Nr. 7 und Rechnung Nr. 1 und 2.

Das ER-Modell bildet Beziehungen zwischen Datenbanktabellen ab

Die Entitäts- und Beziehungstypen des ER-Diagramms werden durch beschriftete Kanten verbunden. Die Beschriftung der Kanten gibt die so genannte *Komplexität* der Beziehung an. Bei der Beziehung zwischen *Kunden* und *Rechnungen* handelt es sich um eine 1:n-Komplexität, die man auch als Eins-zu-Viele-Beziehung bezeichnen kann.

Komplexität einer Beziehung

Neben der 1:n-Beziehung werden im ER-Modell noch weitere Komplexitäten unterschieden. Wenn zwischen zwei Tabellen eine 1:1-Beziehung besteht, verknüpft Access einen Datensatz aus der einen Tabelle mit einem Datensatz aus der anderen Tabelle. Die zweite Möglichkeit ist eine Komplexität in Form einer *n:m-Beziehung*: Jeder Lieferant kann mehrere (*m*) Artikel auf Lager haben. Umgekehrt finden sich zu einem Artikel vielleicht mehrere (*n*) Lieferanten. In diesem Fall handelt es sich um eine typische n:m-Beziehung.

1:1-Komplexität, n:m-Komplexität

Eine 1:1-Beziehung bzw. 1:n-Beziehung können Sie mithilfe der Referenzfelder aus beiden Tabellen definieren, indem Sie den Befehl *Extras > Beziehungen* aufrufen und die Felder verbinden. Den Beziehungstyp legt Access automatisch fest. Auf die Definition einer n:m-Beziehung wird noch in einem der folgenden Abschnitte eingegangen.

Der entscheidende Vorteil eines ER-Diagramms besteht darin, dass Sie sich bei der Gestaltung des Diagramms Gedanken über die Datenstruktur und die Beziehungen der Daten untereinander machen, lange bevor Sie mit der Realisierung der Datenbank unter Access beginnen. Auf diese Weise vermeiden Sie Fehler in der Datenbankstruktur, die nachträglich nur sehr schwer zu beseitigen sind. Auf der Basis eines ER-Diagramms ist der Aufbau einer Datenbank mit Access denkbar einfach.

Sinn und Zweck eines ER-Diagramms

Der zweite positive Aspekt des ER-Diagramms ist die normierte Darstellung der Datenbankstruktur, die sowohl die technische als auch die fachliche Seite einer Datenbankanwendung verdeutlicht. Ein ER-Diagramm bietet für Abteilungsleiter





und für Softwareentwickler eine gemeinsame Verständigungsgrundlage, obwohl beide ansonsten mit unterschiedlichen Fachbegriffen arbeiten. So vermeiden ER-Diagramme Missverständnisse unter den Mitgliedern eines Softwareprojekts.

ER-Diagramme zur Dokumentation

Darüber hinaus bilden ER-Diagramme eine gute Dokumentation der Datenbank. Das Wissen um den Aufbau der Anwendung sollte mithilfe einer ausführlichen Dokumentation personenunabhängig zur Verfügung stehen. Gerade für Unternehmen ist diese Dokumentation von strategischer Bedeutung: Fehlt eine Dokumentation, konzentriert sich das Wissen auf wenige Softwareentwickler, was zu hohen Kosten führt, wenn diese Mitarbeiter das Unternehmen verlassen. Bereits vorhandenes und bezahltes Knowhow muss dann wieder neu erarbeitet und finanziert werden.

Richtlinien für Namenskonventionen

Mit wachsender Anzahl von Datenbankobjekten wie Tabellen, Formularen, Abfragen etc. wächst auch die Unübersichtlichkeit, wenn keine passenden Namensgebungen eingehalten werden. Eine Zuordnung des Objekts *Kunden* zu einer Objektklasse fällt schwer, da nicht klar ist, ob es sich um eine Tabelle oder ein Formular handelt. Ebenso wenig ist beim Objekt *Kunden - Nach Name sortiert* ersichtlich, ob es sich um ein Formular oder eine Abfrage handelt.

Eindeutige Kennzeichnung von Objekten

Zur eindeutigen Identifikation aller Objekte sind deshalb Namenskonventionen nötig, die von der Verallgemeinerung weggehen und eine Spezialisierung der Objekte vornehmen. Jedes Objekt lässt sich so auch noch nach längerer Zeit oder von fremden Personen seiner Objektklasse zuordnen.

Spezialisierung von Namen

Des Weiteren sollte bei der Namensgebung darauf geachtet werden, dass jeder beschreibende Name von links nach rechts mehr und mehr spezialisiert wird. So sollten zum Beispiel die Felder für private und geschäftliche Telefonnummern nicht *txt_privatTelefon* und *txt_geschäftTelefon* heißen, sondern *txt_Telefon_Privat* und *txt_Telefon_Geschäft*. Bei der alphabetischen Auflistung der Objekte im Datenbankfenster werden durch diese Vorgehensweise zusammengehörende Objekte hintereinander gruppiert.

Die Namen von Objekten sollten möglichst aussagekräftig sein. Die Länge ist dabei nicht immer entscheidend. Der Name *Verkaufsbericht für 1994 von Januar bis Dezember* beschreibt die Sache zwar genau, *rep_Verkäufe_94* sagt aber das Gleiche aus und ist wesentlich knapper und damit einfacher zu handhaben.

Die folgenden Tabellen listen für alle Datenbankobjekte die entsprechende Namenskonvention auf. Es handelt sich dabei nur um Vorschläge. Jedem Objekt ist ein Präfix zugeordnet. Das bedeutet, dass dem eigentlichen Namen das Präfix und ein Unterstrich () vorangestellt wird, zum Beispiel *tab_MeineTabelle*.

Datenbankobjekte

Bei den Datenbankobjekten sind neben den in der Tabelle aufgelisteten Konventionen noch Besonderheiten zu beachten, die sich auf Tabellen und Makros beziehen.



Namenskonventionen für Datenbankobjekte

Objekt	Präfix
Tabelle	<i>tab</i>
Abfrage	<i>qry</i>
Formular	<i>frm</i>
Bericht	<i>rep</i>
Makro	<i>mac</i>
Module	<i>mod</i>
Klassenmodule	<i>cls</i>

Tabelle 11.1

Allen Datenbankobjekten, die nicht mehr benötigt werden, aber trotzdem nicht gelöscht werden sollen, ist ein x voranzustellen. Dadurch werden sie im Datenbankfenster automatisch nach unten sortiert.

Tabellennamen ist jeweils ein z voranzustellen, wenn sie von den eigentlichen Datenbanktabellen zu unterscheiden sein sollen und als eine Art Systemtabelle deklariert werden. Dies bezieht sich zum Beispiel auf Tabellen, die für ein Hauptmenü oder für Optionen benötigt werden, da der Anwender die Tabellen nur indirekt benutzt.

Wenn Sie Kompatibilität zu Datenbanken gewährleisten wollen, die Tabellen als DOS-Dateien speichern, dürfen die Tabellennamen maximal aus acht Zeichen bestehen. Lange und aussagekräftige Namen sind dann allerdings nicht mehr möglich.



Hinweis

Für Makros gilt allgemein das Präfix *mac*. Ausnahme bilden das *Autoexec*- und das *Autokeys*-Makro. Des Weiteren sind Makros, die sich auf Menüs und Symbolleisten beziehen, durch spezielle Kennzeichnung von anderen Makros zu unterscheiden. Makros, die in Verbindung mit Symbolleisten stehen und zum Beispiel für das Ein- und Ausblenden von Symbolleisten zuständig sind, ist das Präfix *sym* voranzustellen. Makros, in denen Menüleisten gespeichert werden, können das Präfix *men*, *menu* oder *menü* erhalten.

Tabellenfelder

Tabellenfelder lassen sich ebenso wie Datenbankobjekte mit einem Präfix kennzeichnen. Eine Ausnahme bildet der Felddatentyp *AutoWert*.



Namenskonventionen für Tabellenfelder

Felddatentyp	Präfix
<i>AutoWert</i>	Erhält kein Präfix, sondern <i>Nr</i> am Ende des Feldnamens.
<i>Text</i>	<i>txt</i>
<i>Memo</i>	<i>mem</i>
<i>Zahl</i>	<i>num</i>
<i>Datum/Zeit</i>	<i>dat</i>
<i>Währung</i>	<i>cur</i> (für Currency)
<i>Ja/Nein</i>	<i>boo</i> (für Boolean)
<i>OLE-Objekt</i>	<i>ole</i>
<i>Hyperlink</i>	<i>lnk</i>

Tabelle 11.2

Primärschlüssel- kennung

Das *AutoWert*-Feld weist eine besondere Namensgebung auf. Primärschlüsselfelder werden meistens in der Form *Schlüsselname-Nr* geschrieben. Auch der Tabellenassistent von Access schlägt bei manchen Primärschlüsselfeldern diese Namensgebung vor.

Dies hat aber zur Folge, dass der Feldname in VBA oder SQL in eckige Klammern gesetzt werden muss. Um diesen Mehraufwand, der bei Vergessen der eckigen Klammern zu lästigen Fehlern führt, auszuschalten, werden Primärschlüsselfelder grundsätzlich in der Form *SchlüsselnameNr* geschrieben, also ohne Bindestrich.

Auch wenn Sie Ihre Tabellen mit dem Tabellenassistenten erstellen, haben Sie die Möglichkeit, den Primärschlüsselnamen entsprechend abzuwandeln. Markieren Sie dazu im ersten Dialog des Assistenten im Listenfeld *Felder der neuen Tabelle* das Primärschlüsselfeld und klicken Sie auf die unter dem Listenfeld befindliche Schaltfläche *Feld Umbenennen*. In dem sich öffnenden Dialog können Sie den Bindestrich löschen.

Die oben beschriebene Namenskonvention wird sowohl in der Beispieldatenbank als auch in allen anderen Beispielen beibehalten. Im Folgenden werden nun die speziellen Datenstrukturen behandelt.

Hinweis



Ist Kompatibilität zu anderen Datenbanken gefragt, gilt für die Länge von Feldnamen von Tabellen Ähnliches wie für die Tabellen selbst: Sie darf nur so viel Zeichen betragen wie die Datenbank, zu der Kompatibilität bestehen soll, zulässt.



Steuerelemente

Steuerelemente, wie sie in Formularen und Berichten vorkommen, erhalten zur eindeutigen Identifikation des Steuerelementtyps die in *Tabelle 11.3* aufgeführten Präfixe.

Namenskonventionen für Steuerelemente

Steuerelementtyp	Präfix
Bezeichnungsfeld	<i>lbl</i> (für Label)
Textfeld	<i>txt</i>
Optionsgruppe	<i>ogr</i>
Umschaltfläche	<i>tgl</i> (für Toggle)
Optionsfeld	<i>opt</i>
Kontrollkästchen	<i>chk</i>
Kombinationsfeld	<i>com</i>
Listenfeld	<i>lst</i>
Diagramm	<i>dgr</i>
Bild	<i>img</i> (für Image)
Unterformular > Bericht	<i>sub</i>
Objektfeld	<i>obj</i>
gebundenes Objektfeld	<i>gob</i>
Linie	<i>lin</i>
Rechteck	<i>rec</i>
Seitenumbruch	<i>pbr</i> (für Page break)
Befehlsschaltfläche	<i>btn</i> (für Button)
Register	<i>tab</i>

Tabelle 11.3

Variablen in VBA

Bei der Benennung von Datentypen wird dem Namen das Präfix ohne Unterstrich vorangestellt. Der Datentyp gehört somit direkt zur Variablen. Für die Variable *Anzahl* vom Datentyp *Integer* ist zum Beispiel der Name *iAnzahl* zu wählen.



Namenskonventionen für Datentypen

Datentyp	Präfix
Byte	<i>by</i>
Boolean	<i>b</i>
Integer	<i>i</i>
Long	<i>l</i>
Single	<i>n</i>
Double	<i>d</i>
Currency	<i>c</i>
Date	<i>da</i>
Object	<i>o</i>
String (variabel)	<i>sv</i>
String (fest)	<i>sf</i>
Variant (mit Zahlen)	<i>vn</i>
Variant (mit Zeichen)	<i>vc</i>
Benutzerdefiniert	<i>u</i>

Tabelle 11.4

Private- und Public-Variablen

Um unterscheiden zu können, ob es sich um private oder öffentliche Variablen handelt, ist vor den Variablennamen entweder das Präfix *m* (*Member*) oder *p* (*Public*) zu stellen. Ist *iAnzahl* als *Private* deklariert, lautet der vollständige Name *m_iAnzahl*. Als *Public*-Variable erhält sie den Namen *p_iAnzahl*. Handelt es sich um eine Konstante, wird vor dem Unterstrich zusätzlich ein *c* eingefügt (z.B. *pc_iAnzahl* = **Public**Constante_IntegerAnzahl).

Besonderheiten beim Entwurf der Struktur

Zum Entwurf der Struktur benutzt man das konzeptionelle Datenschema bzw. die Methode des Entity-Relationship-Modells. Zu dieser Grundstruktur gibt es noch weitere Techniken und Besonderheiten, die vor allem bei der Datenbankentwicklung mit Access eine Rolle spielen. Die Möglichkeiten beziehen sich im Besonderen auf spezielle Tabellenstrukturen und Beziehungen und Überlegungen zur Gesamtstruktur der Datenbank.



Tabellenstrukturen

Nach dem ER-Modell wird jeder Objektklasse eine Tabelle zugeordnet, in der die Eigenschaften der Objekte gespeichert werden. Bestehen zwischen diesen Tabellen Beziehungen, dann können diese, zum Beispiel um die referentielle Integrität zu wahren, über relationale Verknüpfungen definiert werden. Manchmal kann es aber auch sinnvoll sein, für zwei Objekte, die nicht oder nur bedingt miteinander in Verbindung stehen, eine Verknüpfung herzustellen. Diese Verknüpfung wird dann über eine Zentraltabelle realisiert, die die einzelnen Tabellen miteinander verbindet. Das Ziel ist dabei die Vermeidung von Redundanzen und eine einfachere Handhabung bei der Programmierung.

Als Beispiel soll eine Videodatenbank dienen. In dieser Datenbank sollen sowohl aufgenommene als auch gekaufte Videos verwaltet und archiviert werden. Aufgenommene Videos unterscheiden sich aber von gekauften Videos in einigen Eigenschaften. Gekauften Videos brauchen keine Kassettennummern zugewiesen zu werden. Auch die Länge der Kassette und die Restspielzeit sind nicht von Bedeutung. Zusätzlich fallen Funktionen wie das Drucken von Etiketten weg, und eine Restzeitliste muss auch nicht berechnet werden.

Es scheint also auf den ersten Blick sinnvoll, aufgenommene und gekaufte Videos als zwei Objektklassen mit jeweils einer Tabelle zu berücksichtigen. Diese Tabellen stehen nicht miteinander in Verbindung. Das Erstellen von Formularen, in denen die Daten eingegeben werden, und zwar für beide Objektklassen getrennt, lässt sich leicht bewerkstelligen. Auch die Restspielzeit ist mit der entsprechenden VBA-Funktion einfach zu berechnen, da nur die Tabelle für aufgenommene Videos verwendet werden muss.

Spätestens aber, wenn Sie Listen drucken wollen oder nach einem Filmtitel suchen, fällt auf, dass dies nicht mehr so einfach zu programmieren ist. Getrennte Listen zu drucken, ist kein Problem, weil auch getrennte Tabellen vorliegen. Meistens möchte man allerdings eine Liste mit allen Titeln, also gekauften und aufgenommenen, nach Namen sortiert vorliegen haben.

Um das zu erreichen, müsste man eine neue temporäre oder auch feste Tabelle anlegen, die alle Eigenschaften, die für die Liste relevant sind, aus beiden Tabellen beinhaltet. Anschließend müssen alle vorhandenen Datensätze in diese Tabelle kopiert werden. Das Suchen nach Titeln geschieht ebenfalls mithilfe dieser Tabelle.

Diese Lösung des Problems hat aber zwei gravierende Nachteile. Erstens treten Redundanzen in den Daten auf. Zu jedem Datensatz werden die Eigenschaften, die in der zu druckenden Liste auftauchen sollen, zweimal in der Datenbank gespeichert. Das sollte aber gerade durch das konzeptionelle Datenschema und die Aufteilung der Tabellen in viele Untertabellen vermieden werden. Zweitens sind starke Redundanzen in der Datenstruktur vorhanden. So taucht zum Beispiel das Datenfeld *Filmtitel* in drei verschiedenen Tabellen auf. Dasselbe passiert mit allen Datenfeldern, die beide Tabellen gemeinsam haben und die die Liste beinhalten soll.

Master- und Detailtabelle





Redundanzen vermeiden



Ein besserer Lösungsansatz ist deshalb, für die zwei Objektklassen drei Tabellen, die miteinander verknüpft sind, zu verwenden. In die Mastertabelle werden alle Eigenschaften eingetragen, die beide Objektklassen gemeinsam haben. Hier finden zum Beispiel die Felder *VideoNr*, *Filmtitel* und *Länge* Platz. Die Unterscheidung der Objektklassen wird durch zwei weitere Tabellen, die Detailtabellen, realisiert. Die eine Tabelle enthält alle Eigenschaften von aufgenommenen Videos, die nicht in der Mastertabelle vorkommen. Die zweite Tabelle umfasst entsprechend die Eigenschaften, die für gekaufte Videos in der Mastertabelle fehlen.

Verknüpfen der Tabellen



Die Verknüpfung der Tabellen kann auf zwei verschiedene Arten erfolgen. Bei der ersten Möglichkeit werden in der Mastertabelle genauso viele Referenzfelder angelegt, wie Detailtabellen vorhanden sind. Die Namen der Referenzfelder entsprechen den Namen der Primärschlüssel aus den Detailtabellen. Bei der Videoverwaltung wären das die Felder *AufgenommenNr* und *KaufNr*.

Referentielle Integrität deaktivieren



Die Verknüpfungen werden durch Ziehen der Primärschlüsselfelder auf die entsprechenden Referenzfelder im *Beziehungen*-Fenster hergestellt. Dabei ist darauf zu achten, dass die referentielle Integrität *nicht* aktiviert ist. Diese Eigenschaft gilt grundsätzlich, wenn Sie mit einer Master- und mehreren Detailtabellen arbeiten. Eine Eingabe von Daten wäre damit unmöglich, weil jedem Datensatz aus der Mastertabelle ein Datensatz aus *jeder* Detailtabelle zugeordnet sein müsste. Das würde aber bedeuten, dass jeder Filmtitel aufgenommen und gekauft sein müsste bzw. eine Eintrag in beiden Tabellen hat.

Die referenzielle Integrität hat noch einen weiteren Nachteil. Es kann vorkommen, dass in ein Feld nichts eingetragen werden soll, wenn der Inhalt des Feldes nicht bekannt ist. Der zweite Hauptdarsteller kann zum Beispiel fehlen. Ist die Tabelle *Hauptdarsteller* aber referenziell mit der Mastertabelle verknüpft, muss ein Eintrag vorgenommen werden. Mit einem Trick kann man allerdings auch mit referenzieller Integrität einen Leereintrag erzeugen. In der Tabelle *Hauptdarsteller* muss in diesem Fall ein Datensatz ein Leerzeichen enthalten. Alternativ dazu kann dort auch *Kein Eintrag*, *Leer*, *Nicht Vorhanden* oder Ähnliches stehen.

Hinweis



Da auf die referenzielle Integrität verzichtet wird, ist besonders beim Löschen von Datensätzen Vorsicht geboten. Ohne referentielle Integrität kann zum Beispiel ein Film gelöscht werden, zu dem noch Daten in einer Detailtabelle vorhanden sind. Der Datensatz in der Detailtabelle wäre dann nach Löschen des Films ein verlorener Datensatz, auf den nicht mehr Bezug genommen werden kann. Um das zu verhindern, sind geeignete Löschabfragen zu entwickeln, die diese Fehler abfangen. Auch bei der Eingabe von Daten können die richtigen Abfragen als Basis für Formulare Fehler verhindern.

Verknüpfung über getrennte Referenzfelder



Sie können nun zu jeder Detailtabelle ein Formular erstellen, indem Sie die Eigenschaften aus der jeweiligen Detailtabelle mit den Daten der Mastertabelle in einer Abfrage verbinden. Beim Hinzufügen eines Datensatzes zu einem der Formulare wird im Referenzfeld für die andere Detailtabelle automatisch eine Null eingetragen. Mit referenzieller Integrität wäre das nicht möglich.



Abbildung 11.3 zeigt das *Beziehungen*-Fenster mit den Tabellen und den Verknüpfungen der Datenbank *Video1.mdb*. Zu jeder Detailtabelle existiert ein Feld in der Mastertabelle.

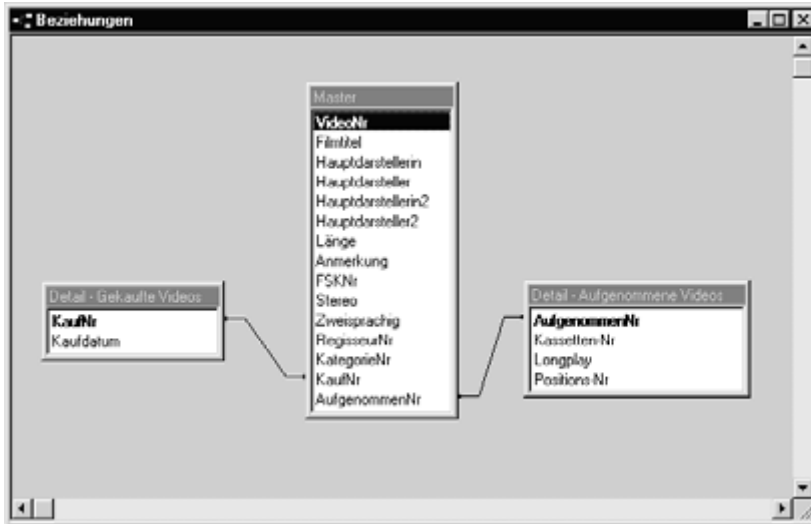


Abb. 11.3

Verknüpfung der Tabellen über Referenzfelder

Mit wachsender Anzahl von Detailtabellen wird diese Methode allerdings unübersichtlich. Da eine Verbindung zum Primärschlüssel existieren muss, wird beim Hinzufügen einer Detailtabelle die Mastertabelle verändert. Die Anzahl der Felder in der Mastertabelle erhöht sich also mit hinzukommenden Detailtabellen. Außerdem stehen in jedem Datensatz $n-1$ Nullen, wenn n die Anzahl der Detailtabellen ist. Diese Daten werden überflüssigerweise gespeichert.

Zur besseren Übersicht und kompakteren Handhabung bei gleichzeitiger Reduzierung überflüssiger Daten eignet sich für große Datenbanksysteme eine zweite Möglichkeit besser, um Master- und Detailtabellen zu verknüpfen. Anstatt in der Mastertabelle für jede Detailtabelle ein Referenzfeld anzulegen, werden nur zwei Felder benötigt.

Die Felder erhalten die Bezeichnungen *PrimärschlüsselNr* und *Typ. PrimärschlüsselNr* ist dabei vom Datentyp *LongInteger*. Mit diesem Feld werden alle Primärschlüsselfelder der Detailtabellen verbunden (wieder ohne referentielle Integrität). Das Feld *Typ* kann ein Text oder eine Zahl sein. Es kennzeichnet die mit dem Datensatz in Beziehung stehende Detailtabelle. Somit wird jedem Datensatz über den Typ genau eine Detailtabelle zugeordnet und über die eingetragene Primärschlüsselnummer auf den entsprechenden Datensatz verwiesen.

Verknüpfung über zwei Referenzfelder

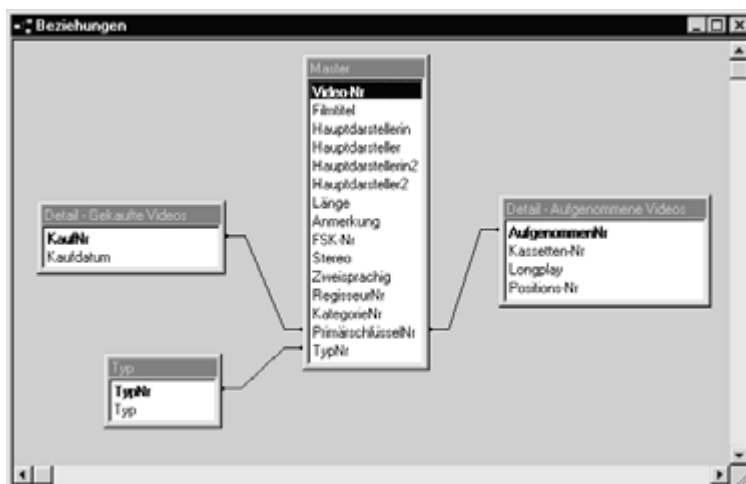


Wird für das Feld *Typ* der Datentyp *Text* verwendet, kann direkt eine Identifikationsbezeichnung für die jeweils benutzte Detailtabelle eingetragen werden. Im vorliegenden Fall wären das zum Beispiel *Gekauft* und *Aufgenommen*. Besser ist es aber, Zahlen, zum Beispiel 1 und 2, zu verwenden. Das Feld *Typ* sollte dann eher *TypNr* heißen. Passend dazu wird eine Typentabelle erstellt, in der die Identifikationsbezeichnungen gespeichert werden.

Um jetzt eine neue Detailtabelle hinzuzufügen, brauchen Sie nur noch einen Identifikationseintrag in der Typentabelle zu machen und die Tabelle mit der Master-tabelle über das Feld *PrimärschlüsselNr* zu verbinden. In *Abbildung 11.4* sind die Beziehungen der Tabellen dargestellt. Sie finden die Tabellenstruktur in der Datenbank *Video2.mdb*.

Abb. 11.4

Verknüpfung



Wenn wie bei der Videodatenbank nur zwei Detailtabellen verwendet werden, lässt sich die Typentabelle auch durch ein *Ja/Nein*-Feld ersetzen. Es könnte zum Beispiel *Gekauft* heißen. Hat das Feld den Wert *Ja*, wird die Detailtabelle für gekaufte Videos benutzt. Lautet der Eintrag *Nein*, wird die Tabelle für aufgenommene Videos verwendet.

Egal, welche Form des Mehrstabellensystems verwendet wird, es treten keine Redundanzen mehr in der Datenstruktur auf. Auch das Suchen nach Datensätzen, zum Beispiel nach einem Titel, ist kein Problem mehr, weil alle Filme in einer Tabelle gespeichert sind.

Hinweis



Bei den oben beschriebenen Strukturen ist unbedingt die Reihenfolge der Eingabe in Formularen zu beachten. Es muss immer zuerst ein neuer Datensatz in der Mastertabelle angelegt werden, bevor in den Feldern der Detailtabelle eine Eingabe erfolgt. Wird zum Beispiel erst die Kassettennummer eingegeben und dann der Name des Filmtitels, tritt ein Fehler auf.



Spezielle Beziehungstypen

Access kennt die zwei Beziehungen 1:1 und 1:n. Sie werden im *Beziehungen*-Fenster einfach mit den Referenzfeldern zweier Tabellen definiert. Es lassen sich aber auch noch andere, komplexere Beziehungen erzeugen, die auf den ersten Blick nicht offensichtlich sind.

Von diesen ist die einfachste und am häufigsten benötigte die n:m-Beziehung. Auch in der Beispieldatenbank *Kapitel11.mdb* kommt diese Beziehung vor. Sie besteht zwischen den Tabellen *Lieferanten* und *Artikel*. Jeder Lieferant kann mehrere Artikel liefern und jeder Artikel kann bei mehreren Lieferanten geordert werden.

Um die Struktur einer n:m-Beziehung zwischen zwei Tabellen zu erzeugen, benötigt man eine Zwischentabelle. In dieser Tabelle stehen die Referenzfelder zu den Primärschlüsseln. Im Beispiel trägt die Zwischentabelle den Namen *Lieferanten und Artikel*. Dieser eindeutige Name basiert auf der Tatsache, dass in ihr jede mögliche Kombination aus Lieferanten und Artikeln abgelegt werden kann. Zu jeder Lieferantennummer können beliebig viele Artikel gespeichert werden und umgekehrt. In *Abbildung 11.5* ist das Verknüpfungsschema dargestellt.

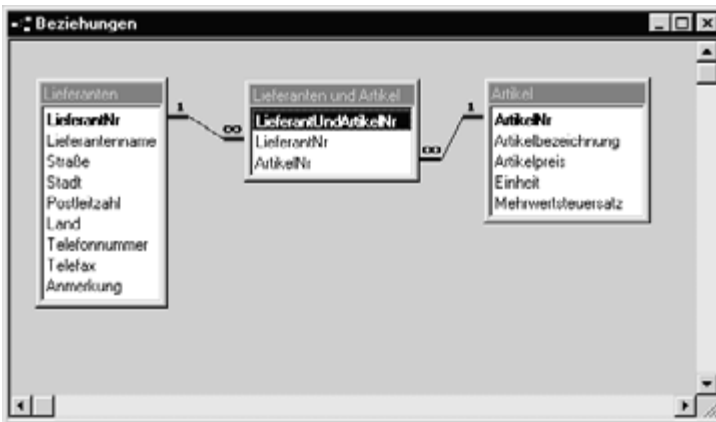


Abb. 11.5

Die Struktur einer n:m-Beziehung im *Beziehungen*-Fenster

Ein weiterer Vorteil der Zwischentabelle ist, dass in ihr noch zusätzlich Daten gespeichert werden können, die sich für jeden Artikel von Lieferant zu Lieferant ändern. Es ist zum Beispiel sinnvoll, den Artikelpreis mit in die Zwischentabelle aufzunehmen, da ein und derselbe Artikel bei jedem Lieferanten einen anderen Preis haben kann. Zusätzlich könnte ein Rabattpfeld angelegt werden, das jedem Pärchen Lieferant/Artikel eine Rabattstaffelungstabelle zuordnet. Die Datenstruktur lässt sich mit diesem Schema auf vielfältige und individuelle Art gestalten.

Eine Besonderheit ist die Realisierung einer Beziehung, die eine Tabelle mit sich selbst verknüpft. Eine solche Verknüpfung kann zum Beispiel dazu genutzt werden, in einer Personaltabelle einer Person die ihr übergeordnete Person zuzuordnen.

Speichern zusätzlicher Daten

Tabelle mit sich selbst verknüpfen



1:n-Beziehung auf sich selbst

Der einfachere Fall ist auch hier die 1:n-Beziehung: Jeder Person (1) können mehrere Personen (n) untergeordnet sein. Dazu wird in der Tabelle *Personen* ein Feld mit dem Namen *ÜbergeordnetePersonNr* angelegt, in dem zu jeder Person die entsprechende übergeordnete Person eingetragen werden kann.

Die Verbindung kann im *Beziehungen*-Fenster definiert werden. Fügen Sie einfach die Tabelle *Personen* zweimal hinzu. Access benennt die zweite Tabelle automatisch *Personen_1*. Ziehen Sie das Feld *ÜbergeordnetePersonNr* aus einer der beiden Tabellen auf das Feld *PersonNr* der anderen Tabelle und definieren Sie die Beziehung. Dabei darf keine referentielle Integrität verwendet werden. Um nämlich dann den ersten Datensatz zu erstellen, müsste schon ein Datensatz vorhanden sein. Ein unmögliches Unterfangen. Die Tabellen und Beziehung finden Sie in der Datenbank *1 zu n auf sich selbst.mdb*.

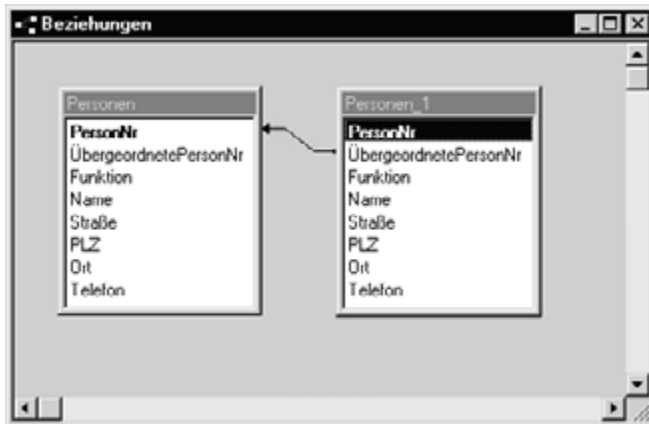


Abb. 11.6

Das *Beziehungen*-Fenster

n:m-Beziehung auf sich selbst

Die komplexeste Verbindung ist eine n:m-Beziehung einer Tabelle auf sich selbst. Im Beispiel der Personentabelle wird dann auch berücksichtigt, dass eine Person sowohl mehrere untergeordnete als auch mehrere übergeordnete Personen haben kann. Erstellt wird die Beziehung über eine Zusatztable, die die Felder *PersonNr* und *ÜbergeordnetePersonNr* beinhaltet. Jeder Person können über die Zwischentabelle mehrere übergeordnete Personen zugewiesen werden, deren Namen wieder aus der Personentabelle geholt werden. Die Beziehung finden Sie in der Datenbank *n zu m auf sich selbst.mdb*.



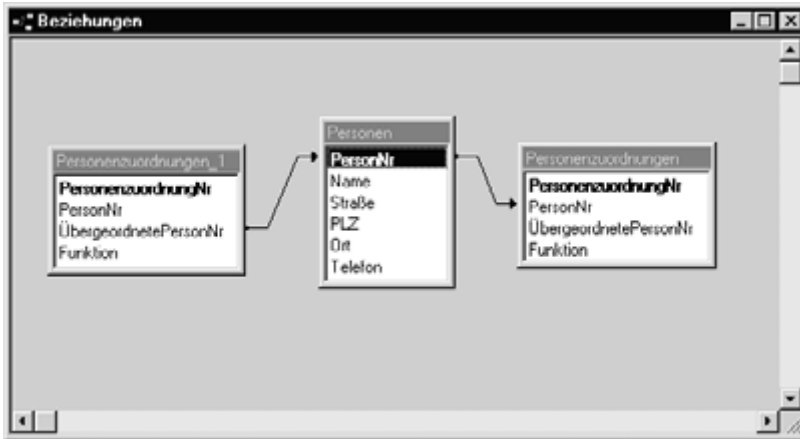


Abb. 11.7

n:m-Verknüpfung einer Tabelle auf sich selbst

Auch hier können in der Zusatztable wie bei der normalen n:m-Verknüpfung weitere Felder auftreten, die für jede Kombination aus *Person* und *Übergeordneter Person* unterschiedliche Werte annehmen. Über das Feld *Funktion* kann zum Beispiel jeder übergeordneten Person ein Aufgabenbereich zugewiesen werden, der für verschiedene untergeordnete Personen variieren kann. Für die Person *Meier* ist die Person *Müller* beispielsweise Abteilungsleiter und für die Person *Schmitz* ist *Müller* Projektleiter.

In einem Formular lassen sich so mit der Methode der n:m-Beziehung einer Tabelle auf sich selbst zu jeder Person in zwei Unterformularen die unter- und übergeordneten Personen auflisten.

Wichtig bei allen Beziehungen ist, dass die Referenzfelder zu den Primärschlüsseln indiziert werden. Dies ist vor allen Dingen bei großen Datenmengen zu beachten, da ansonsten die Performance stark zurückgehen kann. Die Verarbeitung der Daten wird unnötigerweise verlangsamt. Außerdem müssen die Referenzfelder immer vom Datentyp *LongInteger* sein, was dem Datentyp des *AutoWert*-Feldes entspricht.

Referenzfelder indizieren

Gesamtstruktur

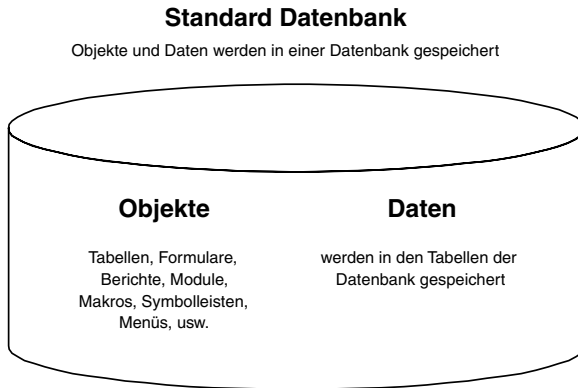
Bei der Erstellung eines Datenbanksystems sind nicht nur Überlegungen zur Feinstruktur nötig, sondern auch zur Grobstruktur, d.h. zur Datenbankverwaltung. Normalerweise werden unter Access alle Objekte (Formulare, Berichte, Makros, Module, Symbolleisten etc.) und die Daten in *einer* Datenbank gespeichert. Bei kleineren Projekten oder privaten Datenbanken ist dies auch sicherlich ausreichend.

Standarddatenbankmodell



Abb. 11.8

Daten und Objekte in einer Datenbank vereint

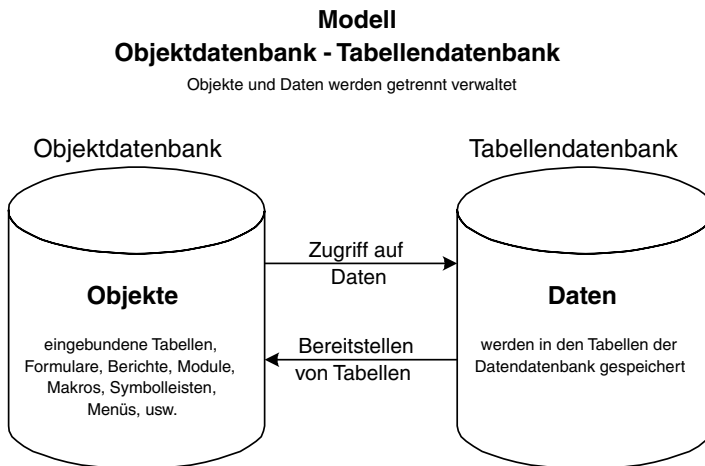


*Objektdatenbank –
Tabellendatenbank-
Modell*

Diese Methode birgt aber vor allem, wenn die Projekte größer werden, einige Nachteile. So lässt sich eine laufende Datenbank beispielsweise nicht aus einem Formular heraus komprimieren. Bessere Möglichkeiten bietet dagegen die Aufteilung von Objekten und Daten in verschiedene Datenbanken. Die Basis dazu bildet das *Objektdatenbank – Tabellendatenbank-Modell*.

Abb. 11.9

Schema für das Modell
*Objektdatenbank –
Tabellendatenbank*



Bei diesem Modell werden zunächst alle Tabellen, die für die Datenbank benötigt werden, in der Datendatenbank abgelegt und die Beziehungen definiert. Diese Tabellen werden dann in die Objektdatenbank eingebunden, wobei die vorher definierten Beziehungen automatisch übernommen werden. Die Verbindung zwischen Objektdatenbank und Datendatenbank entsteht also über die eingebundenen Tabellen. Im nächsten Schritt werden alle Objekte, d.h. Formulare, Berichte usw., die für die Datenbank benötigt werden, erstellt und in der Objektdatenbank gespeichert.



Mit der Aufteilung der Datenbank ist es nun zum Beispiel möglich, mittels eines Formulars die laufende Datenbank zu verwalten, sprich zu komprimieren oder zu reparieren. Das ist deshalb möglich, weil sich diese Operationen auf die Tabellendatenbank beziehen, das Formular, mit dem die Operationen ausgeführt werden, sich aber in der Objektdatenbank befindet. Die Methoden zum Aufrufen solcher Verwaltungsfunktionen werden im Abschnitt *Datenbanken verwalten* weiter unten in diesem Kapitel beschrieben.

Die eingebundenen Tabellen dürfen während des Komprimierens oder Reparierens weder über das Datenbankfenster noch über ein *Recordset*-Objekt geöffnet sein.

Das Trennen von Daten und Objekten bringt noch einen weiteren Vorteil mit sich: Es lassen sich so verschiedene Objektdatenbanken anlegen, die alle auf den gleichen Datenstamm zurückgreifen, aber unterschiedliche Aufgaben bewältigen. In jede Objektdatenbank werden nur die Tabellen eingebunden, die für die Arbeit und den Zugriff auf die Daten erforderlich sind. Die Tabellendatenbank bleibt dabei ein Ganzes.

Datenbank während der Laufzeit verwalten



Hinweis

Modell Objektdatenbank - Tabellendatenbank

Zugriff auf Daten von mehreren Objekten

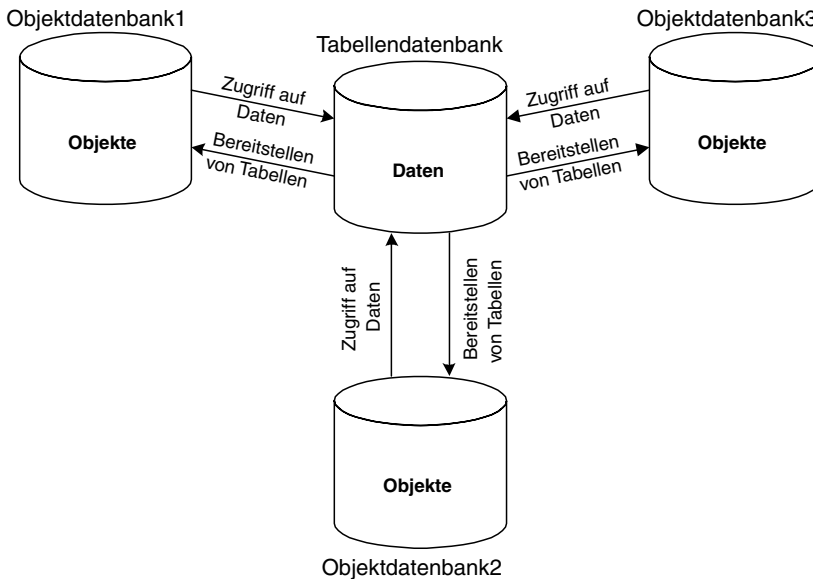


Abb. 11.10

Zugriff auf Daten von verschiedenen Objektdatenbanken

Dieses Prinzip eignet sich ebenfalls sehr gut für die Implementierung in einem Netzwerk. Dabei kann die Datendatenbank auf dem Server installiert werden und die Objektdatenbanken auf den Arbeitsplatzrechnern.





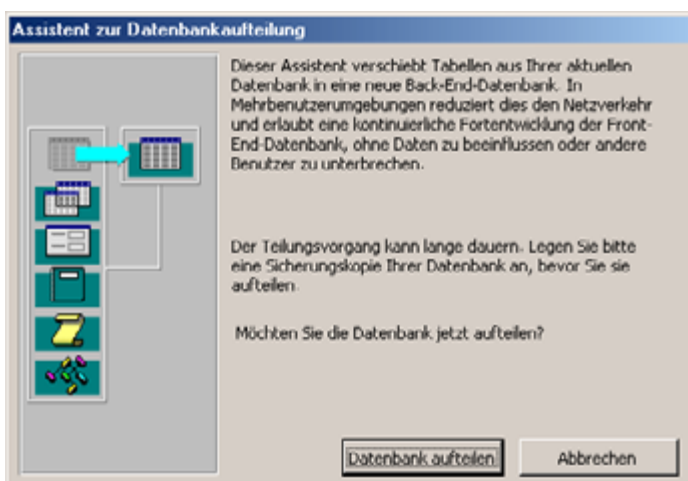
Mit der Aufteilung von Datenbanken wird das konzeptionelle Datenbankschema und damit auch das Entity-Relationship-Diagramm sehr komplex. Folglich sollte man ganz besonders bei diesem Modell viel Zeit in die Planung und den Entwurf der Struktur stecken. Ein ausgereiftes Konzept ist der Schlüssel zum Erfolg.

Assistent zur Datenbankaufteilung

Das Aufteilen einer bereits bestehenden Datenbank kann mit Access automatisch durch einen Assistenten erfolgen. Wählen Sie dazu den Menübefehl *Extras > Datenbank-Dienstprogramme > Assistent zur Datenbankaufteilung*.

Abb. 11.11 

Der Assistent zur Datenbankaufteilung



Wenn Sie die Datenbank aufteilen lassen, können Sie im folgenden Dialog den Namen der Back-End-Datenbank angeben, d. h. der Datenbank, die nur die Tabellen beinhalten soll. Der Pfad, in dem die Back-End-Datenbank abgelegt wird, kann ein beliebiger sein. Der Assistent exportiert anschließend alle vorhandenen Tabellen in die Back-End-Datenbank, löscht die Tabellen in der aktuellen Datenbank und bindet die Tabellen aus der Back-End-Datenbank in die aktuelle Datenbank ein. Nach Beendigung des Assistenten können Sie direkt mit der Datenbank weiterarbeiten.

Replikation von Datenbanken

Mit Access ist es möglich, Datenbanken zu replizieren. Dabei werden von einer Originaldatenbank (*Design-Master-Replikate*) Kopien (*Replikate*) erzeugt, die mit dem Original identisch sind. Diese Kopien können dann auf einem oder mehreren Computern von verschiedenen Benutzern verwendet werden. Das Replizieren einer Datenbank geht allerdings weit über das einfache Kopieren hinaus. Die Aufgabe der Replikation ist, die Daten aller Replikate einer Replikatgruppe zu synchronisieren.



Zum Beispiel könnte die Design-Master-Datenbank in der Zentrale einer Firma gespeichert sein. Jeder Außendienstmitarbeiter erhält dann ein Replikat, mit dem er arbeiten und an dem er Datensätze verändern kann. Die geänderten Daten können durch Synchronisation mit der Design-Master-Datenbank dieser übergeben werden.

Voraussetzung für die Nutzung der Datenbankreplikation ist, dass Sie bei der Installation von Windows den Aktenkoffer und bei der Installation von Access die Aktenkoffer-Replikation mit installiert haben. Die Funktionen sind sonst nicht verfügbar.



Hinweis

Anlegen einer Replikatgruppe

Wenn zum ersten Mal ein Replikat von einer Datenbank erzeugt wird, legt Access zunächst eine Replikatgruppe an, die die Originaldatenbank und eine mit dem Original identische Kopie enthält. Nachdem die Replikatgruppe erstellt wurde, können beliebig viele weitere Replikate vom Original oder von Replikaten erzeugt und damit der Gruppe hinzugefügt werden. Alle Datenbanken, die in diesem Replicationset enthalten sind, können untereinander synchronisiert werden. Es ist jedoch nicht möglich, Replikate aus verschiedenen Replikatgruppen zu synchronisieren.

Um Objekte, die auf Replikation basieren, eindeutig identifizieren zu können, gibt es eine Replikationskennung. Diese Kennung ist ein 16 Byte großes Feld, in dem eine **GUID** (**G**lobal **U**nique **I**dentification) abgelegt wird. Aufgrund der Größe des Feldes und der zufälligen Erzeugung der GUIDs ist es nahezu ausgeschlossen, dass irgendwann doppelte Kennungen vorkommen. Den folgenden Objekten wird bei der Replikation eine GUID zugewiesen:



Identifikation von Objekten

- Replikatgruppen
- Design-Master-Replikaten (*DesignMasterID*)
- Replikaten (*ReplicaID*)
- Tabellen
- Datensätzen
- Objekten (Abfragen, Formulare, Berichte, Makros und Module)

Das **Design-Master-Replikat** (*DMR*) erhält einige neue Eigenschaften, Felder und Tabellen, die für die Replikation der Datenbank benötigt werden.

Zu den neuen Eigenschaften gehört die bereits erwähnte Replikationskennung (*DesignMasterID*). Des Weiteren werden den Tabellen und Objekten der Datenbank die Eigenschaften *Replicable* und *KeepLocal* hinzugefügt. Mit der Eigenschaft *Replicable* wird für jedes Objekt eingestellt, ob es replizierbar sein soll oder nicht. Nur replizierbare Objekte können beim Erstellen eines Replikats an dieses weitergegeben wer-



Neue Eigenschaften





den. Über die Eigenschaft *KeepLocal* kann vor der Replikation entschieden werden, ob das jeweilige replizierbare Objekt an das Replikat weitergegeben wird. Für ein bereits repliziertes Objekt kann diese Eigenschaft nicht mehr verändert werden.

Neue Felder und Tabellen

Den Tabellen des DMR werden drei neue Felder hinzugefügt, die normalerweise nicht sichtbar sind. Über den Menüpunkt *Extras > Optionen* können im Register *Ansicht* die Systemobjekte eingeblendet werden. Diese Felder mit den Namen *s_Guid*, *s_lineage* und *s_generation* sind für den Änderungsstatus und die Identifikation eines Datensatzes zuständig. Die Inhalte sind schreibgeschützt und können somit nicht verändert werden. Die Datenbank beinhaltet auch einige neue Systemtabellen, die ebenfalls für die Organisation der Replikation benötigt werden.

Hinweis



Sie können einer Datenbank über den Menübefehl *Extras > Sicherheit > Datenbankkennwort festlegen* ein Kennwort zuweisen, das bei jedem Start der Datenbank abgefragt wird. Wenn Sie eine Datenbank replizieren wollen, darf diese nicht durch ein Kennwort geschützt sein, die Replikation ist sonst nicht möglich. Bei einer bereits replizierten Datenbank ist der Menüpunkt deaktiviert.

Entwurf von Objekten

Objekte im DMR

Alle Objekte in der Design-Master-Datenbank können wie gewohnt angelegt werden. Anschließend kann für jedes Objekt entschieden werden, ob es replizierbar sein soll. Dazu ist das Objekt im Datenbankfenster zu markieren und im Dialog *Eigenschaften* das Kontrollkästchen *Replizierbar* zu aktivieren. Die Veränderung der Eigenschaften mithilfe von VBA wird in einem noch folgenden Abschnitt erläutert. Wird nun von der Datenbank ein Replikat erzeugt oder ein bestehendes Replikat synchronisiert, dann werden alle als repliziert gekennzeichneten Objekte dem Replikat übergeben. Ausgenommen sind Objekte, bei denen die *KeepLocal*-Eigenschaft aktiviert ist.

Objekte in einem Replikat

Objekte, die in einem Replikat erzeugt werden, sind nur in der Replikatdatenbank sichtbar. Sie können nicht repliziert werden, d. h., sie werden beim Synchronisieren wie DMR-Objekte behandelt, bei denen die *Replizierbar*-Eigenschaft deaktiviert ist. Die Eigenschaft *Replizierbar* kann in Replikatdatenbanken für Objekte auch nicht eingestellt werden.

Hinweis



Standardmäßig wird die Originaldatenbank als DMR verwendet. An dieser Datenbank werden alle Veränderungen vorgenommen, die beim Synchronisieren an die Replikate übergeben werden. Es kann allerdings zu jeder Zeit jedes Replikat einer Gruppe zum DMR gemacht werden. Das muss zum Beispiel geschehen, wenn das DMR zerstört wurde.

Synchronisieren von Replikaten

Das Synchronisieren von zwei Datenbanken bewirkt, dass sowohl die Objekte als auch die Datensätze der Datenbanken aktualisiert werden. Objekte können allerdings nur in eine Richtung synchronisiert werden, nämlich von dem DMR zu einem Replikat. Zusätzliche Objekte in einem Replikat werden aus im vorangegangenen Abschnitt genannten Gründen nicht an das DMR übergeben. Für die Synchronisa-



tion von Datensätzen können vier verschiedene Einstellungen getroffen werden. Die Standardeinstellung beim Erzeugen eines Replikats ist die bidirektionale Synchronisation. Dabei wird ein Austausch in beide Richtungen zugelassen. Veränderte oder neue Datensätze aus dem Replikat werden in dem DMR ergänzt und umgekehrt. Am Beispiel der Firma mit Außendienstmitarbeitern würde bei dieser Einstellung jeder Mitarbeiter alle Daten erhalten.

Wenn das nicht erwünscht ist, kann mit der VBA-Methode *Synchronize* des DAO-Objekts *Database* auch ein unidirektionaler Austausch definiert werden. In diesem Fall kann der Mitarbeiter entweder nur neue Daten empfangen (*Import*), die eigenen Veränderungen aber nicht zurückgeben, oder nur Daten an das DMR übergeben (*Export*), wobei die Datensätze der anderen Mitarbeiter ihm allerdings verschlossen bleiben.

Die vierte Einstellung wird getroffen, wenn zwei Datenbanken über das Internet synchronisiert werden sollen, wobei diese Einstellung zusammen mit einer der oben genannten Einstellungen benutzt wird.

Möglicher Datenaustausch aus Sicht des Replikats

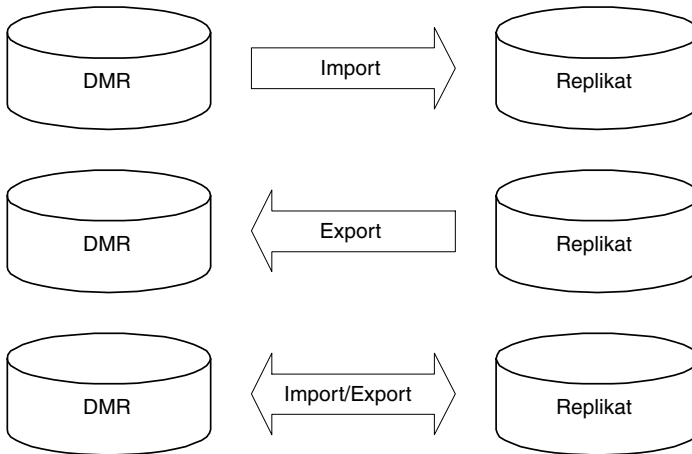


Abb. 11.12

Synchronisieren von
Replikat-Datenbanken

Die Synchronisierungseinstellung wird immer für die gesamte Datenbank definiert. Es ist nicht möglich, einzelnen Objekten bestimmte Synchronisierungsrichtungen zuzuweisen. Allerdings kann mit der *Synchronize*-Methode die Datenflussrichtung bei jedem Synchronisieren neu bestimmt werden.



Hinweis

Mit Access XP ist es möglich, für jede Tabelle einen Filter zu definieren, der die für die Replizierung relevanten Datensätze auswählt. Dazu müssen Sie dem betreffenden *TableDef*-Objekten die Eigenschaft *ReplicaFilter* hinzufügen. Hier legen Sie dann den *WHERE*-Teil einer SQL-Abfrage als Bedingung ab, die die gewünschten Datensätze auswählt. Wenn Sie den Replikationsfilter zwischen mehreren Replikations-

Teilreplikation





schritten verändern, können in der Tabelle so genannte *verwaiste Datensätze* auftauchen, die nicht mehr an der Replikation teilnehmen und daher veraltet sind. Diese Datensätze entfernen Sie durch Einsatz der *PopulatePartial*-Methode.

Mit Access ist es nicht möglich, unterschiedliche Synchronisierungsmethoden für die verschiedenen Datenbankobjekte einzusetzen. Das heißt, dass zum Beispiel die Tabelle *Personal* nicht nur für den Import bestimmt sein kann, die Tabelle *Kunden* aber in beide Richtungen synchronisiert werden kann. Die Richtung wird grundsätzlich für die gesamte Datenbank definiert. Es muss also vorher entschieden werden, ob ein Mitarbeiter nur lesen oder nur schreiben kann oder ob er den vollen Zugriff auf alle Daten bekommt. Mit einem Trick ist es allerdings möglich, diese Beschränkung zu umgehen, so dass eine partielle Replikation simuliert werden kann.

Zu diesem Zweck muss die Design-Master-Datenbank in drei Datenbanken aufgeteilt werden. Eine Datenbank erhält die Synchronisationseigenschaft *Import*, eine die Eigenschaft *Export* und die dritte *Import/Export*. In der Datenbank für Import und Export werden alle Datenbankobjekte, also Abfragen, Formulare usw., gespeichert und die Tabellen, in denen Daten stehen, die für jeden zugänglich sein sollen. Außerdem werden die Tabellen aus den beiden anderen Datenbanken für den unidirektionalen Austausch in diese Datenbank eingebunden. Auf diese Weise kann mit der Import/Export-Datenbank gearbeitet werden.

Sicherheitskopien erzeugen

Beim Anlegen von Sicherheitskopien einer Datenbank in einer Replikatgruppe muss ebenfalls auf die Methode der Replikation zurückgegriffen werden. Die Datenbank nur zu kopieren, reicht nicht aus, weil dadurch der Bezug zu der Ursprungsdatenbank verloren geht. Werden Datensätze in dem benutzten DMR durch Neueingabe oder Synchronisation verändert, stimmen die Daten nicht mehr mit denen in der kopierten Datenbank überein. Bei Beschädigung des Design-Master-Replikats muss mit der Kopie weitergearbeitet werden. Dies ist aber nicht möglich, weil keine Synchronisation zwischen der Kopie und dem Original stattgefunden hat. Der Versuch, die Kopie mit anderen Mitgliedern der Replikatgruppe zu synchronisieren, schlägt deshalb fehl. Eine Sicherheitskopie muss also immer durch Replikation erzeugt werden. Dadurch sind bei ständiger Aktualisierung des Backups die Daten jederzeit auf dem neuesten Stand, so dass mit dem Backup im Notfall sofort weitergearbeitet werden kann.

Replikation mit VBA

Sie können die Replikation auch mit VBA durchführen. Dafür werden zwei Methoden verwendet; eine zum Erzeugen neuer Replikate und eine zum Synchronisieren. Beide Methoden werden ohne Zwischenabfragen oder Meldungsfenster ausgeführt, so dass die Replikation unauffällig für den Benutzer abläuft.





Mithilfe der Methode *Makereplica* können Sie ein neues Replikat in einer Replikatgruppe erzeugen. Sie können die Funktion aber auch zum Anlegen eines Backups in einer Replikatgruppe nutzen. Wichtig ist, dass die Kombination aus Pfad und Name des Replikats noch nicht existiert, da es sonst zu einem Fehler kommt. Die *Makereplica*-Methode hat die folgende Syntax:

```
<Datenbank>.MakeReplica <PathName>, <Description> _  
[, <Optionen>]
```

Datenbank ist der Name eines vorhandenen replizierbaren *Database*-Objekts. Dabei kann es sich um die aktuelle oder eine externe Datenbank der Replikatgruppe handeln. Der Bezeichner *PathName* ist eine Zeichenkette und gibt den Namen und Pfad des neuen Replikats an. Mit dem Bezeichner *Description* muss dem Replikat eine Beschreibung gegeben werden. Die optionalen Konstanten *dbRepMakeReadOnly* und *dbRepMakePartial* können Sie ggf. auch angeben, um Änderungen am neuen Replikat zu verhindern bzw. ein Teilreplikat zu erstellen.

Als Beispiel für die Verwendung der Methode dient eine Prozedur, die ein Backup der aktuellen Datenbank erstellt:

```
Sub Backup()  
  
    Dim DB As Database  
  
    On Error GoTo Fehler  
  
    DoCmd.Hourglass True  
  
    Set DB = DBEngine(0)(0)  
  
    DB.MakeReplica "C:\backup.mdb", "Backup des  
Replikats", _  
  
        dbRepMakeReadOnly  
  
Fehler:  
    DoCmd.Hourglass False  
    Exit Sub  
  
End Sub
```

Da die Replikation einer Datenbank unter Umständen lange dauern kann, wird vor dem Vorgang die Sanduhr eingeschaltet.

Das nächste Beispiel erzeugt aus der aktuellen Datenbank heraus ein Replikat des Backups. Dazu muss lediglich dem *Database*-Objekt die gewünschte Datenbank zugewiesen werden.

Makereplica-
Methode



```

Sub Backup_vom_Backup()

    Dim DB As Database
    On Error GoTo Fehler

    DoCmd.Hourglass True

    Set DB = DBEngine(0).OpenDatabase("C:\backup.mdb")

    db.MakeReplica ("C:\backup2.mdb", "Backup vom
Backup", _
    dbRepMakeReadOnly

Fehler:
    DoCmd.Hourglass False
    Exit Sub

End Sub

```

Synchronize-Methode

Wenn Datensätze in einem oder beiden Replikaten verändert wurden oder wenn in der Design-Master-Datenbank neue Objekte hinzugefügt wurden, die das Replikat erhalten soll, können die Datenbanken mit der *Synchronize*-Methode auf den neuesten Stand gebracht werden. Über eine Konstante kann die schon erwähnte Datenflussrichtung festgelegt werden.

Die *Synchronize*-Methode hat die folgende Syntax:

```
<Datenbank>.Synchronize <PathName> [, <Austausch>]
```

<Datenbank> muss wieder ein replizierbares Replikat der Replikatgruppe sein. Das Gleiche gilt für den Bezeichner <PathName>. Der Bezeichner <Austausch> ist optional und kann eine der folgenden vier Konstanten sein:

Konstanten für die Datenflussrichtung

Konstante	Beschreibung
<i>dbRepExportChanges</i>	Es werden nur die Daten der aktuellen Datenbank in die in <i>Pfad</i> angegebene Datenbank weitergegeben.
<i>dbReplImportChanges</i>	Es werden nur die Daten von der in <i>Pfad</i> angegebenen Datenbank an die aktuelle Datenbank weitergegeben.
<i>dbReplmExpChanges</i>	Der Datenaustausch findet in beide Richtungen statt.



Konstanten für die Datenflussrichtung (Forts.)

Konstante	Beschreibung
<i>dbRepSyncInternet</i>	Der Datenaustausch findet zwischen der aktuellen Datenbank und einer Datenbank in der in <i>Pfad</i> angegebenen Internetadresse statt. Diese Option wird in Zusammenarbeit mit den drei oben genannten Optionen eingesetzt.

Tabelle 11.5

Mit der folgenden Prozedur wird ein Backup-Replikat mit der aktuellen Datenbank synchronisiert. Da keine Konstante angegeben ist, wird die Voreinstellung *Import/Export*, also der Datenaustausch in beide Richtungen benutzt.

```
Sub Synchro()  
  
    Dim DB As DATABASE  
  
    Set DB = DBEngine(0)(0)  
  
On Error GoTo Fehler  
  
    DoCmd.Hourglass True  
  
    DB.Synchronize ("C:\backup.mdb")  
  
    DB.Close  
  
Fehler:  
    DoCmd.Hourglass False  
    Exit Sub  
  
End Sub
```

Bei der Arbeit mit Replikaten können selbstverständlich Konflikte auftreten. Wenn ein und derselbe Datensatz in zwei verschiedenen Replikaten bearbeitet und verändert wird, tritt beim Synchronisieren der Datenbanken ein Konflikt auf, d.h. in einem Replikat können die gemachten Änderungen nicht übernommen werden.

Auftretende
Konflikte

Einerseits könnten widersprüchliche Daten eingegeben worden sein, d.h. ein Benutzer hat den Namen in Meier geändert und der andere in Müller. Da ein Konflikt auf Datensatzebene auftritt, kann andererseits auch bei keiner widersprüchlichen Eingabe das Anlegen eines Datensatzes verweigert werden, also wenn ein Benutzer Änderungen am Namen vornimmt und der andere an der Telefonnummer.





Der nicht übernommene Datensatz wird in einer Konflikttabelle gespeichert, deren Namen Sie über die *ConflictTable*-Eigenschaft abfragen können. Auch das Abfangen und Lösen von Konflikten per Funktion ist möglich. Für den Aufruf der Funktion ist die *ReplicationConflictFunction*-Eigenschaft zuständig. Mit deren Hilfe können Sie als Programmierer entscheiden, was bei einem eintretenden Konflikt passieren soll.

Formulare und Berichte

Formulare und Berichte sind die für die Kommunikation mit dem Anwender wichtigsten Objekte in einer Datenbank unter Access. Sie ermöglichen die Eingabe und Ausgabe von Daten. Beim Design von Formularen und Berichten kommt es zum einen auf Übersichtlichkeit und einfache, für jedermann durchschaubare Bedienbarkeit an, zum anderen sind individuelle Designfragen von Bedeutung. Darunter fallen zum Beispiel Firmenlogos oder interne Designrichtlinien, wobei vor allem Einheitlichkeit zur Übersicht beiträgt.

Planen der Formulare und Berichte

Bevor mit dem Erstellen von Formularen und Berichten unter Access begonnen wird, müssen die entsprechenden Abfragen entworfen werden. Um diese korrekt definieren zu können, muss die Struktur der Formulare und Berichte aber schon bekannt sein. Zu diesem Zweck ist am Anfang ein Entwurf auf Papier zu machen.

Formulare werden einerseits für die Eingabe von Daten benötigt, andererseits bieten sie die Möglichkeit, Daten zu präsentieren, sei es in Form von Listen, Grafiken oder Berechnungen. Sie dienen zum Verwalten, Suchen und Selektieren von Daten. Bei der Entwicklung von Formularen sind alle diese Eigenschaften zu berücksichtigen. Für jede Aufgabe ist das entsprechende Formular handschriftlich zu entwerfen. Das Formular kann dabei Unterformulare, Grafiken, Steuerschaltflächen oder andere beliebige Objekte enthalten.

Richtlinien zum Formulardesign

- Der wichtigste Grundsatz bei der Planung des Entwurfs ist ein einheitliches Design für alle Formulare. Dieses Grunddesign garantiert ein leichtes Arbeiten mit der fertigen Datenbankanwendung und eine kurze Einarbeitungszeit in selbige. Einheitliches Design bedeutet im Einzelnen:
 - Für alle Formulare besteht die gleiche Farbkonvention, d.h. Gleichheit in Bezug auf Hintergrundfarbe, Steuerelementfarbe, Textfarbe etc.
 - Einheitliche Anordnung von Steuerelementen, zum Beispiel Platzierung von Befehlsschaltflächen.
 - Einheitliche Schriftart und Schriftgröße für äquivalente Formularobjekte.
 - Einheitliche Verwendung von Spezialeffekten (*Vertieft*, *Erhöht*, *Normal*) für sich entsprechende Steuerelemente.



Nachdem das Grunddesign festgelegt ist, steht die Überlegung an, welche Elemente in den einzelnen Formularen untergebracht werden sollen. Diese Überlegung ist anwendungsspezifisch und auf das vorliegende Problem abzustimmen.

Für den handschriftlichen Entwurf von Berichten gilt im Grunde genommen die gleiche Vorgehensweise wie für Formulare. Es wird allerdings nicht explizit ein Grunddesign benötigt, abgesehen von Firmenlogos oder Briefköpfen, die in allen Berichten erscheinen sollen. Hier richtet sich der Entwurf nach schon vorhandenen Richtlinien oder vorgegebenen Geschäftsformularen und kann grundsätzlich für jeden Bericht verschieden sein. Ein Zeitnachteil in Bezug auf Einarbeitung und Handhabung entsteht nicht, da mit den Berichten in der Datenbank nicht direkt gearbeitet wird. Übersichtlichkeit, besonders leichte Lesbarkeit, schadet aber auch bei Berichten nicht.

○
Richtlinien zum
Berichtdesign

Erstellen der entsprechenden Abfragen

Wenn die handschriftlichen Entwürfe für alle Formulare und Berichte mit allen verwendeten Objekten vorliegen, folgt die Definition der Abfragen. Mithilfe von Abfragen können Tabellen zusammengeführt werden, wenn ein Formular oder ein Bericht auf Datensätze mehrerer Tabellen zugreift. Abfragen erlauben ebenso das Suchen nach Daten und die Selektion.

Eine besondere Bedeutung kommt den Löschabfragen zu. Diese sind auf jeden Fall zu verwenden, wenn bei Verknüpfungen keine referentielle Integrität verwendet wurde. Sie vermeiden das ungewollte Löschen von Daten und damit die Entstehung von verlorenen Datensätzen. Mit der richtigen Löschabfrage lässt sich das Löschergebnis vor dem unwiderruflichen Entfernen der Datensätze betrachten, so dass Fehler vermieden werden.

○
Löschabfragen

Verwendet man SQL-Abfragen, können sogar neue Tabellen erstellt und Primärschlüssel und Indizes definiert werden, und auch der Zugriff auf einen SQL-Server ist möglich.

Anlegen der Formulare und Berichte

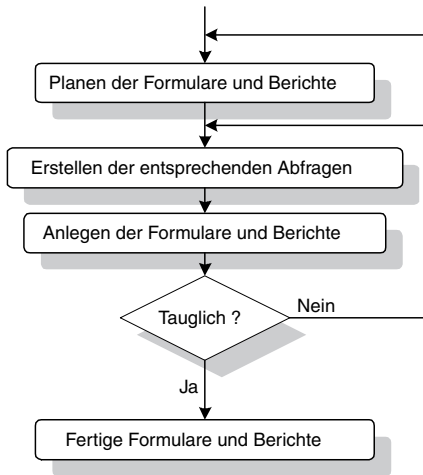
Nachdem alle Abfragen definiert sind, können die darauf basierenden Formulare und Berichte auf Basis des handschriftlichen Entwurfs in Access angelegt werden. Um die Objekte der Formulare und Berichte zu erstellen, bietet Access ausreichend Assistenten und Funktionen.

Die angelegten Formulare sind anschließend auf Tauglichkeit für den alltäglichen Betrieb zu prüfen. Ist das Ergebnis nicht zufrieden stellend, müssen die Abfragen überarbeitet werden. Wenn das nicht reicht, geht man noch einen Schritt weiter zurück, um an den Entwürfen der Formulare und Berichte zu feilen. Der Prozess wird so lange wiederholt, bis das gewünschte Ergebnis erreicht ist. In *Abbildung 11.13* ist die Vorgehensweise bei der Entwicklung von Formularen und Berichten schematisch dargestellt.

○
Testen der
Formulare



Abb. 11.13

Entwicklung von
Formularen und
Berichten

Einrichten der Benutzeroberfläche

Zu einer kompletten Datenbank gehört eine gute Benutzeroberfläche, die die komfortable Bedienung aller Elemente der Datenbank ermöglicht. Access bietet dazu die passenden Gestaltungsmöglichkeiten in Form von Symbolleisten und Menüleisten. Zusätzlich können Sie ein Formular erstellen, das die Funktion eines Hauptmenüs übernimmt oder mit dem benutzerdefinierte Optionen getroffen werden können. In diesem Abschnitt geht es um die Funktionen der Symbolleisten, die für den Entwickler einer komfortablen Datenbank wichtig sind. Sie erfahren beispielsweise, wie Sie einzelne Datenbankobjekte von einer Symbolleiste aus öffnen oder wie Sie VBA-Prozeduren bei der Auswahl eines Menüpunkts ausführen.

Mit der Einführung von Office 97 hat Microsoft das Konzept der Menü- und Symbolleisten gründlich überarbeitet. Das Ergebnis sind einheitliche Befehlszeilen, die nicht nur in Access, sondern auch in allen anderen Office-Anwendungen zum Einsatz kommen.

Die wichtigste Neuerung ist, dass Microsoft die Menü- und Symbolleisten zu einem einheitlichen Bedienelement – den Befehlsleisten – zusammengefasst hat. Dass diese weiterhin als eigenständige Menü- und Symbolleisten auf Ihrem Bildschirm erscheinen, liegt lediglich an der unterschiedlichen Ansicht. Es ist beispielsweise kein Problem, innerhalb einer Menüleiste ein Symbol unterzubringen. Probieren Sie einmal folgende Schritte aus:

Befehle mittels **[Alt]**-Taste verschieben

Drücken Sie die **[Alt]**-Taste und halten Sie sie gedrückt.

Klicken Sie auf ein beliebiges Symbol der Symbolleiste und ziehen Sie es zwischen zwei Menütitel (z. B. zwischen *Datei* und *Bearbeiten*).



Das Symbol erscheint nun inmitten der Menüleiste.

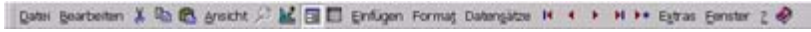


Abb. 11.14

Eine neue konfigurierte Menüleiste

Wenn Sie während des Drag & Drop-Vorgangs zusätzlich die **(Strg)**-Taste gedrückt halten, kopiert Access das Symbol.



Hinweis

Auch das Verschieben eines Symbols in ein Menü ist möglich. Ziehen Sie das Symbol dazu einfach auf einen Menütitel. Nach kurzer Zeit klappt das Menü herunter, und Sie können das Symbol platzieren. Access ergänzt das Symbol hierbei allerdings um eine Textbeschreibung zu einem vollständigen Menüpunkt.

Verwalten von Symbolleisten

Mit dem Menüpunkt *Ansicht > Symbolleisten > Anpassen* öffnen Sie das Dialogfenster *Anpassen*, das die Schaltzentrale für die Verwaltung aller Symbolleisten ist. In der Liste *Symbolleisten* sind alle Befehlsleisten mit ihren Namen aufgeführt, die momentan zur Verfügung stehen. Dies sind sowohl die von Access vordefinierten als auch die von Ihnen selbst erstellten Befehlsleisten.

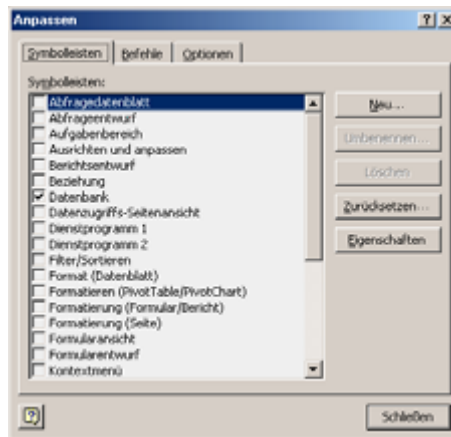


Abb. 11.15

Das Dialogfenster *Anpassen*

Die *Standard*-Menüleiste von Access verbirgt sich hinter dem Eintrag *Menüleiste*.

Durch Ankreuzen blenden Sie die jeweilige Symbolleiste ein. Dies ist beispielsweise dann sinnvoll, wenn Sie die Symbole neu anordnen oder zwischen den einzelnen Symbolleisten verschieben möchten.



Hinweis

Normalerweise sorgt Access allerdings selbst dafür, dass immer die passende Symbolleiste eingeblendet ist. Wenn Sie beispielsweise das Datenbankfenster aktivieren,



zeigt Access die Symbolleiste *Datenbank* an. In der Formularansicht blendet Access dagegen die Symbolleiste *Formularansicht* ein. Solche Symbolleisten werden auch als *kontextsensitiv* bezeichnet.

Änderungen, die Sie beispielsweise via Drag & Drop an den vordefinierten Symbolleisten vornehmen, werden von Access datenbankübergreifend gespeichert. Selbstdefinierte Symbolleisten stehen dagegen nur in der Datenbank zur Verfügung, in der Sie sie angelegt haben.

Sollten Sie einmal eine vordefinierte Symbolleiste in ihren Ursprungszustand zurücksetzen wollen, markieren Sie diese im Dialogfenster *Anpassen* und klicken auf *Zurücksetzen*.

Eigenschaften

Mit *Eigenschaften* aktivieren Sie ein weiteres Dialogfenster, in dem Sie Einstellungen zur momentan markierten Symbolleiste vornehmen. Es stehen folgende Eigenschaften zur Verfügung:

Eigenschaften einer Symbolleiste

Eigenschaft	Beschreibung
<i>Symbolleistenname</i>	Der Name der Symbolleiste. Änderungen sind nur bei selbstdefinierten Symbolleisten zulässig.
<i>Typ</i>	Der Typ der Symbolleiste. Bei benutzerdefinierten Symbolleisten legen Sie hier fest, ob diese als Menüleiste, Symbolleiste oder Popup-Menü erscheint.
<i>Andocken</i>	Bestimmt, ob und wie die Symbolleiste an einen Fensterrand angedockt werden kann.
<i>In Symbolleistenmenü anzeigen</i>	Bestimmt, ob der Name der Symbolleiste im Menü <i>Ansicht > Symbolleisten</i> auftaucht.
<i>Anpassen zulassen</i>	Legt fest, ob die Symbolleiste vom Benutzer verändert werden kann (z. B. durch Drag & Drop).
<i>Größenanpassung zulassen</i>	Wenn dieses Kontrollkästchen aktiviert ist, kann der Benutzer die Symbolleiste in der Größe verändern.
<i>Verschieben zulassen</i>	Bestimmt, ob der Benutzer die Symbolleiste verschieben darf.
<i>Einblenden > Ausblenden zulassen</i>	Legt fest, ob das Ein- oder Ausblenden der Symbolleiste erlaubt ist.

Tabelle 11.6



Mit der Schaltfläche *Standardeinstellungen wiederherstellen* setzen Sie eine vordefinierte Symbolleiste wieder auf ihre Standardeigenschaften zurück.

Abb. 11.16

Das Dialogfenster *Symboleisten-Eigenschaften*



Im Register *Optionen* des Dialogfensters *Anpassen* legen Sie allgemeine Einstellungen zu den Symbolleisten fest. Dazu gehört beispielsweise die Größe der Symbole. Außerdem können Sie bestimmen, ob Access QuickInfos und (zusammen mit den QuickInfos) Tastenkombinationen anzeigt. Auch die Art und Weise, wie Access ein Menü herunterklappt, können Sie hier einstellen. Alle Einstellungen dieses Registers wirken sich nicht nur auf Access, sondern auf alle Office XP-Anwendungen der Firma Microsoft aus.



Hinweis

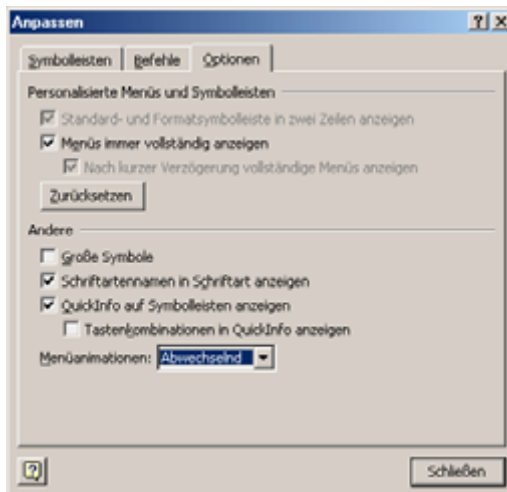


Abb. 11.17

Das Register *Optionen* des *Anpassen*-Dialogs

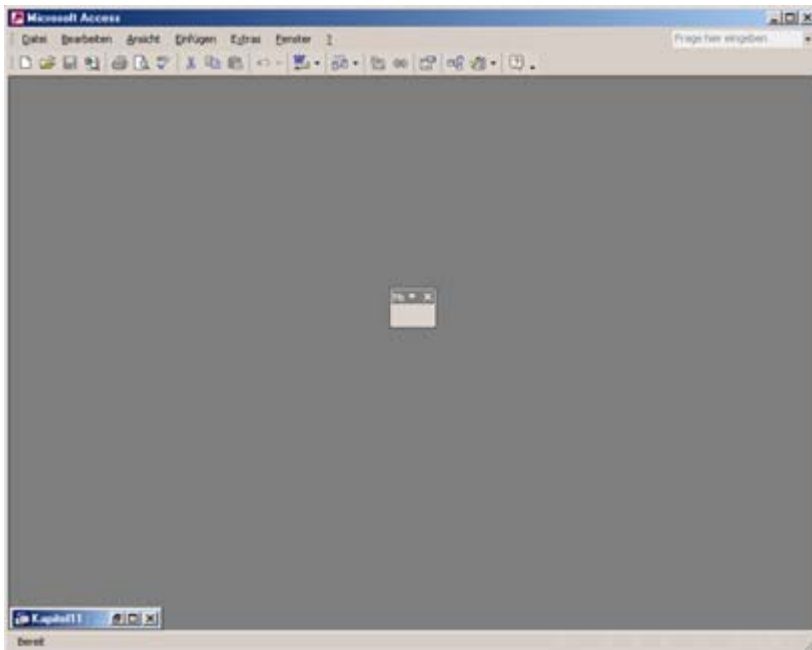




Selbstdefinierte Symbolleisten

Zu einer benutzerfreundlichen Datenbankanwendung gehört natürlich auch eine selbsterstellte Symbolleiste. Nur so ist sichergestellt, dass sich die Benutzer Ihrer Datenbank schnell und ohne großen Einarbeitungsaufwand zurechtfinden.

Neue Symbolleisten legen Sie über das Dialogfenster *Anpassen* an. Klicken Sie hier auf *Neu* und geben Sie der Symbolleiste einen eindeutigen Namen. Die neue Symbolleiste erscheint daraufhin auf Ihrem Bildschirm. Da sie noch keine Symbole enthält, ist sie sehr klein und leicht zu übersehen. Außerdem platziert Access die Symbolleiste inmitten Ihres Bildschirms.



○ Abb. 11.18

Eine neue, leere
Symbolleiste

Wenn Sie eine benutzerdefinierte Symbolleiste wieder löschen möchten, markieren Sie diese im *Anpassen*-Dialog und klicken auf *Löschen*.

Befehle hinzufügen und verwalten

Der neuen Symbolleiste können Sie nun Symbole – auch als *Befehle* bezeichnet – hinzufügen. Das Dialogfenster *Anpassen* enthält zu diesem Zweck ein Register namens *Befehle*, über das Sie eine Liste aller von Access zur Verfügung gestellten Befehle einblenden.



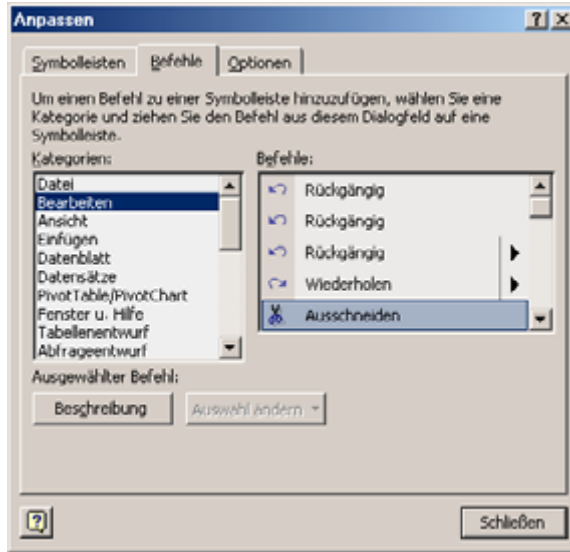


Abb. 11.19

Das Register *Befehle*

Alle Befehle sind nach Kategorien geordnet. Wenn Sie beispielsweise die Kategorie *Bearbeiten* auswählen, finden Sie hier Befehle wie *Ausschneiden*, *Kopieren* oder *Suchen*. Einige Befehle sind mit einem kleinen schwarzen Dreieck gekennzeichnet. Dies bedeutet, dass ein Klick auf das Symbol eine Liste mit weiteren Befehlen öffnet. Dies ist beispielsweise bei dem Befehl *Ansicht* der gleichnamigen Kategorie der Fall. Damit sich diese Gruppenbefehle von normalen Befehlen unterscheiden, werden sie in der fertigen Symbolleiste ebenfalls durch ein kleines Dreieck gekennzeichnet.

Um Ihrer neuen Symbolleiste einen Befehl hinzuzufügen, ziehen Sie ihn einfach per Drag & Drop aus der Befehlsliste in Ihre neue Symbolleiste. Sobald Sie den Mauszeiger über die Symbolleiste bewegen, markiert Access die genaue Position, an der der Befehl eingefügt wird, durch eine Einfügemarke. Dies ist besonders dann hilfreich, wenn Sie einen Befehl zwischen zwei bereits vorhandenen Befehlen anordnen möchten.

Befehl hinzufügen

Sollten Sie Ihrer Symbolleiste einmal einen falschen Befehl hinzugefügt haben, können Sie diesen problemlos wieder entfernen. Dazu gibt es zwei Möglichkeiten:

Löschen eines Befehls

- Aktivieren Sie mit der rechten Maustaste das Kontextmenü des überflüssigen Symbols und wählen Sie den Befehl *Löschen* aus.
- Ziehen Sie das Symbol aus der Symbolleiste auf den Access-Fensterhintergrund.
- Wenn Sie einen Befehl nachträglich an eine andere Stelle verschieben möchten, ziehen Sie ihn einfach per Drag & Drop an die neue Position.

Verschieben von Befehlen



Hinweis



Falls Sie einen Befehl kopieren und nicht verschieben möchten, halten Sie während des Drag & Drop-Vorgangs die **(Strg)**-Taste gedrückt. Natürlich funktioniert sowohl das Verschieben als auch das Kopieren nicht nur innerhalb einer, sondern auch zwischen zwei Symbolleisten.

Gruppieren von Befehlen

Damit bei umfangreichen Symbolleisten nicht die Übersicht verloren geht, haben Sie die Möglichkeit, eine Gruppe von Symbolen optisch voneinander abzugrenzen. Aktivieren Sie dazu das Kontextmenü des Befehls, bei dem die Gruppierung beginnen soll. Aus dem Menü wählen Sie anschließend den Befehl *Gruppierung beginnen* aus. Access fügt nun vor dem Symbol eine vertikale Trennlinie ein. Mit dem gleichen Menüpunkt können Sie diese bei Bedarf wieder entfernen.

Abb. 11.20

Gruppierte Befehle



Symbol auswählen

Bei Standardbefehlen ordnet Access diesen automatisch ein passendes Symbol zu. Dieses können Sie bei Bedarf abändern. Das Kontextmenü jedes Symbols stellt zu diesem Zweck die folgenden Befehle zur Verfügung, die alle mit dem Text *Schaltflächensymbol* beginnen:

Befehle zur Änderung eines Symbols

Befehl	Beschreibung
... kopieren	Kopiert die Grafik des Symbols in die Zwischenablage. Von hier aus können Sie diese beispielsweise einem anderen Befehl zuordnen oder zur Weiterbearbeitung in eine Grafikanwendung einfügen.
... einfügen	Fügt die momentan in der Zwischenablage enthaltene Grafik als Symbol ein. Die Auflösung sollte idealerweise 16 mal 16 Pixel betragen. Bei anderen Auflösungen nimmt Access eine Skalierung vor, worunter die Qualität in der Regel leidet.
... zurücksetzen	Stellt die von Access vorgegebene Standardgrafik wieder her.
... bearbeiten	Öffnet einen kleinen Grafik-Editor zur Bearbeitung des Symbols.
... ändern	Zeigt ein Untermenü zur Auswahl einiger Grafiken an.

Tabelle 11.7



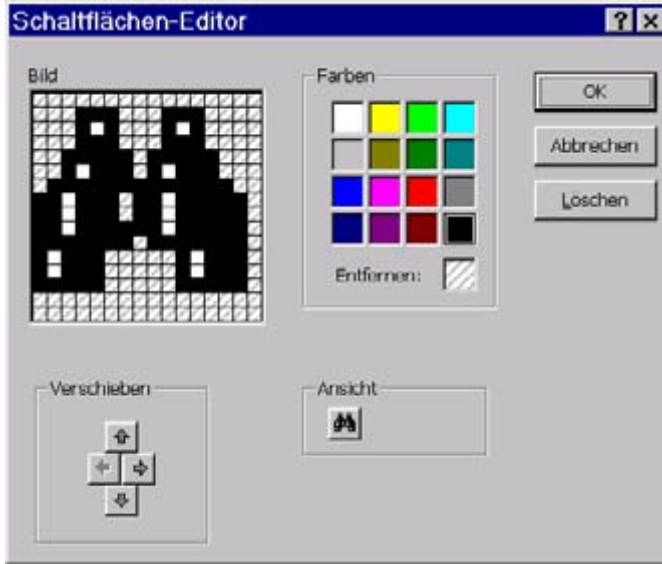


Abb. 11.21

Der Editor zur Bearbeitung einer Symbolgrafik

Zusätzlich zu einem Symbol kann ein Befehl auch durch einen Text ergänzt werden. Auch die alleinige Anzeige eines Textes ist möglich. Das Kontextmenü eines Befehls stellt zu diesem Zweck vier Menübefehle zur Verfügung:

Ansicht anpassen

Befehle zur Auswahl der Ansicht

Befehl	Beschreibung
<i>Standard</i>	Zeigt den Befehl in der von Access vorgegebenen Standardansicht an.
<i>Nur Text (immer)</i>	Der Befehl wird stets durch einen Text repräsentiert. Dies ist unabhängig davon, ob Sie den Befehl in einer Symbol- oder einer Menüleiste anordnen.
<i>Nur Text (in Menüs)</i>	Stellt den Befehl in Symbolleisten als Symbol und in Menüleisten als Text dar.
<i>Schaltflächensymbol und Text</i>	Zeigt sowohl ein Symbol als auch einen Text an.

Tabelle 11.8

Den Text, den Access zu einem Befehl anzeigt, bestimmen Sie über den Namen des Symbols. Um diesen zu ändern, aktivieren Sie das Kontextmenü und klicken hier auf den Menüpunkt *Name*. Access aktiviert nun innerhalb des Kontextmenüs ein Textfeld, über das Sie den Namen anpassen können.





Wenn Sie dem Befehl eine Zugriffstaste zuordnen möchten, stellen Sie dem gewünschten Buchstaben einfach ein &-Zeichen voran. Dies ist besonders bei selbsterstellten Menübefehlen wichtig, die Sie später noch kennen lernen werden. Wenn das &-Zeichen keine Zugriffstaste definieren, sondern in der Beschriftung auftauchen soll, müssen Sie dieses doppelt angeben. Der folgende Name bewirkt beispielsweise, dass Access die Beschriftung "Drucken & Versenden" anzeigt:

Drucken && Versenden

Hinweis



Beachten Sie, dass die von Access vorgegebenen Symbole größtenteils kontextsensitiv sind. Dies bedeutet, dass Access Befehle deaktiviert, die momentan nicht verwendet werden können. Wenn Sie Ihrer Symbolleiste beispielsweise ein Symbol zur Umschaltung eines Formulars in die Entwurfsansicht hinzufügen, ist dieses ausgegraut, wenn kein Formular geöffnet ist.

Zusätzliche Symboleigenschaften



Weitere Eigenschaften zu einem Symbol können Sie einstellen, indem Sie aus dem Kontextmenü den Befehl *Eigenschaften* auswählen. Auf einen Großteil dieser Eigenschaften haben Sie allerdings auch direkt über das Kontextmenü Zugriff. Die folgende Tabelle gibt Ihnen eine Übersicht:

Eigenschaften von Symbolen



Eigenschaft	Beschreibung
<i>Beschriftung</i>	Ist mit dem Namen des Symbols identisch.
<i>Verknüpfungstext</i>	Mit dieser Eigenschaft definieren Sie einen Text, den Access in Menüs rechtsbündig neben den einzelnen Menübefehlen anzeigt. Üblicherweise wird diese Eigenschaft dazu verwendet, die Tastenkombination einer Zugriffstaste anzuzeigen (z. B. <i>Strg+A</i>). Die Zugriffstaste wird dadurch jedoch nicht automatisch aktiv! Hierzu müssen Sie beispielsweise ein Tastaturbelegungsmakro einsetzen.
<i>QuickInfo</i>	Bestimmt den Text, der als QuickInfo des Befehls erscheint. Access zeigt QuickInfos als kleine gelbe Fenster unterhalb eines Symbols an, wenn Sie den Mauszeiger einige Zeit über einem Symbol belassen.
<i>Bei Aktion</i>	Ermöglicht die Verknüpfung des Befehls mit einer VBA-Prozedur oder einem Makro. Mehr dazu erfahren Sie in den folgenden Abschnitten. Wichtig: Wenn Sie hier etwas eintragen, geht die ursprüngliche Standardfunktion des Befehls verloren.
<i>Stil</i>	Bestimmt, ob der Befehl mit Grafik oder Text angezeigt wird. Das gleiche Ergebnis erzielen Sie auch über die betreffenden Befehle des Kontextmenüs.



Eigenschaften von Symbolen (Forts.)

Eigenschaft	Beschreibung
<i>Hilfedatei</i>	Der Name einer Hilfedatei mit Informationen zu einem selbstdefinierten Befehl.
<i>Hilfe-KontextID</i>	Die Kontext-ID einer Hilfeseite innerhalb der Hilfedatei.
<i>Parameter</i>	Ein beliebiger Text, der zusätzliche Informationen zu einem Befehl enthält. Diese Eigenschaft wird von Access ignoriert.
<i>Marke</i>	Gleiche Bedeutung wie die <i>Parameter</i> -Eigenschaft.
<i>Eine Gruppe beginnen</i>	Gleiche Funktion wie der Befehl <i>Gruppierung beginnen</i> des Kontextmenüs.

Tabelle 11.9

Mit dem Kombinationsfeld *Ausgewähltes Steuerelement* wechseln Sie schnell zu einem anderen Befehl, um dessen Eigenschaften zu bearbeiten. Dies erspart Ihnen Zeit, wenn Sie die Eigenschaften mehrerer Befehle in einem Rutsch verändern möchten.

Befehle selbst definieren

Neben den von Access vorgegebenen Befehlen haben Sie auch die Möglichkeit, eigene Befehle zu definieren und in Ihrer Symbolleiste unterzubringen. Im Register *Befehle* des Dialogs *Anpassen* stehen Ihnen zu diesem Zweck die Kategorien *Alle Tabellen* bis *Alle Makros* zur Verfügung. Wenn Sie beispielsweise die Kategorie *Alle Formulare* auswählen, zeigt die Liste *Befehle* alle Formulare Ihrer Datenbank an. Diese können Sie nun per Drag & Drop in eine beliebige Symbolleiste ziehen. Über das Kontextmenü und das Eigenschaftenfenster passen Sie das Symbol auf die gleiche Weise wie bei den von Access vorgegebenen Befehlen an. Sie können beispielsweise eine andere Symbolgrafik zuordnen oder den Befehlstext ändern.

Sobald Sie das Fenster *Anpassen* schließen, können Sie einen Test Ihres neuen Symbols vornehmen. Durch einen Klick öffnet Access das Formular in der durch die *Standardansicht*-Eigenschaft des Formulars vorgegebenen Ansicht. Die folgende Tabelle gibt Ihnen darüber Auskunft, wie Access die unterschiedlichen Datenbankobjekte beim Klick auf einen Befehl öffnet:



Öffnungsweise von Datenbankobjekten

Objektyp	Öffnungsweise
<i>Tabelle</i>	Datenblattansicht
<i>Abfrage</i>	Datenblattansicht
<i>Formular</i>	Wird durch die <i>Standardansicht</i> -Eigenschaft des Formulars vorgegeben.
<i>Bericht</i>	Seitenansicht
<i>Makro</i>	Führt das erste Makro aus. Bei Gruppenmakros startet Access das erste Makro der Gruppe.

Tabelle 11.10

Tipp



Noch komfortabler ordnen Sie ein Datenbankobjekt in einer Symbolleiste an, indem Sie dieses direkt via Drag & Drop aus dem Datenbankfenster in die Symbolleiste ziehen. Das Dialogfenster *Anpassen* darf hierbei nicht geöffnet sein.

Makros und VBA-Prozeduren verknüpfen

Im vorherigen Abschnitt haben Sie bereits die *BeiAktion*-Eigenschaft kennen gelernt, die Sie im Eigenschaftfenster von Befehlen finden. Diese Eigenschaft ist vergleichbar mit einer Ereignisseigenschaft, wie Sie sie z. B. aus Formularen kennen.

Wenn Sie von vornherein den Einsatz eines benutzerdefinierten Befehls planen, ziehen Sie einfach den Befehl *Benutzerdefiniert* aus der Befehlsliste in Ihre Symbolleiste. Sie finden diesen Befehl in der Kategorie *Datei*. Alternativ können Sie auch einen bereits vorhandenen Befehl abändern. Hierbei ist es unerheblich, ob es sich um einen selbsterstellten oder um einen vordefinierten Befehl handelt.

Sobald Sie in die *BeiAktion*-Eigenschaft eines vordefinierten Access-Befehls einen Makro- oder Funktionsnamen eintragen, verliert der Befehl diese Standardfunktion. Die Funktion des Befehls wird in diesem Fall durch Ihr Makro bzw. Ihre VBA-Prozedur ersetzt und damit zu einem selbstdefinierten Befehl. Um die ursprüngliche Funktion wiederherzustellen, wählen Sie aus dem Kontextmenü den Befehl *Zurücksetzen* aus.

Makro ausführen

Die Kombinationsfeldliste der Eigenschaft *BeiAktion* enthält alle Makros, die momentan in Ihrer Datenbank vorhanden sind. Wenn Sie hier ein Makro auswählen, führt Access dieses bei der Auswahl des Befehls aus.

VBA-Funktion ausführen

Alternativ können Sie auch den Namen einer VBA-Funktion eintragen. Die Funktion muss sich hierbei in einem Standardmodul befinden und mit dem Attribut *Public* versehen sein. Der folgende Eintrag führt beispielsweise dazu, dass Access beim Anklicken des Befehls die Funktion *KundeSuchen* ausführt:



=KundeSuchen ()

Die Angabe des Gleichheitszeichens und des leeren Klammernpaars ist erforderlich, damit Access den Prozeduraufruf von einem Makroaufruf unterscheiden kann.

Auch die Übergabe von Parametern ist möglich. Der folgende Aufruf startet die Prozedur *KundeSuchen* mit zwei Textparametern:

=KundeSuchen ("Kundenformular", "Liste")

Parameterübergabe

Anlegen neuer Menüleisten

Wie bereits erwähnt, sind Menüleisten nichts anderes als spezielle Symbolleisten. Die Handhabung ist also fast mit der von Symbolleisten identisch. Im Gegensatz zu Symbolleisten ist eine Menüleiste jedoch hierarchisch aufgebaut, wodurch sich einige Unterschiede ergeben.

Die Anlage einer neuen Menüleiste funktioniert auf die gleiche Weise wie bei Symbolleisten: Klicken Sie im Dialogfenster *Ansicht > Symbolleisten > Anpassen* auf die Schaltfläche *Neu* und geben Sie den Namen der neuen Menüleiste ein.

Anlage neuer Menüleisten

Um Access mitzuteilen, dass es sich bei der Symbolleiste um eine Menüleiste handelt, markieren Sie diese und klicken auf *Eigenschaften*. Wählen Sie im Eigenschaftfenster den Typ *Menüleiste* aus.

Abb. 11.22

Einstellung des Typs
Menüleiste





Mithilfe der folgenden Schritte fügen Sie der Menüleiste ein neues Pulldown-Menü hinzu:

- ① Wechseln Sie im Dialogfenster *Anpassen* zum Register *Befehle*.
- ② Wählen Sie die Kategorie *Neues Menü* aus.
- ③ Ziehen Sie den Befehl *Neues Menü* aus der Befehlsliste in Ihre neue Menüleiste.
- ④ Aktivieren Sie das Kontextmenü des neuen Menüs und vergeben Sie einen Namen.

Genau wie bei den Befehlen einer Symbolleiste können Sie für das Menü ebenfalls eine Zugriffstaste definieren, indem Sie einem Buchstaben des Namens ein &-Zeichen voranstellen. Der betreffende Buchstabe wird im fertigen Menü unterstrichen dargestellt.

Befehle hinzufügen

Dem neu erstellten Menü können Sie nun neue Befehle hinzufügen. Gehen Sie hierbei genau wie bei Symbolleisten vor, indem Sie einen Befehl aus der Befehlsliste in das Menü ziehen. Während des Drag & Drop-Vorgangs müssen Sie den Mauszeiger kurze Zeit über dem Menütitel ruhen lassen. Access klappt dann das Menü herunter und Sie können den Befehl platzieren.

Hinweis



Wenn Sie die Kategorie *Integrierte Menüs* im *Anpassen*-Dialog auswählen, zeigt Access in der Befehlsliste alle Standardmenüs an. Sie können Ihrer Menüleiste damit beispielsweise in einem Schritt das komplette *Bearbeiten*-Menü hinzufügen. Dies erspart Ihnen die Anordnung der betreffenden Befehle innerhalb des Menüs.

Menü bearbeiten

Zum Bearbeiten der Befehle eines Menüs klicken Sie dieses kurz an, damit Access das Dropdown-Menü und die darin enthaltenen Befehle anzeigt. Diese können Sie nun auf die gleiche Weise wie die Befehle einer Symbolleiste konfigurieren.

Gruppierungen

Ähnlich wie die Symbole einer Symbolleiste können Sie auch die Menübefehle eines Menüs gruppieren. Aktivieren Sie dazu das Kontextmenü eines Befehls, und wählen Sie den Menüpunkt *Gruppierung beginnen* aus. Access zeigt nun oberhalb des Menüs eine horizontale Trennlinie an.

Untermenüs

Menüs lassen sich nahezu beliebig verschachteln. Wenn Sie beispielsweise ein Untermenü anlegen möchten, ziehen Sie einfach den Befehl *Neues Menü* aus der Befehlsliste in ein bereits vorhandenes Menü. Hier platzieren Sie nun Befehle oder weitere Menüs.

Hinweis



Access stellt sehr viele Möglichkeiten zur Gestaltung von Menüleisten zur Verfügung. So können Sie beispielsweise Symbole zwischen den Menüs anordnen oder die einzelnen Menüs gruppieren. Bei einer bedienungsfreundlichen Datenbank-anwendung ist es jedoch wichtig, dass der Benutzer vertraute Elemente vorfindet.



Daher sollten Sie Ihre Menüleiste nach dem Vorbild der Microsoft Office-Anwendungen gestalten. Beim Einsatz von Untermenüs sollten Sie außerdem zu tiefe Verschachtelungen vermeiden oder sogar komplett auf deren Einsatz verzichten.

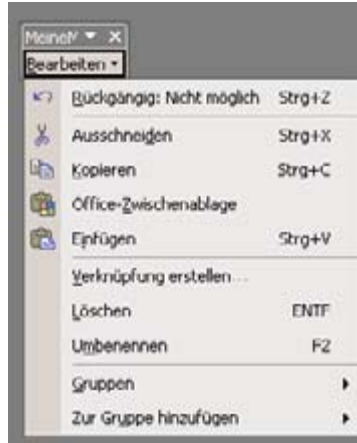


Abb. 11.23

Gruppierte Befehle eines Menüs

Einsatz von Symbol- und Menüleisten

Zur Integration von Befehlsleisten in Ihre Datenbankanwendung stellt Access verschiedene Möglichkeiten zur Verfügung.

Bei Menüleisten haben Sie die Möglichkeit, diese komplett gegen die *Standard-Menüleiste* von Access auszutauschen. Wählen Sie dazu den Menüpunkt *Extras > Start* aus. Im Kombinationsfeld *Menüleiste* wählen Sie anschließend die gewünschte Menüleiste über ihren Namen aus. Sobald Sie die Datenbank nun schließen und erneut öffnen, erscheint die neue Menüleiste.

Menüleisten integrieren

Um wieder die ursprüngliche Menüleiste zu aktivieren, öffnen Sie die Datenbank mit gedrückt gehaltener -Taste. Access ignoriert dadurch alle Einstellungen des *Start*-Dialogfensters und zeigt die *Standard-Menüleiste* an. Damit haben Sie dann wieder die Möglichkeit, das *Start*-Dialogfenster zu öffnen und Änderungen vorzunehmen. Wenn Sie Ihre benutzerdefinierte Menüleiste generell deaktivieren möchten, entfernen Sie deren Namen einfach aus dem Textfeld *Menüleiste*.



Tipp

Leider stellt Access zur Definition einer *Standard-Symbolleiste* keine passende Eigenschaft zur Verfügung. Stattdessen müssen Sie ein Makro namens *Autoexec* einsetzen, das Access automatisch beim Öffnen einer Datenbank ausführt. Innerhalb dieses Makros ordnen Sie die Aktion *EinblendenSymbolleiste* an und wählen die gewünschte Symbolleiste unter dem Aktionsargument *Symbolleistenname* aus. Unter *Show* wählen Sie zusätzlich *Ja* aus. Sobald Sie die Datenbank nun erneut öffnen, zeigt Access Ihre Symbolleiste an.



Standard-Symbolleisten ausblenden

Ein Problem ist hierbei, dass Access die Symbolleiste zusätzlich zu den *Standard-Symbolleisten* anzeigt. Die Lösung besteht darin, dass Sie vorher alle *Standard-Symbolleisten* deaktivieren. Dies erledigen Sie ebenfalls mittels *EinblendenSymbolleiste*-Aktion. Wählen Sie unter *Show* allerdings *Nein* aus, damit Access die ausgewählte Symbolleiste deaktiviert.

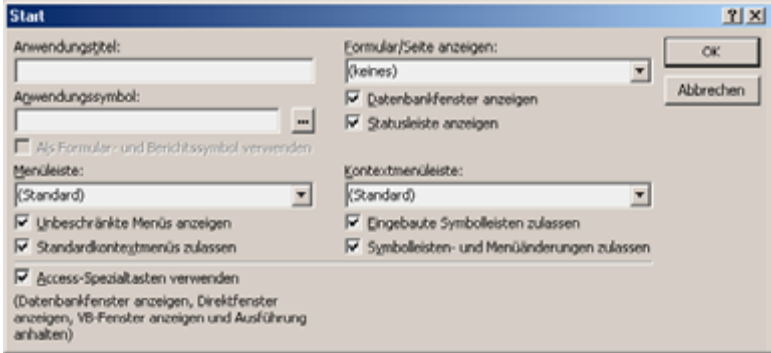


Abb. 11.24

Auswahl der Menüleiste im Dialogfenster Extras > Start

Abb. 11.25

Das *Autoexec*-Makro zum Einblenden einer Symbolleiste beim Start der Datenbank



Hinweis



Wenn Sie darauf Wert legen, dass die Benutzer Ihrer Datenbank die von Ihnen einblendete Symbolleiste nicht deaktivieren können, öffnen Sie über das Dialogfenster *Anpassen* das Eigenschaftsfenster der Symbolleiste. Löschen Sie hier die Markierung der Option *Einblenden/Ausblenden zulassen*. Mit den übrigen Eigenschaften dieses Fensters können Sie weitere Einschränkungen definieren und so beispielsweise das Verschieben oder Anpassen verhindern.



Kontextsensitive Befehlszeilen

Neben einer Menü- oder Symbolleiste, die während der kompletten Arbeitszeit mit einer Datenbank auf dem Bildschirm bleibt, stellt Access kontextsensitive Befehlszeilen zur Verfügung. Diese werden an ein bestimmtes Formular oder einen Bericht geknüpft und erscheinen immer dann, wenn das Formular oder der Bericht (in der Seitenansicht) aktiviert wird.

Um eine Menüzeile kontextsensitiv an ein Formular zu knüpfen, öffnen Sie dieses in der Entwurfsansicht und tragen unter der *Menüleiste*-Eigenschaft den Namen der gewünschten Menüleiste ein. Sobald Sie nun in die Formularansicht umschalten, ersetzt Access die aktuelle Menüzeile. Hierbei ist es unerheblich, ob es sich bei der ursprünglichen Menüleiste um eine Standard- oder eine selbstdefinierte Menüleiste handelt, die Sie im *Start*-Dialog aktiviert haben.

Sobald Sie z.B. das Datenbankfenster oder ein anderes Formular aktivieren, verschwindet die kontextsensitive Menüzeile wieder.

Mit der Eigenschaft *Symbolleiste* knüpfen Sie eine Symbolleiste kontextsensitiv an ein Formular. Die so aktivierte Symbolleiste erscheint allerdings *zusätzlich* zu den bereits vorhandenen Symbolleisten.

Kontextsensitive
Menüleisten

Kontextsensitive
Symbolleisten

Kontextmenüs

Kontextmenüs haben einen besonderen Status, da sie nicht mit Symbol- oder Menüzeilen vergleichbar sind, sondern einzelne Menüs einer Menüzeile darstellen. Dementsprechend verwaltet Access alle Kontextmenüs in einer speziellen Menüleiste namens *Kontextmenü*. Sie aktivieren diese Menüleiste, indem Sie im Dialogfenster *Anpassen* die Befehlsleiste *Kontextmenü* abhaken.

Abb. 11.26

Die Befehlsleiste
Kontextmenü



In der Befehlsleiste *Kontextmenü* sind alle vordefinierten Kontextmenüs von Access nach Kategorien angeordnet. Wenn Sie beispielsweise auf *Formular* klicken, erscheinen alle in Formularen auftauchenden Kontextmenüs als Untermenüs. Diese enthalten wiederum die einzelnen Befehle, die Sie bei Bedarf anpassen und ergänzen können.



Abb. 11.27

Die Befehle eines
Kontextmenüs



Erstellen eines
Kontextmenüs

- Um ein neues Kontextmenü anzulegen, gehen Sie wie folgt vor:
 - 1 Erstellen Sie eine normale Symbolleiste und ordnen Sie hier die gewünschten Befehle an, die später im Kontextmenü erscheinen sollen.
 - 2 Wechseln Sie im Dialogfenster *Anpassen* zum Register *Symbolleisten* und aktivieren Sie hier das Eigenschaftsfenster Ihrer neuen Symbolleiste.
 - 3 Wählen Sie unter *Typ* den Eintrag *Popup* aus. Dies teilt Access mit, dass es die Symbolleiste als Kontextmenü ansehen soll.
 - 4 Bestätigen Sie die nachfolgende Meldung mit *OK*.
 - 5 Schließen Sie das Eigenschaftsfenster.

Durch diese Schritte verschwindet Ihre Symbolleiste aus der gleichnamigen Liste. Den Zugriff auf das Kontextmenü erhalten Sie nun, indem Sie die Befehlsleiste *Kontextmenü* aktivieren. Das neue Kontextmenü finden Sie in der Gruppe *Anpassen*.

Hier können Sie bei Bedarf noch Änderungen an den Befehlen des Kontextmenüs vornehmen.

Hinweis



Löschen eines
Kontextmenüs

- Da Sie über die Liste *Symbolleisten* des *Anpassen*-Dialogfensters keinen Zugriff auf das Kontextmenü mehr haben, können Sie diese nicht so ohne weiteres löschen. Die folgenden Schritte zeigen Ihnen, wie Sie trotzdem ein Kontextmenü entfernen:
 - 1 Öffnen Sie vom *Anpassen*-Dialog aus das Eigenschaftsfenster einer beliebigen Symbolleiste.
 - 2 In der Liste *Ausgewählte* zeigt Access nun wieder eine vollständige Liste aller Befehlsleisten an. Wählen Sie hier Ihr Kontextmenü aus.
 - 3 Ändern Sie den *Typ* in *Symbolleiste* oder *Menüleiste*.
 - 4 Schließen Sie das Eigenschaftsfenster.

Die auf diese Weise umgewandelte Befehlsleiste erscheint nun in der Liste *Symbolleisten*. Löschen Sie die Befehlsleiste mit der *Löschen*-Schaltfläche.





Einsatz eines Kontextmenüs

Der Einsatz eines Kontextmenüs funktioniert auf ähnliche Weise wie die Verknüpfung einer kontextsensitiven Befehlsleiste mit einem Formular. Formulare stellen zu diesem Zweck eine Eigenschaft namens *Kontextmenüleiste* zur Verfügung, in die Sie den Namen Ihres gewünschten Kontextmenüs eintragen. Sobald Sie nun mit der rechten Maustaste auf den Formularhintergrund klicken, erscheint das Kontextmenü.

Neben Formularen stellen auch Berichte und alle Steuerelemente die *Kontextmenüleiste*-Eigenschaft zur Verfügung. Sie haben also die Möglichkeit, für jedes Steuerelement ein individuelles Kontextmenü zu definieren.

Wenn Sie in einem Formular grundsätzlich auf alle Kontextmenüs – also auf selbst-definierte oder vorgegebene – verzichten möchten, stellen Sie die *Kontextmenü*-Eigenschaft auf *Nein* ein.



Hinweis

Der Übersichtsmanager

Access bietet die Möglichkeit, ein Hauptmenü, genannt *Hauptübersicht*, auf ähnlich bequeme Weise wie Menüleisten generieren zu lassen. Wählen Sie dazu den Menübefehl *Extras > Datenbank-Dienstprogramme > Übersichts-Manager* aus. Wenn noch keine Übersicht vorhanden ist, wird automatisch eine neue erzeugt. Dabei geht der Manager genauso vor, wie bereits beim Programmbeispiel beschrieben. Es wird zunächst eine Tabelle angelegt, in der die einzelnen Menüeinträge gespeichert werden können. Zusätzlich wird ein Formular erstellt, das später zur Menüpunkt-auswahl dient. Im anschließend erscheinenden Dialogfenster können die einzelnen Menüpunkte und Befehle eingegeben werden. Als Aktionen für einen Menüpunkt stehen verschiedene Einträge zur Verfügung, darunter auch zum Ausführen von Makros und Funktionen oder zum Wechseln in ein Untermenü.

Mit dem Übersichts-Manager lassen sich auf einfache Art schön gestaltete Hauptmenüs entwerfen. Es sind auch für verschiedene Standardanwendungen, wie zum Beispiel eine Videosammlung, passende Bitmaps vorhanden, die exakt in das Übersichtsformular passen. Die Bitmaps, im *GIF*-Format, finden Sie in dem Unterverzeichnis *\Bitmaps\Dbwiz* von Office.



Hinweis

Zugriff auf Optionen

Einstellungen, die sich während der Arbeit mit der Datenbank nicht verändern, aber trotzdem ab und zu benötigt werden, lassen sich ebenso wie die Punkte eines Hauptmenüs in einer Tabelle speichern. Diese Optionen können zum Beispiel Angaben zum Benutzernamen und der Benutzernummer sein. Des Weiteren kann dort das Verzeichnis von Windows oder WinWord eingetragen werden.

Die Liste lässt sich je nach Anwendung erweitern. Der Anwendungsentwickler kann dort auch versteckte Eintragungen machen, an denen nur er selbst Änderungen vornehmen kann. Mit einer Option kann auch geprüft werden, ob die Datenbank zum ersten Mal geöffnet wurde, so dass dann zum Beispiel eine entsprechende Erststartgrafik erscheint.



Optionen in Tabelle speichern



Datenfelder der Optionentabelle

Die Felder der Tabelle *zOptionen* sind in *Tabelle 11.11* aufgeführt. Jede Option erhält bei der Eingabe einen Optionsnamen zur Identifikation, einen Namen, der im Formular angezeigt werden kann, eine Beschreibung, die genauere Auskünfte über den Zweck der Einstellung gibt, und eine Einstellung, die den Wert der Option setzt.

Felder der Tabelle *Optionen*

Feldname	Datentyp	Feldgröße
<i>OptionNr</i>	<i>AutoWert</i>	
<i>OptionName</i>	<i>Text</i>	100
<i>Name</i>	<i>Text</i>	100
<i>Beschreibung</i>	<i>Memo</i>	
<i>Einstellung</i>	<i>Text</i>	255
<i>Versteckt</i>	<i>Ja/Nein</i>	
<i>Standardwert</i>	<i>Text</i>	255

Tabelle 11.11

Zusätzlich sind hier noch die Felder *Versteckt* und *Standardwert* aufgeführt. Das Feld *Versteckt* wird benötigt, um solche Einstellungen wie *Erststart* nicht mit im Eingabeformular für die Optionen aufzuführen. Über das Feld *Standardwert* können vom Anwendungsentwickler Grundeinstellungen gemacht werden, die der Anwender nicht verändern kann. Mit diesen Standardwerten lassen sich die Einstellungen wieder in den Ausgangszustand zurücksetzen. Als Beispiel für verwendete Optionen soll wieder die Datenbank *Kapitel11.mdb* dienen. Die folgende Abbildung zeigt die Tabelle mit ihren Werten

OptionNr	OptionName	Name	Beschreibung	Einstellung	Versteckt	Standardwert
1	StandardSerienbriefvorlage	Standard-Serienbriefvorlage für Word	Bestimmt die Dokur	Normal	Nein	Normal
2	WinwordPfad	Pfad zum Installationsverzeichnis von WinWord	Gibt den Pfad und c:\winword	Nein	C:\WINWORD	
3	ErstStart		Bestimmt, ob die De	Nein	Ja	Ja
5	BenutzerNummer	BenutzerNummer	Eine Nummer (z.B. 195736)	Nein	0	
6	BenutzerName	Name des Benutzers	Der Name des Ben	Hans Peter	Nein	Benutzername
7	WindowsPfad	Pfad von Windows	Gibt das Arbeitsv	C:\WINDOWS	Nein	C:\WINDOWS
*	AutoWert				Nein	

Abb. 11.28

Eintragungen der Tabelle *zOptionen*

Über ein Formular kann dem Anwender die Möglichkeit gegeben werden, die Optionen zu verändern. Dazu lassen sich Prozeduren schreiben, die abfragen, ob ein Benutzername und eine Benutzernummer eingegeben wurden oder ob die angegebenen Verzeichnisse existieren. Für die oben getroffenen Einstellungen kann zur Eingabe zum Beispiel das in der folgenden Abbildung dargestellte Formular dienen. Die Einstellung *ErsterStart* wird im Formular nicht aufgeführt, da sie in der Optionentabelle als versteckt gekennzeichnet wurde.

Formular für Optionen

Name	Einstellung
Standard-Serienbriefvorlage für WinWord	Normal
Pfad zum Installationsverzeichnis von WinWord	C:\WINWORD
BenutzerNummer	195736
Name des Benutzers	Hans Peter

Ok

Abb. 11.29

Formular zur Eingabe von Optionen

Um die eingestellten Optionen in der Datenbank benutzen zu können, wird die Funktion *GetOption* verwendet. Damit ist es möglich, den Wert jeder beliebigen Option über ihren Namen zu ermitteln.

Funktionen für Optionen

```
Function GetOption(OptionName As String) As Variant
```

```
Dim Wert As Variant
```

```
Wert = DLookup("Einstellung", "zOptionen", _  
"OptionName = " & OptionName & "")
```

```
If IsNull(Wert) Then MsgBox "Option " & _  
OptionName & " nicht gefunden!"
```

```
GetOption = Wert
```

```
End Function
```

Der Funktion *GetOption* wird der Optionsname übergeben, über den jede Option eindeutig identifiziert werden kann. Die Funktion ermittelt zu diesem übergebenen Optionsnamen aus der Tabelle *Optionen* die zugehörige Einstellung.





Datenbanken verwalten

In jeder fertigen Datenbankanwendung sollte auch die Datenpflege mit berücksichtigt sein. Darunter versteht man das Komprimieren und Reparieren von Datenbanken.

Ziel des Komprimierens

Das Komprimieren einer Datenbank ist besonders wichtig, wenn häufig Daten gelöscht und hinzugefügt werden. Die gespeicherten Daten können dann fragmentiert vorliegen, was einen größeren Platzbedarf auf der Festplatte zur Folge hat. Außerdem wird die Leistung der Datenbank negativ beeinflusst, so dass bestimmte Aktionen, etwa das Suchen, langsamer ablaufen.

Ziel des Reparierens

Das Reparieren einer Datenbank ist erforderlich, wenn eine Datenbank beschädigt wurde. Dies kann passieren, wenn Access nicht auf normalem Wege verlassen wird, zum Beispiel bei Stromausfall, Hardwareproblemen oder einem Systemabsturz. Eine Datenbank kann nicht beschädigt werden, wenn sie regulär beendet wird.

Datenbanken reparieren und komprimieren

Datenpflege über Icons

Um die Datenpflege für den Anwender so angenehm und unkompliziert wie möglich zu gestalten, kann man auf der Grundlage der in der Datei *Msaccess.exe* eingebauten Schalter Symbole im Ordner der Datenbankanwendung erzeugen, die die Funktionen des Komprimierens und Reparierens übernehmen. Mit diesen Symbolen lassen sich die Aktionen mit einem Doppelklick ausführen.

Schalter zum Komprimieren und Reparieren

Die Befehlszeile, die bei den Icons eingetragen werden muss, lautet für die Komprimierung, wenn das Verzeichnis von Access *C:\PROGRAMME\MICROSOFT OFFICE\OFFICE* ist

```
"C:\PROGRAMME\MICROSOFT OFFICE\OFFICE\MSACCESS.EXE"  
"<Datenbank>" /Compact "<Zieldatenbank>"
```

und für das Reparieren

```
"C:\PROGRAMME\MICROSOFT OFFICE\OFFICE\MSACCESS.EXE"  
"<Datenbank>" /Repair
```

Wenn Sie eine Datenbank in einem Schritt komprimieren und reparieren wollen, können Sie beide Schalter hintereinander angeben. Die Befehlszeile lautet dann:

```
"C:\PROGRAMME\MICROSOFT OFFICE\OFFICE\MSACCESS.EXE"  
"<Datenbank>" /Compact /Repair
```

Hinweis



Wird eine Datenbank repariert, wird Access gestartet und die angegebene Datenbank repariert. Anschließend öffnet sich ein Meldungsfenster, das mit OK vom Anwender geschlossen werden muss. Das Komprimieren kann dagegen ohne Zutun des Benutzers geschehen. Access wird nach dem Komprimieren automatisch wieder geschlossen.

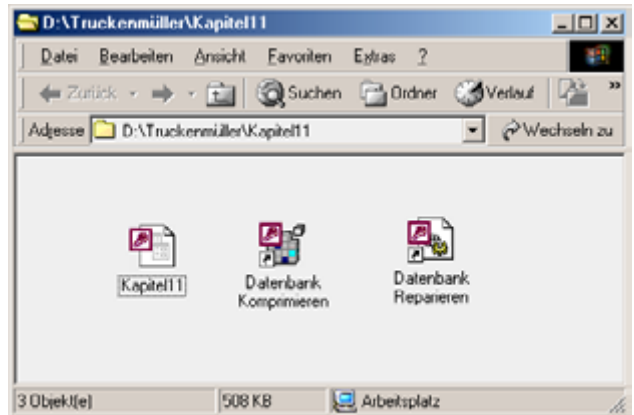


Beim Komprimieren einer Datenbank kann optional der Name einer Zieldatenbank angegeben werden. Da die zu komprimierende Datenbank dadurch in ihrem ursprünglichen Zustand verbleibt, kann sie als Sicherheitskopie verwendet werden. Wird keine Zieldatenbank angegeben, wird die Komprimierung direkt an der Datenbank vorgenommen. Aus Sicherheitsgründen sollte deshalb vor dem Komprimieren ein Backup der Datenbank angelegt werden.

Die Symbole erzeugen Sie einfach, indem Sie im gewünschten Ordner eine neue Verknüpfung erzeugen und im Eingabefeld *Befehlszeile* die entsprechende Kommandozeile eintragen. Mithilfe der erzeugten Symbole kann der Anwender dann komfortabel die Datenpflege erledigen.

Abb. 11.30

Programmgruppe der
Datenbank
Kapitel11.mdb



Der entscheidende Punkt beim Komprimieren und Reparieren einer Datenbank ist, dass die Datenbank in diesem Augenblick nicht geöffnet sein darf. Die Funktionen können auch nicht durchgeführt werden, wenn in einer Mehrbenutzerumgebung ein anderer Anwender auf die Datenbank zugreift.

Datenpflege während der Laufzeit

Damit eine Datenbank, während sie geöffnet ist, komprimiert und repariert werden kann, muss sie in eine Objektdatenbank und eine Tabellendatenbank aufgeteilt sein. Die Beschreibung dieses Modells erfolgte bereits im Abschnitt *Gesamtstruktur*. Als Beispiel liegt die Datenbank *Kapitel11.mdb* aufgeteilt in die Datenbanken *Master.mdb* und *Slave.mdb* vor. In diesem Fall kann die Datenbank *Slave.mdb*, die die Daten enthält, mittels der VBA-Funktionen *CompactDatabase* und *RepairDatabase* aus der Datenbank *Master.mdb* heraus komprimiert und repariert werden.

Im vorliegenden Beispiel enthält die Datenbank *Master.mdb* keine Tabellen, da vorher eingebundene Tabellen in neuen Verzeichnissen (Ihrem Installationsverzeichnis) von Access nicht mehr gefunden werden können. Damit Sie nicht den Tabellenverknüpfungs-Manager bemühen müssen, enthält die Datenbank eine



Hinweis

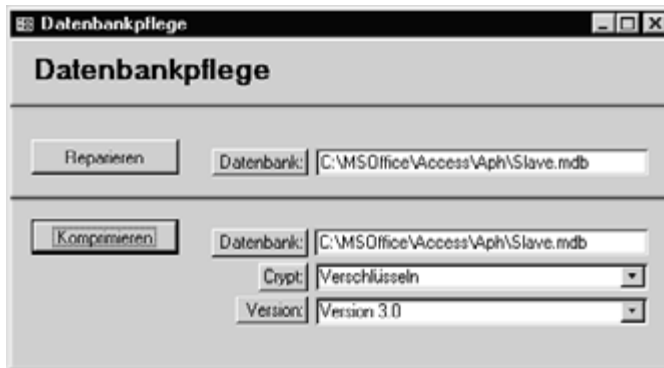


Prozedur, mit der die Tabellen der Datenbank *Slave.mdb* eingebunden werden können. Starten Sie zuerst die Datenbank *Master.mdb* und öffnen Sie das Formular *Einbinden*. Geben Sie den Pfad der Datenbank *Slave.mdb* an und klicken Sie auf die Schaltfläche *Tabellen einbinden*. Nun kann Access auf die Tabellen zugreifen.

Das Hauptmenü ließe sich zum Beispiel um den Punkt *Datenpflege* erweitern. Unter diesem Punkt kann dann ein Formular geöffnet werden, mit dem die Datenbank komprimiert und repariert werden kann. Die Einstellung der erforderlichen Parameter lässt sich über Text- bzw. Kombinationsfelder regeln. Die Parameter könnten auch in den benutzerdefinierten Optionen gespeichert werden.

Abb. 11.31

Beispiel für ein Formular zur Datenbankpflege



Funktionen zur Datenbankpflege

Mit den beiden neuen Funktionen *Reparieren* und *Komprimieren*, die im Folgenden aufgelistet sind, lassen sich die Aktionen über Befehlsschaltflächen ausführen. Beim Reparieren wird nur der Name der Datenbank samt Pfad übergeben.

```
Sub Reparieren(Pfad As Variant)
```

```
    DoCmd.Hourglass True
```

```
On Error GoTo Fehlerrep
```

```
    DBEngine.RepairDatabase Pfad
```

```
    MsgBox "Datenbank wurde erfolgreich repariert!", _
        vbInformation
```

```
    DoCmd.Hourglass False
```

```
End
```

```
Fehlerrep:
```

```
    DoCmd.Hourglass False
```





```
If Err.Number = 3005 Or Err.Number = 3024 Or _
    Err.Number = 53 Or Err.Number = 3044 Or _
    Err.Number = 76 Then

    Rem Fehler 3024: Angegebene Datei nicht gefunden

    Rem Fehler 3005: kein zulässiger Datenbankname

    Rem Fehler 3044: kein zulässiger Pfad

    MsgBox "Die angegebene Datenbank konnte nicht
" & _
        "gefunden werden.", vbExclamation

End If

If Err.Number = 3196 Then

    Rem Fehler 3196: kann nicht verwendet werden

    Rem Datenbank wird bereits verwendet.

    MsgBox "Die angegebene Datenbank wird zur " & _
        "Zeit verwendet.", vbExclamation

End If

Exit Sub

End Sub
```

Beim Komprimieren mittels VBA-Funktion muss beachtet werden, dass die komprimierte Datenbank einen neuen Namen erhält. Um direkt mit der Datenbank weiterarbeiten zu können, muss die alte Datenbank gelöscht werden (mittels *Kill*-Befehl) und die komprimierte Datenbank den Namen des Originals erhalten (mittels *Name...As*-Methode).

```
Sub Komprimieren(Pfad As Variant,
Crypt As Variant, Version As Variant)

    DoCmd.Hourglass True

    Dim Dummydatei As String
    Dim Dummypfad As String
    Dim Sortierung As Long
    Dim Datei As String

    On Error GoTo Fehlerkomp

    Datei = Dir(Pfad)
```





```
Dummpfad = Left(Pfad, Len(Pfad) - Len(Datei))
Dummydatei = Dummpfad & "Dummy.mdb"
DBEngine.CompactDatabase Pfad, Dummydatei, _
    dbLangGeneral, Val(Crypt) + Val(Version)

Kill Pfad

Name Dummydatei As Pfad

MsgBox "Datenbank wurde erfolgreich " & _
    "komprimiert!", vbInformation

DoCmd.Hourglass False

End

Fehlerkomp:

DoCmd.Hourglass False
If Err.Number = 3005 Or Err.Number = 3024 Or _
    Err.Number = 53 Or Err.Number = 3044 Or _
    Err.Number = 76 Then

    Rem Fehler 3024: Datei nicht gefunden

    Rem Fehler 3005: kein zulässiger
Datenbankname

    Rem Fehler 3044: ist kein zulässiger Pfad

    MsgBox "Die angegebene Datenbank konnte "
& _
        "nicht gefunden werden.", vbExclamation

End If

If Err.Number = 3196 Then

    Rem Fehler 3196: "|" kann nicht verwendet
    Rem werden;
    Rem Datenbank wird bereits verwendet.

    MsgBox "Die angegebene Datenbank wird " & _
        "zur Zeit verwendet.", vbExclamation

End If

If Err.Number = 3301 Then

    Rem Fehler 3301: Diese Operation kann nicht
```



```
Rem ausgeführt werden. Features dieser  
Rem Version von Microsoft Access sind nicht  
Rem verfügbar in Datenbanken mit älterem  
Format.
```

```
MsgBox "Die Datenbank kann nicht  
komprimiert" _  
_" werden. Einige Features aus der aktuellen  
" & _  
"Version sind nicht in der angegebenen " & _  
"Version verfügbar.", vbExclamation  
  
End If  
  
Exit Sub
```

End Sub

Die Werte der Konstanten für die Verschlüsselung und die Versionsnummer sind in den *Tabellen 11.12 und 11.13* aufgelistet. Durch Addition von Werten können beliebige Kombinationen erzeugt werden. Wird kein Wert angegeben, also Null, dann wird der Zustand der alten unkomprimierten Datenbank übernommen

Verschlüsselung
und Versions-
nummer

Konstantenwerte für die Ver- und Entschlüsselung

Konstante	Wert	Beschreibung
<i>dbEncrypt</i>	2	Verschlüsselt die Datenbank beim Komprimieren.
<i>dbDecrypt</i>	4	Entschlüsselt die Datenbank beim Komprimieren.

Tabelle 11.12

Konstantenwerte für die Versionsnummer

Konstante	Wert	Beschreibung
<i>dbVersion10</i>	1	Verwendet die JET-Datenbank-Engine der Version 1.0 beim Komprimieren.
<i>dbVersion11</i>	8	Verwendet die JET-Datenbank-Engine der Version 1.1 beim Komprimieren.
<i>dbVersion20</i>	16	Verwendet die JET-Datenbank-Engine der Version 2.0 beim Komprimieren.



Konstantenwerte für die Versionsnummer (Forts.)

Konstante	Wert	Beschreibung
<i>dbVersion30</i>	32	Verwendet die JET-Datenbank-Engine der Version 3.0 bzw. 3.5 beim Komprimieren.
<i>dbVersion40</i>	64	Verwendet die JET-Datenbank-Engine der Version 4.0 beim Komprimieren.

Tabelle 11.13

Sortierreihenfolge

Die Sortierreihenfolge wurde im vorliegenden Beispiel nicht im Formular berücksichtigt, sondern mit *dbLangGeneral* als konstant angenommen. Sie könnte über ein zusätzliches Kombinationsfeld im Formular ausgewählt werden. Die Eingabe der Konstanten in ein Kombinationsfeld wäre aber sehr mühsam und wahrscheinlich auch überflüssig, wenn die Datenbank zum Beispiel nur auf die Sortierreihenfolge *General* zurückgreifen soll. Eventuell ist eine Auswahl von Konstanten zu treffen, die wirklich in der Datenbank benötigt werden. Die Konstanten für die Sortierreihenfolge und ihre Bedeutung finden Sie in *Tabelle 9.1* dieses Buches oder in der Hilfe von Access unter dem Stichwort *CompactDatabase-Methode*.

Die Fehlerbehandlungsroutinen sind ein wichtiger Bestandteil der Funktionen *Reparieren* und *Komprimieren*. Sie prüfen in beiden Funktionen, ob die angegebene Datenbank vorhanden und geschlossen ist. Beim Komprimieren tritt zusätzlich ein Fehler auf, wenn die Optionen *Verschlüsseln* und *Entschlüsseln* mit den Versionsnummern 1.0 oder 1.1 kombiniert werden, da die Optionen in diesen Versionen nicht enthalten sind.

Zusätzliche Befehlszeilenoptionen

Neben dem Komprimieren und dem Reparieren lassen sich noch weitere Optionen beim Aufruf von Access angeben. Diese sind in der folgenden Tabelle erklärt.

Befehlszeilenoptionen

Option	Beschreibung
<i>/Excl</i>	Die Datenbank wird exklusiv geöffnet.
<i>/Ro</i>	Die Datenbank wird schreibgeschützt geöffnet.
<i>/User</i> Benutzername	Startet Microsoft Access unter Verwendung des angegebenen Benutzernamens.
<i>/Pwd</i> Kennwort	Startet Microsoft Access unter Verwendung des angegebenen Kennworts.



Befehlszeilenoptionen (Forts.)

Option	Beschreibung
<i>/Profile</i> Benutzerprofil	Startet Microsoft Access unter Verwendung des angegebenen Benutzerprofils.
<i>/Wrkgrp</i>	Startet Microsoft Access unter Verwendung der angegebenen Arbeitsgruppen-Informationsdatei.
<i>/Nostartup</i>	Startet Microsoft Access, ohne das Dialogfeld <i>Starten</i> anzuzeigen.
<i>/X</i> Makro	Beim Start von Access wird das angegebenen Makro ausgeführt.
<i>/Cmd</i>	Jeder Teil der Befehlszeile nach der Option <i>/Cmd</i> wird als Befehlszeilenargument an das Programm übergeben.
<i>/Runtime</i>	Startet Microsoft Access im Laufzeitmodus.
<i>/Convert</i> Zieldatenbank	Konvertiert eine Datenbank der Version 1.x oder 2.0 in eine Microsoft Access für Windows 95/98-Datenbank unter einem neuen Namen.
<i>/Compact</i> Zieldatenbank	Komprimiert eine Datenbank.
<i>/Repair</i>	Repariert eine Datenbank.

Tabelle 11.14

Die Option */Profile* ersetzt die Option *//ni* von früheren Access-Versionen. Mithilfe des Access Developer Toolkit können benutzerdefinierte Profile erzeugt werden.

Durch die Option */Runtime* wird Access im Laufzeitmodus gestartet. Dies verhindert, dass Datenbankobjekte in der Entwurfsansicht geöffnet werden können.

Die mit der Option */Cmd* übergebenen Informationen können durch die VBA-Funktion *Command* abgerufen werden. Außerdem werden die übergebenen Argumente im Register *Weitere* der *Optionen* von Access im Feld *Befehlszeilen-Argumente* eingetragen. Den Dialog *Optionen* öffnen Sie mit dem Menübefehl *Extras > Optionen*.

Benutzerdefinierte Hilfe

Zu jeder ausgereiften Datenbankanwendung gehört auch ein Hilfesystem, die so genannte *Onlinehilfe*. Sie ist eine gute Ergänzung zu einer schriftlichen Dokumentation. Meistens findet der Anwender sogar schneller die Lösung zu einem Problem, da mit umfangreichen Such- und Verzweigungsfunktionen gezielt gesucht werden kann und die Lösung in Form einer kurzen und referenzartigen Erklärung vorliegt.





Ein weiterer Vorteil des Hilfesystems ist die kontextsensitive Hilfe. Sie ermöglicht, an einem beliebigen Punkt durch Drücken der Taste **F1** die zum Thema passende Hilfe aufzurufen. Das Suchen nach der richtigen Hilfe entfällt so völlig.

Alle Funktionen, die Sie von den Hilfesystemen unter Windows gewöhnt sind, können Sie auch in Ihre eigene Datenbankanwendung implementieren. Um eine benutzerdefinierte Hilfe zu erstellen, benötigen Sie drei Programme.

Winhelp.exe



Zum einen die Datei *Winhelp.exe*, die Bestandteil von Windows ist und sich im Windows-Installationsverzeichnis befindet. Mit ihr können Hilfedateien mit der Endung *.hlp* geöffnet werden. Wenn Sie in einer Anwendung die Hilfe aufrufen, wird *Winhelp.exe* automatisch mit der angegebenen Hilfedatei gestartet. Der Anwender merkt dabei nichts vom Aufruf der Datei *Winhelp.exe*.

Des Weiteren benötigen Sie ein Textverarbeitungsprogramm, mit dem Sie die einzelnen Hilfeseiten erstellen können. Voraussetzung ist, dass die Textverarbeitung Dateien im **Rich Text Format (RTF)** speichern kann. Nur solche Dateien können vom Help-Compiler verarbeitet werden. Das Standardprogramm zum Erstellen von Hilfedateien ist die Textverarbeitung *Word für Windows* der Firma Microsoft. Wird mit anderen Textverarbeitungen gearbeitet, muss im Einzelnen geprüft werden, ob diese mit dem Help-Compiler zusammenarbeiten.

Der Help-Compiler



Als Drittes benötigen Sie den Help-Compiler. Er wandelt die mit der Textverarbeitung erstellte Hilfedatei in das von WinHelp lesbare Format um. Das Kompilieren der Hilfe-Textdatei hat einige Vorteile. WinHelp kann eine kompilierte Datei wesentlich schneller ausführen. Außerdem werden beim Kompilieren Fehler im Quelltext erkannt, die dann nachträglich behoben werden können. Zu guter Letzt bietet es dem Entwickler Schutz, da die kompilierte Version des Hilfesystems nicht abgeändert werden kann.

Hinweis



Der Compiler kann bei Microsoft angefordert werden.

Die Struktur des Hilfesystems

Entwicklersicht



Bei der Struktur eines Hilfesystems kann man zwei Sichtweisen unterscheiden. Auf der einen Seite steht die Sicht des Entwicklers. Für ihn ist das Hilfesystem aus einer Vielzahl von einzelnen, aufeinander folgenden Seiten aufgebaut. Jede Seite hat einen eindeutigen Namen, den *Kontextnamen*, und Sprungmarken, mit denen zu anderen Seiten des Systems gewechselt werden kann.

Anwendersicht



Aus der Sicht des Anwenders ist das Hilfesystem hierarchisch aufgebaut. Er wird von der obersten Ebene, dem Inhaltsverzeichnis, zu den einzelnen Kapiteln geführt, die die entsprechenden Unterthemen umfassen.



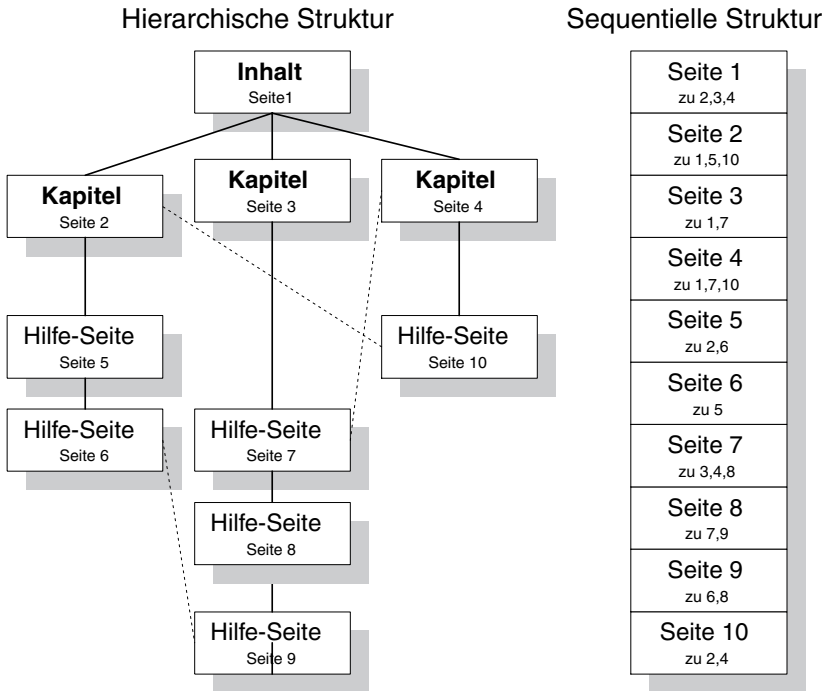


Abb. 11.32

Umsetzung der hierarchischen in die sequenzielle Struktur

Aufgabe des Entwicklers ist, die hierarchische Struktur in einer sorgfältigen Planung aufzustellen und sie dann in die sequenzielle Form umzusetzen. Dabei werden die einzelnen Seiten mit den entsprechenden Sprungmarken versehen, die nur die vom Entwickler vorgesehenen Wechsel zwischen den Hilfeseiten zulassen.

Zu Dokumentationszwecken empfiehlt es sich, für alle Seiten des Hilfesystems den Kontextnamen und eine Kurzbeschreibung sowie Verzweigungen von und zu dieser Seite in einer Tabelle zu speichern.



Hinweis

Implementierung in Access

Für die Implementierung der benutzerdefinierten Hilfe in Access werden die Eigenschaften *Hilfedatei* und *Hilfekontext* verwendet.

Die Eigenschaft *Hilfedatei* gilt nur für Formulare und Berichte. In ihr wird der Name und der Pfad einer benutzerdefinierten Hilfedatei angegeben, die für das aktuelle Formular oder den aktuellen Bericht gelten soll.

Hilfedatei-Eigenschaft

Die Eigenschaft *Hilfekontext* gilt sowohl für Formulare und Berichte als auch für Steuerelemente auf einem Formular. Hier kann jedem Objekt eine Kontextnummer zugewiesen werden, die der Seitennummer des gewünschten Hilfethemas in der Hilfedatei entspricht. Der Wertebereich der eintragbaren Ganzzahl liegt zwischen 0 und 2.147.483.647. Der Standardwert lautet 0.

Hilfekontext-Eigenschaft





Wenn Sie sich nun in einem Formular in der Formularansicht befinden und die Taste **[F1]** drücken, wird die in der Eigenschaft *Hilfedatei* eingetragene Hilfe geladen. Danach wird zu der Seite gesprungen, die im Feld *Hilfekontext* des Formulars eingetragen ist, unter der Voraussetzung, dass in den Eigenschaften der Steuerelemente der Standardwert *o* eingetragen ist. Ansonsten wird der Wert des aktiven Steuerelements als Suchmarke für die Seitenzahl verwendet.



Hinweis

Lautet der Eintrag in der Eigenschaft *Hilfekontext* sowohl für das Formular als auch für alle Steuerelemente *o*, dann wird ein Access-Hilfethema angezeigt.

Das Sicherheitssystem

Am Schluss des Entwicklungszyklus steht das Einrichten des Sicherheitssystems. Dabei werden zwei Zugriffsbeschränkungen unterschieden. Der Zugriff auf Objekte und der Zugriff auf Daten.

Das Schützen von Objekten dient dem Anwendungsentwickler. Er kann so verhindern, dass an der von ihm erstellten Datenbank Veränderungen, seien diese absichtlich oder aus Versehen herbeigeführt, vorgenommen werden. Beim Schützen von Daten kommt es darauf an, gewisse Datensätze nur bestimmten Personen zugänglich zu machen, d.h., eine Hierarchie aufzubauen, in der die höchste Person Zugriff auf alle Daten hat und die Personen am unteren Ende nur die für ihre Arbeit wichtigen Informationen erhalten. Das Schützen der Daten ist also im Interesse des Anwenders bzw. des Käufers der Anwendung.

Im folgenden Kapitel wird der Zugriff auf das Sicherheitssystem mithilfe von VBA ausführlich beschrieben.





Zugriff auf das Sicherheitssystem



Zugriff auf Benutzer und Benutzergruppen	605
Anlegen neuer Benutzer und Benutzergruppen	608
Zuordnung von Benutzern und Gruppen	613
<i>Container- und Document-Objekte</i>	616
Sichern einer Datenbank	622





Access stellt zwei unterschiedliche Sicherheitsmechanismen zur Verfügung. Der erste erlaubt es, eine Datenbank mit einem Kennwort zu schützen. Dies erledigen Sie beispielsweise über den Menüpunkt *Extras > Sicherheit > Datenbankkennwort festlegen*. Beim Öffnen der Datenbank müssen Sie nun das Kennwort eingeben, bevor diese geöffnet wird. Das Kennwort wird hierbei in verschlüsselter Form innerhalb der Datenbank gespeichert. Wenn Sie die Datenbank also kopieren und weitergeben, bleibt der Schutz aktiv.

Das Schützen einer Datenbank über ein Kennwort ist sehr einfach zu handhaben. Es eignet sich besonders für Einzelplatzsysteme oder kleine Benutzergruppen. Die Nachteile liegen auf der Hand:

- Jeder Benutzer, dem das Kennwort bekannt ist, hat Zugriff auf alle Datenbanken, die mit diesem Kennwort geschützt werden.
- Wenn ein Benutzer das Kennwort verrät, ist uneingeschränkter Zugriff möglich.
- Es kann nur die komplette Datenbank geschützt werden. Wenn der Benutzer einmal Zugriff auf die Datenbank hat, kann er alle Objekte einsehen und verändern.
- Eine mit einem Kennwort geschützte Datenbank kann nicht repliziert werden.

Alle Schwachpunkte werden durch einen weiteren Sicherheitsmechanismus behoben, der jedoch aufgrund der hohen Flexibilität und Sicherheit in der Handhabung komplizierter ist. Dieses Kapitel beschäftigt sich mit dem Zugriff auf diesen Sicherheitsmechanismus des DAO-Datenbankzugriffs. Im Folgenden wird dieser Mechanismus auch kurz als *Sicherheitssystem* bezeichnet.

Das Sicherheitssystem von Access ermöglicht es, Teile einer Datenbank oder auch die komplette Datenbank gegen den unbefugten Zugriff zu schützen. Zu diesem Zweck können in Access einzelne Benutzer angelegt werden, denen ein Passwort zugewiesen wird. Wenn der Benutzer Access startet, wird er von Access zur Eingabe eines Passwortes aufgefordert. Erst wenn dieses korrekt eingegeben wurde, startet Access, und der Benutzer ist nun "angemeldet". Wenn er im Folgenden eine Datenbank öffnet, überprüft Access, ob und welche Zugriffsberechtigungen der Benutzer auf die einzelnen Datenbankobjekte hat.

Benutzergruppen —○ In der Praxis kommt es häufig vor, dass mehrere Benutzer die gleichen Rechte haben. Aus diesem Grund werden die Benutzer in Benutzergruppen eingegliedert. Sobald ein Benutzer Mitglied in einer bestimmten Gruppe ist, erhält er automatisch die Zugriffsrechte dieser Gruppe.

Systemdatenbank —○ Damit dieses Zusammenspiel zwischen Access-Anmeldung, Benutzern, Benutzergruppen und geschützten Datenbanken funktioniert, ist das Sicherheitssystem in zwei Teile gegliedert. Der erste Teil beschäftigt sich mit den für die Anmeldung relevanten Dingen. Dazu gehören die Benutzer und die Benutzergruppen. Die entsprechenden Daten werden innerhalb von Access in einer Systemdatenbank mit dem Namen *system.mdw* gespeichert, die Sie im Access-Installationsverzeichnis finden.



Diese Systemdatenbank wird bei der Installation von Access neu angelegt und ist damit von diesem Zeitpunkt an wirksam. Alle Benutzer und Gruppen, die Sie nach der Installation anlegen, werden in dieser Datenbank gespeichert.

Die Gesamtheit aller Benutzer und Benutzergruppen der Datei *system.mdw* wird auch als *Arbeitsgruppe* bezeichnet. Über den Arbeitsgruppenadministrator von Access haben Sie die Möglichkeit, sich einer anderen Arbeitsgruppe und damit einer anderen *system.mdw*-Datenbank anzuschließen, die beispielsweise auf einem Netzwerkservers platziert ist.

Bei der Installation von Access wird nicht automatisch ein Startsymbol für den Arbeitsgruppenadministrator angelegt. Zum Start wählen Sie den Menüpunkt *Ausführen des Start-Menüs* von Windows an. Unter *Öffnen* geben Sie den folgenden Text (ggf. mit angepasstem Windows-Systemordner) an und bestätigen die Eingabe mit *OK*:

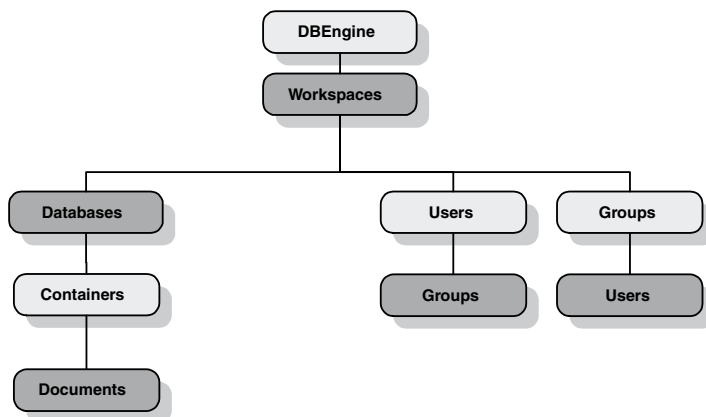
C:\Windows\System\WrkgAdm.exe

Die zweite Komponente des Sicherheitssystems bezieht sich auf die einzelnen Datenbanken. Hier werden Informationen darüber abgelegt, welche Objekte welchen Zugriff erlauben. So können Sie beispielsweise definieren, dass ein bestimmter Benutzer die Daten einer Tabelle zwar verändern kann, die Struktur der Tabelle aber geschützt ist.

In diesem Kapitel geht es darum, wie Sie das Sicherheitssystem von VBA aus steuern. Alle Beispiele zu diesem Kapitel finden Sie in der Datenbank *Kapitel12.mdb*.

Zugriff auf Benutzer und Benutzergruppen

Ähnlich wie der Zugriff auf Tabellen, Daten und Abfragen wird auch der Zugriff auf das Sicherheitssystem über eine Objekthierarchie bewerkstelligt.



Arbeitsgruppe



Hinweis

Abb. 12.1

Der für das Sicherheitssystem relevante Teil der Objekthierarchie





Workspace-Objekt

An der Spitze dieser Objekthierarchie steht das *DBEngine*-Objekt, das über die DAO-Datenbank-Engine von Access den Zugriff auf die *Workspaces*-Auflistung erlaubt. In der Regel ist in dieser Auflistung nur ein einzelnes *Workspace*-Objekt enthalten, auf das Sie über die Positionsnummer 0 Bezug nehmen:

```
DBEngine.Workspaces(0)
```

Auch die folgende Kurzschreibweise ist zulässig:

```
DBEngine(0)
```

Dieses *Workspace*-Objekt korrespondiert direkt mit der Datei *system.mdw* und stellt damit eine Arbeitsgruppe oder Arbeitsumgebung dar. Zur besseren Handhabung von *Workspace*-Objekten stellt Access den *Workspace*-Objektdatentyp zur Verfügung, der eine Referenz auf eine Arbeitsgruppe speichern kann:

```
Dim WS As Workspace
```

```
Set WS = DBEngine.Workspaces(0)
```

Users- und Groups-Auflistungen

Auf den tieferen Ebenen der Hierarchie werden die zwei Komponenten des Sicherheitssystems sichtbar. Der linke Ast erlaubt den Zugriff auf Datenbanken und die darin enthaltenen Objekte. Hier können den einzelnen Objekten gezielt Zugriffsrechte zugewiesen werden. Der rechte Ast ermöglicht den direkten Zugriff auf Benutzer (*Users*-Auflistung) und Benutzergruppen (*Groups*-Auflistung).

Mit diesen Auflistungen lässt sich auf die gleiche Weise arbeiten, wie Sie dies bereits vom Zugriff auf Daten oder Datenbankobjekte kennen. So stellt die Auflistung *Users* mehrere Objekte des Typs *User* zur Verfügung, die einzelne Benutzer repräsentieren. Auch ein passender Objektdatentyp steht unter dem Namen *User* zur Verfügung.

Benutzername

Jeder Benutzer besitzt einen Benutzernamen, der mithilfe der *Name*-Eigenschaft des betreffenden *User*-Objekts in Erfahrung gebracht werden kann. Das folgende Beispiel listet die Namen aller Benutzer der aktuellen Arbeitsumgebung im Testfenster auf:

```
Sub STestprozedur1()
```

```
Dim WS As Workspace  
Dim Benutzer As User
```

```
Set WS = DBEngine(0)
```

```
WS.Users.Refresh
```

```
For Each Benutzer In WS.Users
```

```
Debug.Print Benutzer.Name
```



Next Benutzer

End Sub

Zunächst wird eine Referenz auf die aktuelle Arbeitsgruppe in der Variablen *WS* gespeichert. Anschließend wird die *Users*-Auflistung aktualisiert, damit auch erst kürzlich hinzugefügte Benutzer berücksichtigt werden. Die folgende Schleife gibt schließlich die Namen der einzelnen Benutzer im Testfenster aus.

Wenn Sie noch keine Benutzer angelegt haben, sieht das Ergebnis der Prozedur folgendermaßen aus:

Standardbenutzer

Admin
Creator
Engine

Die beiden Benutzer *Creator* und *Engine* sind für Access-interne Zwecke reserviert. *Admin* steht als Abkürzung für *Administrator*. Der Administrator ist für die komplette Pflege einer Datenbank verantwortlich und hat daher in der Regel unbeschränkten Zugriff auf alle Datenbanken und Objekte einer Arbeitsgruppe.

Wenn Sie Access starten, werden Sie automatisch immer als Administrator unter dem Namen *Admin* angemeldet. Da diesem bei der Installation von Access allerdings kein Passwort zugewiesen wird, erscheint das entsprechende Dialogfenster zur Passworтеingabe nicht und die Anmeldung wird im Hintergrund durchgeführt.

Den aktuellen Benutzer können Sie leicht mithilfe der *CurrentUser*-Funktion ermitteln. Wenn Sie die Anweisung

Print CurrentUser

bei deaktiviertem Anmeldeverfahren im Testfenster ausführen, gibt die Funktion die Zeichenkette *Admin* zurück.

Genau wie bei dem Zugriff auf Benutzer kann mithilfe der *Groups*-Auflistung auf die einzelnen Benutzergruppen (*Group*-Objekte) verwiesen werden. Auch hier stellt Access wieder einen passenden Objektdatentyp mit dem Namen *Group* zur Verfügung. Das folgende Beispiel listet alle Benutzergruppen im Testfenster auf:

Zugriff auf
Benutzergruppen

Sub STestprozedur2 ()

Dim WS As Workspace
Dim Gruppe As GROUP

Set WS = DBEngine(0)

WS.Groups.Refresh

For Each Gruppe In WS.Groups





```
Debug.Print Gruppe.Name
```

```
Next Gruppe
```

```
End Sub
```

Standardbenutzer-
gruppen

Die Prozedur gibt das folgende Ergebnis aus:

```
Admins  
Users
```

Diese zwei Gruppen werden ebenfalls bei der Installation von Access vordefiniert und können durch eigene Gruppen ergänzt werden. Die Gruppe *Admins* besitzt die umfangreichsten Zugriffsrechte. Hier ist beispielsweise der Benutzer *Admin* Mitglied, unter dem Sie beim Start von Access automatisch angemeldet werden.

Jeder Benutzer kann Mitglied in mehreren Gruppen sein. So ist der Benutzer *Admin* gleichzeitig auch Mitglied in der Gruppe *Users* (Benutzer).

Hinweis



Wenn Sie über das Menü *Sicherheit* von Access auf das Sicherheitssystem zugreifen, gibt Access sowohl für die Standardbenutzer als auch für die Standardbenutzergruppen deutsche Namen an (z. B. *Administratoren* und *Benutzer*). Diese Objekte sind identisch mit den betreffenden *User*- und *Group*-Objekten. Wenn Sie selbst neue Benutzer und Gruppen anlegen, stimmen die Namen wieder überein.

Anlegen neuer Benutzer und Benutzergruppen

Das Anlegen neuer Benutzer wird mithilfe des *New*-Schlüsselworts bewerkstelligt, das sowohl die *Dim*- als auch die *Set*-Anweisung zur Verfügung stellt. Bevor der neue Benutzer mithilfe der *Append*-Methode der *Users*-Auflistung gespeichert wird, muss der Benutzername (*Name*-Eigenschaft), die persönliche Identifikationsnummer (*PID*-Eigenschaft) und das Passwort (*Password*-Eigenschaft) festgelegt werden.

Benutzername

Bei der Namensvergabe über die *Name*-Eigenschaft muss darauf geachtet werden, dass dieser für die aktuelle Arbeitsgruppe eindeutig ist. Es dürfen also nicht zwei Benutzer mit dem gleichen Namen angelegt werden. Ansonsten gelten die gleichen Regeln wie bei der Namensvergabe von Objekten.

PID

Die PID ist ebenfalls eine Zeichenkette, die eine Länge zwischen 4 und 20 Zeichen besitzen muss. Access nutzt intern eine Kombination aus Benutzername und PID, um einen Benutzer eindeutig zu identifizieren. Die PID müssen Sie auf jeden Fall geheim halten! Wenn jemandem der Benutzername sowie die PID bekannt ist, kann er sich unberechtigten Zugriff zur Datenbank verschaffen! Dies gilt auch dann, wenn das Passwort des Benutzers unbekannt ist.



Das Passwort darf eine Länge von 14 Zeichen nicht überschreiten. Abgesehen von dem ASCII-Zeichen o sind alle Zeichen erlaubt. Im Gegensatz zum Benutzernamen wird die Groß- und Kleinschreibung des Passworts bei der Anmeldung des Benutzers berücksichtigt!

Passwort

Die folgende Prozedur legt einen neuen Benutzer mit dem Namen *Willi Müller* an:

```
Sub STestprozedur3()  
  
Dim WS As Workspace  
Dim NeuerBenutzer As User  
  
Set WS = DBEngine(0)  
  
Set NeuerBenutzer = New User  
  
With NeuerBenutzer  
    .Name = "Willi Müller"  
    .PID = "AB1234"  
    .Password = "Willi"  
End With  
  
End Sub
```

Der neue Benutzer wird nicht automatisch der *Users*-Auflistung hinzugefügt. Bei dem neuen Benutzer handelt es sich also nur um ein temporäres *User*-Objekt. Um das Objekt endgültig zu speichern, muss es explizit der *Users*-Auflistung hinzugefügt werden. Diese Aufgabe erledigt die *Append*-Methode der *Users*-Auflistung. Diese Methode erwartet als Parameter den Variablennamen des *User*-Objekts, das Sie der Auflistung hinzufügen möchten. Die vollständige Prozedur hat folgenden Aufbau:

Die *Append*-Methode

```
Sub STestprozedur3()  
  
Dim WS As Workspace  
Dim NeuerBenutzer As User  
  
Set WS = DBEngine(0)  
  
Set NeuerBenutzer = New User  
  
With NeuerBenutzer  
    .Name = "Willi Müller"  
    .PID = "AB1234"  
    .Password = "Willi"  
End With  
  
WS.Users.Append NeuerBenutzer  
  
End Sub
```





Um sich von dem neu angelegten Benutzer zu überzeugen, führen Sie einfach die Prozedur *STestprozedur* aus, die alle Benutzer im Testfenster auflistet. Eine andere Möglichkeit besteht darin, dass Sie das Datenbankfenster von Access aktivieren und anschließend den Menüpunkt *Extras > Sicherheit > Benutzer und Gruppenkonten* ausführen. In der Liste des Kombinationsfeldes *Name* erscheint nun der neue Benutzer.

Hinweis



Sollte bereits ein Benutzer mit dem Namen *Willi Müller* vorhanden sein, löst Access bei der Ausführung der *Append*-Methode einen Laufzeitfehler mit der Nummer 3390 aus. Das folgende Beispiel zeigt, wie dieser abgefangen werden kann:

```
Sub STestprozedur4()  
  
On Error GoTo Err  
  
Dim WS As Workspace  
Dim NeuerBenutzer As User  
  
Set WS = DBEngine(0)  
Set NeuerBenutzer = New User  
  
With NeuerBenutzer  
    .Name = "Willi Müller"  
    .PID = "AB1234"  
    .Password = "Willi"  
End With  
  
WS.Users.Append NeuerBenutzer  
  
Exit Sub  
  
Err:  
  
Beep  
  
If Err = 3390 Then  
    MsgBox "Der Benutzer ist bereits vorhanden!"  
Else  
    MsgBox Err.Description  
End If  
  
Exit Sub  
  
End Sub
```

Hinweis



Fehlt eine der drei Angaben *Name*, *PID* oder *Password* bei Ausführung der *Append*-Methode, tritt ebenfalls ein Laufzeitfehler auf!



Eine Besonderheit der beiden Eigenschaften *PID* und *Password* ist, dass der Zugriff nur so lange möglich ist, wie der Benutzer noch nicht per *Append*-Methode gespeichert wurde. Es gibt also (sinnvollerweise) keine Möglichkeit, bei einem bereits vorhandenen Benutzer das Passwort abzufragen.

Wenn Sie das Passwort eines Benutzers nachträglich ändern möchten, verwenden Sie die *NewPassword*-Methode des betreffenden *User*-Objekts:

```
<User>.NewPassword <Altes Passwort>, <Neues Passwort>
```

Das folgende Beispiel ändert das Passwort des vorher angelegten Benutzers *Willi Müller* von *Willi* auf *Müller*:

```
Sub STestprozedur5()  
  
Dim WS As Workspace  
  
    Set WS = DBEngine(0)  
  
    WS.Users![Willi Müller].NewPassword "Willi",  
    "Müller"  
  
End Sub
```

Passwort nach-
träglich ändern

Aktivierung des Anmeldeverfahrens

Das Anmeldeverfahren von Access wird erst dann aktiv, wenn Sie dem Benutzer *Administrator* ein Passwort zuweisen. Dies können Sie entweder über den Menüpunkt *Extras > Sicherheit > Benutzer und Gruppenkonten* oder die folgende Prozedur erledigen:

```
Sub STestprozedur6()  
  
Dim WS As Workspace  
  
    Set WS = DBEngine(0)  
  
    WS.Users!Admin.NewPassword "", "Access"  
  
End Sub
```

Wenn Sie die Prozedur ausführen und Access anschließend verlassen, müssen Sie sich beim nächsten Start von Access unter dem Namen *Administrator* mit dem Passwort *Access* anmelden.



Hinweis

Um das Anmeldeverfahren wieder zu deaktivieren, wählen Sie den Menüpunkt *Sicherheit > Benutzer und Gruppenkonten* aus. Klicken Sie dann auf die Registerkarte *Anmeldungskennwort ändern* und weisen Sie dem Administrator wieder ein leeres Passwort zu, oder führen Sie die folgende Prozedur aus:

Anmeldeverfahren
deaktivieren





```
Sub STestprozedur7()  
  
Dim WS As Workspace  
  
    Set WS = DBEngine(0)  
  
    WS.Users!Admin.NewPassword "Access", ""  
  
End Sub
```

Anlegen neuer Benutzergruppen

Das Anlegen einer neuen Benutzergruppe erledigen Sie auf die gleiche Weise wie das Anlegen eines neuen Benutzers. Im Gegensatz zu Benutzern wird bei Benutzergruppen kein Passwort benötigt. Es ist also lediglich die Angabe eines Namens (*Name*-Eigenschaft) und einer PID (*PID*-Eigenschaft) erforderlich. Das folgende Beispiel legt eine neue Gruppe mit dem Namen *PowerUsers* an:

```
Sub STestprozedur8()  
  
Dim WS As Workspace  
Dim NeueBenutzergruppe As Group  
  
Set WS = DBEngine(0)  
  
Set NeueBenutzergruppe = New Group  
  
    NeueBenutzergruppe.Name = "PowerUsers"  
    NeueBenutzergruppe.PID = "ER123"  
  
WS.Groups.Append NeueBenutzergruppe  
  
End Sub
```

Sollte eine Gruppe oder ein Benutzer mit dem Namen *PowerUsers* bereits vorhanden sein, löst Access einen abfangbaren Laufzeitfehler mit der Nummer 3390 aus.



Hinweis

Beachten Sie, dass alle Gruppen- und Benutzernamen zusammen eindeutig sein müssen. Sie dürfen also z. B. keine Gruppe unter einem Namen anlegen, den bereits ein Benutzer trägt!

Benutzer und Benutzergruppen löschen

Delete-Methode

Zum Entfernen eines Benutzers oder einer Gruppe wenden Sie die *Delete*-Methode an. Diese Methode darf nicht mit der *Delete*-Methode zum Löschen von Datensätzen aus *Recordset*-Objekten verwechselt werden! Die Methode hat folgende Syntax:



```
<Auflistung>.Delete <Objektname>
```

Die *Delete*-Methode entfernt ein Objekt aus der angegebenen Auflistung. Hierbei muss kein Verweis auf das Objekt, sondern lediglich der Name des Objekts angegeben werden. Das folgende Beispiel löscht den Benutzer *Willi Müller*, indem das betreffende *User*-Objekt aus der *Users*-Auflistung des *Workspace*-Objekts entfernt wird.

```
Sub STestprozedur9()  
  
Dim WS As Workspace  
  
    Set WS = DBEngine(0)  
  
    WS.Users.Delete "Willi Müller"  
  
End Sub
```

Sollte der Benutzer nicht vorhanden sein, tritt ein Laufzeitfehler mit der Nummer 3265 auf.

Beachten Sie, dass die von Access vorgegebenen Benutzer und Gruppen nicht gelöscht werden können!



Hinweis

Zuordnung von Benutzern und Gruppen

Wie bereits zu Beginn dieses Kapitels erwähnt, kann ein Benutzer einer oder mehreren Benutzergruppen zugewiesen werden. Dieser Benutzer erhält damit automatisch die Zugriffsrechte der betreffenden Gruppen.

Innerhalb von VBA erfolgt diese Zuweisung mithilfe der *Users*-Auflistung der betreffenden Benutzergruppe. Diese Auflistung darf nicht mit der *Users*-Auflistung verwechselt werden, die *Workspace*-Objekte zur Verfügung stellen. Obwohl beide Auflistungen den gleichen Namen haben, unterscheiden sie sich aufgrund ihrer Einordnung in die Objekthierarchie. Die *Users*-Auflistung des *Workspaces*-Objekts enthält *alle* Benutzer und ist damit die Hauptauffassung für alle Benutzer. Ein neuer Benutzer muss immer zuerst dieser Auflistung hinzugefügt werden. Die *Users*-Auflistung des *Group*-Objekts enthält dagegen nur die Benutzer der betreffenden Gruppe.

Der Benutzer *Admin* ist beispielsweise Mitglied der Gruppen *Admins* und *Users*. Für diesen Benutzer existiert also ein *User*-Objekt in der *Users*-Auflistung des *Workspace*-Objekts, ein weiteres *User*-Objekt in der *Users*-Auflistung der *Admins*-Benutzergruppe und schließlich ein letztes *User*-Objekt in der *Users*-Auflistung der Benutzergruppe *Users*. Das folgende Beispiel listet alle Benutzer der Gruppe *Users* auf:

```
Sub STestprozedur10()  
  
Dim WS As Workspace  
Dim Benutzergruppe As Group
```



Administrator





```
Dim Benutzer As User

Set WS = DBEngine(0)

Set Benutzergruppe = WS.Groups!Admins

For Each Benutzer In Benutzergruppe.Users

    Debug.Print Benutzer.Name

Next Benutzer

End Sub
```

Benutzer einer
Gruppe zuordnen

Wenn Sie einer bestimmten Gruppe einen Benutzer hinzufügen möchten, fügen Sie der *Users*-Auflistung des betreffenden *Group*-Objekts einfach ein neues *User*-Objekt hinzu. Der Name dieses *User*-Objekts muss hierbei dem Benutzer entsprechen, den Sie hinzufügen möchten. Die Angabe weiterer Eigenschaften ist nicht erforderlich. Das folgende Beispiel macht den Benutzer *Willi Müller* zum Mitglied der Gruppe *Admins*:

```
Sub STestprozedur11()

Dim WS As Workspace
Dim EinBenutzer As User

Set WS = DBEngine(0)

Set EinBenutzer = New User

EinBenutzer.Name = "Willi Müller"

WS.Groups!Admins.Users.Append EinBenutzer

End Sub
```

Hinweis



Entfernen eines
Benutzers aus einer
Gruppe

Sollte der Benutzer bereits Mitglied der Gruppe sein, tritt ein Laufzeitfehler mit der Nummer 3032 auf.

Zum Entfernen eines Benutzers aus einer Gruppe, löschen Sie diesen einfach mithilfe der *Delete*-Methode aus der *Users*-Auflistung der betreffenden Gruppe. Das folgende Beispiel entfernt den Benutzer *Willi Müller* aus der Gruppe *Admins*:

```
Sub STestprozedur12()

Dim WS As Workspace

Set WS = DBEngine(0)
```



```
WS.Groups!Admins.Users.Delete "Willi Müller"
```

End Sub

Wenn Sie einen Benutzer vollständig – also aus der *Users*-Auflistung des *Workspace*-Objekts – löschen, entfernt Access den Benutzer automatisch auch aus allen Benutzergruppen.



Hinweis

Wie Sie der Objekthierarchie von Access entnehmen können, besitzen *User*-Objekte ebenfalls eine *Groups*-Auflistung. Über diese lässt sich auf einfache Weise ermitteln, welchen Gruppen ein bestimmter Benutzer angehört. Das folgende Beispiel listet alle Gruppen im Testfenster auf, bei denen der Benutzer *Admin* Mitglied ist:



**Die *Groups*-
Auflistung von
User-Objekten**

```
Sub STestprozedur13()
```

```
Dim Gruppe As GROUP
```

```
For Each Gruppe In DBEngine(0).Users!Admin.Groups
```

```
    Debug.Print Gruppe.Name
```

```
Next Gruppe
```

End Sub

Um einer Benutzergruppe einen neuen Benutzer hinzuzufügen, haben Sie bereits eine Möglichkeit kennen gelernt. Die zweite Möglichkeit besteht nun darin, dass Sie der *Groups*-Auflistung eines Benutzers ein neues *Group*-Objekt hinzufügen. Der Name dieses *Group*-Objekts muss mit dem Namen einer vorhandenen Benutzergruppe übereinstimmen. Das folgende Beispiel nutzt diesen Weg, um den Benutzer *Willi Müller* der Gruppe *Users* hinzuzufügen:

```
Sub STestprozedur14()
```

```
Dim WS As Workspace
```

```
Dim EineGruppe As GROUP
```

```
Set WS = DBEngine(0)
```

```
Set EineGruppe = New GROUP
```

```
EineGruppe.Name = "Users"
```

```
WS.Users![Willi Müller].Groups.Append EineGruppe
```

End Sub



Hinweis



Access achtet darauf, dass die Zuweisung von Benutzern zu Benutzergruppen immer konsistent ist. Wenn Sie also der *Groups*-Auflistung eines Benutzers ein neues *Group*-Objekt hinzufügen, fügt Access automatisch der *Users*-Auflistung des betreffenden *Group*-Objekts einen neuen Benutzer hinzu.

Container- und Document-Objekte

Bis jetzt haben Sie die Verwaltung von Benutzern und Benutzergruppen kennen gelernt. Diese Informationen gelten für eine komplette Arbeitsgruppe und werden zusammen mit der Systemdatenbank *system.mdw* gespeichert.

Datenbank schützen

Um eine Datenbank zu schützen, muss für jedes der Objekte innerhalb einer Datenbank definiert werden, welcher Benutzer welche Zugriffsrechte haben soll. Den Zugriff auf die einzelnen Objekte der Datenbank erlaubt die *Containers*-Auflistung. In dieser Auflistung sind die folgenden neun Objekte vorhanden:

Inhalte der Containers-Auflistung

Container	Beschreibung
<i>DataAccessPages</i>	Enthält alle Datenzugriffsseiten der aktuellen Datenbank.
<i>Databases</i>	Enthält Informationen zur aktuellen Datenbank.
<i>Forms</i>	Enthält alle Formulare der aktuellen Datenbank.
<i>Modules</i>	Enthält alle Module der aktuellen Datenbank.
<i>Relationships</i>	Enthält alle Verknüpfungen zwischen Tabellen der aktuellen Datenbank. Dieser Container ist für das Sicherheitssystem nicht relevant.
<i>Reports</i>	Enthält alle Berichte der aktuellen Datenbank.
<i>Scripts</i>	Enthält alle Makros der aktuellen Datenbank.
<i>SysRel</i>	Wird Access-intern benötigt, um das Layout des <i>Beziehungen</i> -Dialogfensters zu speichern. Dieser Container ist für das Sicherheitssystem nicht relevant.
<i>Tables</i>	Enthält alle Tabellen der aktuellen Datenbank.

Tabelle 12.1



Die Container sind vergleichbar mit den einzelnen Kategorien des Datenbankfensters. Der Zugriff auf die Container erfolgt in der für Auflistungen typischen Weise. Die folgende Prozedur gibt eine Liste der Containernamen im Testfenster aus:

```
Sub STestprozedur15()  
  
Dim DB As Database  
Dim ContainerObjekt As Container  
  
    Set DB = CurrentDb  
  
    For Each ContainerObjekt In DB.Containers  
  
        Debug.Print ContainerObjekt.Name  
  
    Next ContainerObjekt  
  
End Sub
```

Innerhalb der einzelnen Container befinden sich *Document*-Objekte, die die einzelnen Datenbankobjekte repräsentieren. Wenn Ihre Datenbank beispielsweise fünf Formulare enthält, befinden sich im Container mit dem Namen *Forms* fünf *Document*-Objekte, die die betreffenden Formulare repräsentieren. Die Namen der *Document*-Objekte stimmen hierbei mit den Formularnamen überein. Das folgende Beispiel listet die Objekte des *Forms*-Containers im Testfenster auf:

```
Sub STestprozedur16()  
  
Dim DB As Database  
Dim DocumentObjekt As Document  
  
    Set DB = CurrentDb  
  
    For Each DocumentObjekt In _  
        DB.Containers!Forms.Documents  
  
        Debug.Print DocumentObjekt.Name  
  
    Next DocumentObjekt  
  
End Sub
```

Der Container *Databases* enthält nur ein einziges *Document*-Objekt, das die aktuelle Datenbank repräsentiert.

Sowohl der *Containers*-Auflistung als auch den darin enthaltenen *Documents*-Auflistungen können Sie direkt keine Objekte hinzufügen. Access verwaltet diese Auflistungen automatisch, wenn Sie vom Datenbankfenster aus Objekte hinzufügen oder löschen. Bei der Aktualität der *Documents*-Auflistungen hinkt Access jedoch oft deutlich hinter dem Datenbankfenster hinterher. Vor dem Zugriff auf die *Docu-*

Document-Objekte

Document-Auflistung aktualisieren





ments-Auflistung sollten Sie daher mithilfe der *Refresh*-Methode stets eine Aktualisierung durchführen. Der folgende Programmausschnitt erledigt dies am Beispiel des Containers *Tables*:

```
Dim DB As Database
Dim Tabellen As Container

Set DB = CurrentDb
Set Tabellen = DB.Containers!Tables
Tabelle.Documents.Refresh
```

Hinweis



Mithilfe der Eigenschaften *DateCreated* und *LastUpdated* von *Document*-Objekten können Sie herausfinden, wann ein Objekt angelegt bzw. die Struktur zuletzt geändert wurde. Informationen dazu entnehmen Sie bitte der auf der CD zum Buch enthaltenen Referenz.

Erteilen von Zugriffsberechtigungen

Access erlaubt es, den Zugriff auf einzelne Dokumente – also z.B. Formulare oder Tabellen – oder auf Container und damit alle enthaltenen Dokumente festzulegen. Die letzte Variante hat den Vorteil, dass ein neu hinzugefügtes Objekt automatisch die gleichen Zugriffsbeschränkungen wie das betreffende *Container*-Objekt erhält.

UserName-Eigenschaft



Zur Steuerung des Zugriffs besitzen alle *Container*- und *Document*-Objekte zwei Eigenschaften mit den Namen *UserName* und *Permissions*. Mithilfe der *UserName*-Eigenschaft des gewünschten Objekts legen Sie fest, für welchen Benutzer oder welche Benutzergruppe Sie die Zugriffsrechte auf das Objekt ermitteln oder verändern möchten.

Wenn Sie beispielsweise die Zugriffsrechte der Tabelle *Kunden* für den Benutzer *Willi Müller* erkunden möchten, weisen Sie der *UserName*-Eigenschaft des *Document*-Objekts *Kunden* den Text *Willi Müller* zu. Erst wenn Sie dies erledigt haben, enthält die *Permissions*-Eigenschaft die passenden Zugriffsrechte.

Neben dem Namen eines Benutzers können Sie der *UserName*-Eigenschaft auch den Namen einer Benutzergruppe zuweisen, um so die Zugriffsrechte der Gruppe bezüglich des Objekts zu ermitteln oder zu verändern.

Das folgende Beispiel geht davon aus, dass Ihre Datenbank eine Tabelle mit dem Namen *Kunden* enthält und Sie einen Benutzer mit dem Namen *Willi Müller* angelegt haben. Die Prozedur ermittelt nun für diesen Benutzer die Zugriffsrechte:

```
Sub STestprozedur17()

Const Benutzername = "Willi Müller"

Dim DB As Database
Dim Kundentabelle As Document
```



Dim Tabellen As Container

```
Set DB = CurrentDb
```

```
Set Tabellen = DB.Containers!Tables  
Tabelle.Documents.Refresh
```

```
Set Kundentabelle = Tabellen.Documents!Kunden
```

```
Kundentabelle.UserName = Benutzername
```

```
Debug.Print Kundentabelle.Permissions
```

End Sub

Die Zugriffsrechte werden in der *Permissions*-Eigenschaft als numerischer Wert abgelegt. Jeder Wert repräsentiert hierbei eine bestimmte Zugriffsberechtigung. Der Wert der Konstanten *dbSecInsertData* bedeutet beispielsweise, dass Daten hinzugefügt werden dürfen. Ein zweiter Wert, *dbSecRetrieveData*, bestimmt dagegen, ob das Abfragen von Daten zulässig ist. Eine komplette Liste aller Werte finden Sie in der auf der CD zum Buch enthaltenen Referenz unter der *Permissions*-Eigenschaft.

○ —————
Permissions-
Eigenschaft

Um eine bestimmte Zugriffsberechtigung zuzuweisen, wird in der Regel die Summe mehrerer Werte verwendet. In diesem Fall müssen die entsprechenden Konstanten addiert werden. Um beispielsweise uneingeschränkten Datenzugriff auf eine Tabelle zu ermöglichen, die Struktur der Tabelle aber gegen Veränderungen zu schützen, geben Sie die Summe der Konstanten *dbSecRetrieveData*, *dbSecInsertData*, *dbSecDeleteData* und *dbSecReplaceData* an. Das folgende Beispiel sichert auf diese Weise die Tabelle *Kunden*:

○ —————
Zugriffsrechte
kombinieren

```
Sub STestprozedur18()
```

```
Const Benutzername = "Willi Müller"
```

```
Dim DB As Database
```

```
Dim Kundentabelle As Document
```

```
Dim Tabellen As Container
```

```
Set DB = CurrentDb
```

```
Set Tabellen = DB.Containers!Tables  
Tabelle.Documents.Refresh
```

```
Set Kundentabelle = Tabellen.Documents!Kunden
```

```
Kundentabelle.UserName = Benutzername
```





```
Kundentabelle.Permissions = dbSecRetrieveData _  
    + dbSecInsertData _  
    + dbSecDeleteData _  
    + dbSecReplaceData
```

End Sub

 Zugriffsbeschränkungen von Objekten ermitteln

Auf dem umgekehrten Weg ist es ebenfalls möglich, zu ermitteln, ob ein bestimmtes Objekt einer bestimmten Zugriffsbeschränkung unterliegt. Der folgende Ausdruck besitzt beispielsweise den logischen Wert *True*, wenn Daten aus der betreffenden Tabelle gelöscht werden können:

```
(Kundentabelle.Permissions And dbSecDeleteData) <> 0
```

Hinweis



Ein einfacher Vergleich des Inhalts der *Permissions*-Eigenschaft mit einer der angegebenen Konstanten ist nicht möglich, da die Eigenschaft eine Summe aus mehreren Konstanten enthalten kann.

Das folgende Beispiel untersucht, ob der Benutzer *Willi Müller* aus der Tabelle *Kunden* Datensätze löschen darf:

```
Sub STestprozedur19()  
  
    Const Benutzername = "Willi Müller"  
  
    Dim DB As Database  
    Dim Kundentabelle As Document  
    Dim Tabellen As Container  
  
    Set DB = CurrentDb  
  
    Set Tabellen = DB.Containers!Tables  
    Tabellen.Documents.Refresh  
  
    Set Kundentabelle = Tabellen.Documents!Kunden  
  
    Kundentabelle.UserName = Benutzername  
  
    If (Kundentabelle.Permissions And dbSecDeleteData) _  
        <> 0 Then  
  
        MsgBox "Der Benutzer darf Daten aus der _  
            Tabelle löschen!"  
  
    Else
```





```
MsgBox "Der Benutzer darf keine Daten " & _  
      "aus der Tabelle löschen!"
```

```
End If
```

```
End Sub
```

Die Berechtigungen der *Permission*-Eigenschaft beziehen sich immer nur auf den betreffenden Benutzer. Wenn ein Benutzer in eine bestimmte Benutzergruppe eingeordnet ist, erhält er eventuell zusätzliche Rechte, die jedoch über die *Permissions*-Eigenschaft nicht berücksichtigt werden. Aus diesem Grund stellt Access die *AllPermissions*-Eigenschaft zur Verfügung, die neben den Rechten des Benutzers auch die Gruppenzugehörigkeit berücksichtigt. Die Eigenschaft ermöglicht also, die kompletten Rechte des Benutzers zu ermitteln. Nähere Informationen dazu entnehmen Sie bitte der auf der CD zum Buch enthaltenen Referenz.

Wenn Sie die komplette Datenbank schützen möchten, verweisen Sie auf das *Document* des Containers *Databases*. Das hier mit der Positionsnummer 0 enthaltene Objekt repräsentiert die aktuelle Datenbank. Wenn Sie der *Permissions*-Eigenschaft dieses Objekts beispielsweise die Konstante *dbSecNoAccess* zuweisen, ist die komplette Datenbank für den ausgewählten Benutzer gesperrt und kann nicht mehr geöffnet werden.



Hinweis

Neben *Document*-Objekten kann die *Permissions*-Eigenschaft auch bei *Container*-Objekten verwendet werden. Die hier getroffenen Einstellungen wirken sich jedoch nur auf neu erstellte Objekte aus. Damit dies funktioniert, muss die *Inherit*-Eigenschaft des *Container*-Objekts auf *True* eingestellt sein. Bei einem neuen Objekt kopiert Access die Zugriffsrechte des Containers auf das neue *Document*-Objekt.



**Zugriffsrechte von
Containern**

Eigentümerrechte

Besonders wichtig im Zusammenhang mit dem Sichern einer Datenbank sind die Eigentümerrechte der einzelnen Datenbankobjekte. Als Eigentümer (*Owner*) eines Objekts wird immer derjenige Benutzer bezeichnet, der das Objekt angelegt hat. Der Eigentümer hat immer alle Zugriffsrechte auf ein Objekt.

Sie können den Eigentümer eines Objekts ermitteln, indem Sie die *Owner*-Eigenschaft des betreffenden *Document*-Objekts auslesen. Die folgende Prozedur gibt den Eigentümer der Tabelle *Kunden* in einem Meldungsfenster aus:

```
Sub STestprozedur20()
```

```
Dim DB As Database
```

```
Set DB = CurrentDb
```

```
DB.Containers!Tables.Documents.Refresh
```





```
MsgBox DB.Containers!Tables.Documents!Kunden.Owner
```

End Sub

Standardmäßig wird als Eigentümer eines Objekts immer der Benutzer *Admin* eingetragen. Wenn Sie allerdings das Sicherheitssystem aktiviert und sich unter einem anderem Benutzernamen als *Admin* angemeldet haben, trägt Access beim Anlegen neuer Objekte diesen Benutzernamen als Eigentümer ein.

Das Ändern des Eigentümers ist ebenfalls möglich. Dazu weisen Sie der *Owner*-Eigenschaft einfach einen anderen Benutzernamen zu.

Hinweis



Auch eine komplette Datenbank besitzt einen Eigentümer, der mit dem Benutzer übereinstimmt, welcher beim Anlegen der Datenbank angemeldet war. Wenn das Sicherheitssystem also deaktiviert ist, ist der Eigentümer neuer Datenbanken stets der Benutzer *Admin*.

Der Eigentümer einer Datenbank lässt sich leider nicht ohne weiteres ändern. Die einzige Möglichkeit, dies zu erledigen, besteht darin, sich unter dem gewünschten Benutzernamen anzumelden, eine neue Datenbank anzulegen und anschließend alle Objekte aus der Datenbank in die neue Datenbank zu importieren. Genau diese Aufgabe erledigt der Sicherheitsassistent, den Sie über den Menüpunkt *Extras > Sicherheit > Benutzer-Datensicherheits-Assistent* starten.

Sichern einer Datenbank

Besonders wichtig für den Datenbankentwickler ist der Schutz des Programmcodes und aller übrigen Datenbankobjekte vor neugierigen Blicken anderer Benutzer.

Sicherheitssystem aktivieren



Ein vollständiger Schutz, der auch nach einem Wechsel der aktuellen Arbeitsgruppe noch aktiv ist, kann nur dann gewährleistet werden, wenn der Eigentümer der Datenbank nicht der Benutzer *Admin* ist. Wenn Sie also die Entwicklung einer geschützten Datenbank planen, aktivieren Sie zuerst das Sicherheitssystem, indem Sie dem Benutzer *Admin* (bzw. *Administrator*) ein Passwort zuweisen. Anschließend legen Sie unter Ihrem Namen in der Benutzergruppe *Administratoren* einen neuen Benutzer an.

Nachdem Sie dies erledigt haben, starten Sie Access neu und melden sich unter dem neuen Benutzernamen an. Wenn Sie nun über die Menüfunktionen von Access oder aber auch von VBA aus eine neue Datenbank anlegen, erhalten Sie automatisch die Eigentümerrechte für diese Datenbank.

Eigentümer ändern



Wenn Sie eine bereits vorhandene Datenbank schützen möchten, müssen Sie ebenfalls die Eigentumsrechte übernehmen. Hierbei gehen Sie wie oben beschrieben vor und legen eine neue leere Datenbank an. Anschließend importieren Sie alle Objekte der zu schützende Datenbank über den Menüpunkt *Datei > Externe Daten > Importieren*.



Im jetzigen Stadium ist Ihre Datenbank nur scheinbar vollständig geschützt. Wenn Sie ihre "gesicherte" Datenbank nun weitergeben, wird eine andere Arbeitsgruppe aktiv. In dieser befinden sich der vordefinierte Benutzer *Admin* sowie die vordefinierten Gruppen *Admins* und *Users*. Diese besitzen in der Regel keine Zugriffsbeschränkungen, so dass der Zugriff auf Ihre geschützten Objekte wieder möglich ist.

Zugriffsrechte des Administrators entziehen

Um dieses Problem zu lösen, müssen Sie dem Benutzer *Admin* alle gewünschten Zugriffsrechte für die betreffenden Objekte entziehen. Wenn Sie überhaupt keinen Zugriff zulassen möchten, entziehen Sie dem Administrator einfach alle Rechte. Das Gleiche gilt für die Benutzergruppen *Admins* (deutsch *Administratoren*) und *Users* (deutsch *Benutzer*). Erst jetzt sind Ihre Datenbankobjekte vollständig geschützt.

Das folgende Beispiel legt eine neue Datenbank unter dem Namen *C:\SICHER.MDB* an und exportiert die Tabelle *Kunden*. Anschließend werden die Benutzerrechte des Administrators und der beiden Gruppen zum Verändern der Tabellenstruktur der Tabelle *Kunden* entzogen. Die Sicherung funktioniert nur, wenn Sie unter Ihrem Namen und nicht als Administrator angemeldet sind! Die Prozedur überprüft dies und bricht ggf. ab.

```
Sub KundentabelleSichern()

Const NeueDB_Name = "C:\SICHER.MDB"
Const Zugriffsberechtigung = dbSecRetrieveData +
dbSecInsertData _
+
dbSecDeleteData + dbSecReplaceData

Dim NeueDB As Database
Dim Kunden As Document

On Error GoTo Err

If CurrentUser() = "admin" Then
    Beep
    MsgBox "Sie dürfen nicht als Administrator _
angemeldet sein!"
    Exit Sub
End If

Set NeueDB = DBEngine(0).CreateDatabase(NeueDB_Name, _
dbLangGeneral)

DoCmd.TransferDatabase acExport, "Microsoft Access", _
NeueDB_Name, acTable,
"Kunden", "Kunden", False

NeueDB.Containers!Tables.Documents.Refresh
Set Kunden = NeueDB.Containers!Tables.Documents!Kunden
```





```
Kunden.UserName = "admin"
Kunden.Permissions = Zugriffsberechtigung

Kunden.UserName = "Admins"
Kunden.Permissions = Zugriffsberechtigung

Kunden.UserName = "Users"
Kunden.Permissions = Zugriffsberechtigung

NeueDB.Close

Beep
MsgBox "Die Tabelle "Kunden" der Datenbank "" & _
      NeueDB_Name & "" ist nun gesichert!"

Exit Sub

Err:

Beep
MsgBox Error (Err)
Exit Sub

End Sub
```

Wenn Sie sich nun unter einem anderen Namen anmelden und die Datenbank *C:\SICHER.MDB* öffnen, können Sie zwar auf die Daten der Tabelle *Kunden* zugreifen und die Struktur begutachten; das Ändern der Struktur ist aber nicht möglich.

Hinweis



Wenn Sie Ihre komplette Datenbank schützen möchten, weisen Sie einfach dem Datenbankdokument des *Databases*-Containers die Berechtigung *dbSecNoAccess* zu. Dies muss ebenfalls für den Administrator und die beiden Benutzergruppen *Admins* und *Users* erfolgen. Die Datenbank kann nun nur noch von berechtigten Personen geöffnet werden.

Das folgende Beispiel zeigt noch einmal, wie dem Administrator der Zugriff auf eine Datenbank (in diesem Fall die aktuelle Datenbank) verwehrt wird:

```
Sub STestprozedur21()

Dim DB As Database
Dim Datenbank As Document

Set DB = CurrentDb

Set Datenbank = DB.Containers!Databases.Documents(0)

Datenbank.UserName = "admin"
```





```
Datenbank.Permissions = dbSecNoAccess
```

End Sub

Noch besseren Schutz – besonders bei der Sicherung von Daten aus Tabellen – erreichen Sie, indem Sie die Datenbank verschlüsseln. Dazu geben Sie z.B. beim Anlegen der Datenbank über die *CreateDatabase*-Methode als dritten Parameter die Konstante *dbEncrypt* an. Die Datenbank wird nun verschlüsselt auf Ihrer Festplatte abgelegt, so dass selbst mit einem Dateieditor keine Daten mehr gelesen werden können.



Hinweis





OLE und DDE



OLE – Verknüpfen und Einbetten	628
Automation	654
ActiveX-Steuerelemente	667
DDE	673





Access stellt mit seinem enorm großen Funktionsumfang ein leistungsfähiges Werkzeug zur Entwicklung ausgefeilter Datenbank Anwendungen dar. Doch von Zeit zu Zeit lässt auch Access die eine oder andere Funktion schmerzlich vermissen. Ein gutes Beispiel hierfür ist die fehlende Fähigkeit von Access, formatierte Texte zu speichern. Selbst einfache Formatierungen wie fette oder unterstrichene Textpassagen lassen sich nicht in ein einzelnes Textfeld eingeben. Die Entwickler von Access hätten nun die Textfelder von Access um solche einfachen Funktionen erweitern können. Doch was ist mit den Anwendern, die ihre Texte aufwändiger – beispielsweise mit Tabellen – gestalten möchten? Um auch diese Bedürfnisse zu befriedigen, hätte Microsoft in Access nahezu den kompletten Funktionsumfang der Textverarbeitung Word für Windows einbauen müssen.

Doch zum Glück gibt es *OLE* (**O**bject **L**inking and **E**mbadding, übersetzt Objektverknüpfung und -einbettung). Damit lassen sich kurz gesagt fast alle Funktionen anderer OLE-fähiger Windows-Applikationen – auch die von WinWord – von Access aus nutzen. Die Integration geht sogar so weit, dass Sie alle Befehle von WinWord XP von Access aus aufrufen können.

In diesem Kapitel geht es darum, wie Sie OLE von Access aus optimal einsetzen. Die Beispiele beziehen sich auf die Anwendungen WinWord und Excel, die Bestandteile des *Microsoft Office-Pack-Professionell* sowie der Premium-Edition sind.

Auch OCX-Steuerelemente, jetzt *ActiveX-Steuerelemente* genannt, kommen in diesem Kapitel nicht zu kurz. Hierbei handelt es sich um einfache OLE-Server, mit denen die Palette der Steuerelemente von Access erweitert werden kann. Sie lernen den grundlegenden Umgang mit diesen interessanten Steuerelementen kennen.

Neben OLE geht es in diesem Kapitel auch um den dynamischen Datenaustausch (kurz *DDE*, **D**ynamic **D**ata **E**xchange). Diese Art der Kommunikation mit anderen Windows-Anwendungen ist inzwischen zwar durch OLE technisch überholt, findet aber in der Praxis bei nicht OLE-fähigen Applikationen weiterhin Verwendung.

OLE – Verknüpfen und Einbetten

OLE ist eine Abkürzung für *Object Linking and Embedding*, was soviel wie *Objekt verknüpfen und einbetten* bedeutet. Als Objekt wird hierbei beispielsweise ein Text bezeichnet, den Sie mit der Textverarbeitung *WinWord* geschrieben haben. Das Word-Dokument wird allerdings erst dann zu einem Objekt, wenn Sie diesen Text in eine andere Windows-Anwendung einbetten oder es mit ihr verknüpfen. Diese auf den ersten Blick etwas uneinsichtige Tatsache basiert auf dem Grundgedanken von OLE. Um diesen zu verstehen, ist ein Rückblick zur Windows-Version 3.0 erforderlich, in der OLE noch nicht zur Verfügung stand.

Um mit dieser alten Windows-Version beispielsweise eine Excel-Tabelle in ein Word-Dokument einzufügen, können Sie sich der Zwischenablage bedienen. Dies bedeutet, dass Sie den gewünschten Zellbereich in Excel markieren, ihn mittels *Bearbeiten > Kopieren* in die Zwischenablage einfügen und anschließend per *Bearbeiten > Einfügen* wieder aus der Zwischenablage in das WinWord-Dokument ein-



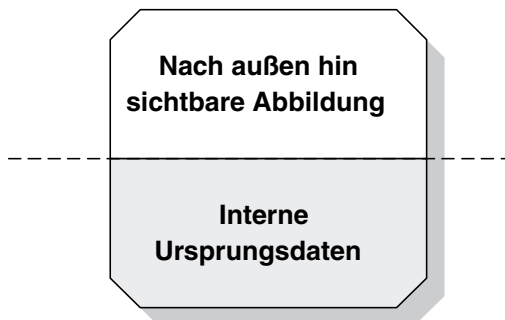
fügen. Das große Manko bei dieser Vorgehensweise ist allerdings, dass die Excel-Tabelle in dem Textdokument nur als Grafik oder Abbild abgelegt wurde. Wenn überhaupt, dann können Sie diese Grafik nachträglich nur noch mit einem Zeichenprogramm bearbeiten. Wenn Sie also planen, später noch Änderungen an der Tabelle vorzunehmen, müssen Sie sie innerhalb von Excel als normale Excel-Tabelle auf der Festplatte speichern. Zu jeder Excel-Tabelle, die Sie in ein Word-Dokument einfügen, müssen Sie also zusätzlich eine Excel-Datei auf Ihrer Festplatte anlegen.

Wenn Sie nun eine Tabelle Ihres Word-Dokuments nachträglich bearbeiten möchten, müssen Sie zunächst die passende Excel-Datei ausfindig machen, diese laden, die Änderungen vornehmen und diese anschließend – anstelle der alten Tabelle – wieder in das Word-Dokument einfügen. Alles in allem eine sehr umständliche Aufgabe.

Mit Windows 3.1 hat Microsoft OLE in der Version 1.0 eingeführt, das diesen Nachteil weitestgehend behebt. Wenn Sie hier eine Excel-Tabelle in ein Word-Dokument einfügen, speichert Word – für den Anwender unsichtbar – neben dem reinen Abbild der Tabelle auch die ursprünglichen Excel-Daten der Tabelle. Es ist also nicht mehr notwendig, die Daten extra noch einmal auf der Festplatte zu speichern, da nun Word diese Aufgabe übernimmt. Ein eher unerwünschter Nebeneffekt dieser Tatsache ist allerdings, dass der Umfang des Word-Dokuments in einigen Fällen erheblich anwächst.

Wenn die Tabelle nun nachträglich bearbeitet werden soll, genügt ein Doppelklick auf deren Abbild. Windows startet daraufhin automatisch Excel und übergibt die Excel-Daten von Word an Excel. Die Tabelle wird hier wie gewohnt dargestellt und kann bearbeitet werden.

Da in dem Word-Dokument nun nicht nur das Abbild der Tabelle, sondern auch die ursprünglichen Excel-Daten gespeichert werden, wird diese Tabelle auch als *Objekt* bezeichnet, das in das Word-Dokument eingebettet ist. Windows geht sogar so weit, dass das Objekt nicht nur aus zwei, sondern aus mehreren Komponenten bestehen kann. Neben den Daten im ursprünglichen Format sind zusätzlich weitere Präsentationsformate enthalten, aus denen sich eine Windows-Anwendung wie Word das passende Format zur Darstellung auswählen kann.



○ Abb. 13.1

Der Aufbau eines OLE-Objekts





Neben dem Einbetten von Objekten stellt OLE auch die Möglichkeit zum Verknüpfen von Objekten zur Verfügung. Verknüpfungen sind beispielsweise dann erforderlich, wenn mehrere Personen eine Tabelle in verschiedene Word-Dokumente einfügen möchten. Wird die Tabelle nachträglich bearbeitet, wirken sich die Änderungen auf alle Word-Dokumente aus, mit denen die Tabelle verknüpft wurde.

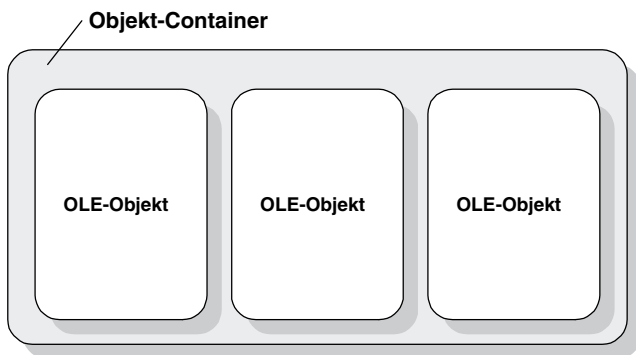
Zum Anlegen einer Verknüpfung gehen Sie genauso wie beim Einbetten eines Objekts vor. Anstelle des Menüpunkts *Einfügen* wählen Sie allerdings den Menüpunkt *Inhalte einfügen* aus und markieren die Option *Verknüpfen*. Bitte beachten Sie, dass sich diese Menüpunkte auf die Textverarbeitung *Word für Windows* beziehen. Bei anderen Windows-Anwendungen kann die Vorgehensweise zum Anlegen einer Verknüpfung unterschiedlich sein.

OLE-Server und
-Container

Im Zusammenhang mit OLE wird oft von *Objektservern* und *Objekt-Clients* bzw. *Objektcontainern* gesprochen. Ein Objektserver ist eine Windows-Anwendung, die einer anderen Windows-Anwendung ihre Dienste zur Verfügung stellt. In dem vorgestellten Beispiel wäre also Excel der OLE-Server. Word arbeitet dagegen als Objekt-Client (oder Objektcontainer).

Abb. 13.2

Objektcontainer und
Objektserver



Das Gespann aus Word und Excel kann dagegen auch die Rollen vertauschen, indem ein Word-Dokument oder ein Teil eines Word-Dokuments in eine Excel-Tabelle eingefügt wird. In diesem Fall verhält sich Excel als Client und Word als Server.

Nach diesen eher theoretischen Vorüberlegungen geht es nun im nächsten Abschnitt um die praktische Anwendung des Einbettens und Verknüpfens von Objekten mit Access.

Einfügen von Objekten

OLE wird neben Word und Excel natürlich auch von Access zur Verfügung gestellt. Allerdings gibt es eine Einschränkung: Access kann in Bezug auf das Einbetten und Verknüpfen nur als Objekt-Client und nicht als Server arbeiten. In der Praxis bedeutet dies, dass Sie zwar beliebige Objekte in Access einfügen und speichern können – Sie können dagegen keine Access-Objekte in andere Windows-Anwendungen wie



z. B. Word oder Excel einfügen. Obwohl dies in einigen Fällen sinnvoll wäre (man denke nur an die Möglichkeit, ein voll funktionsfähiges Access-Formular in ein Win-Word-Dokument einzufügen), lassen sich die entsprechenden Problemstellungen oft auch auf andere Weise lösen.

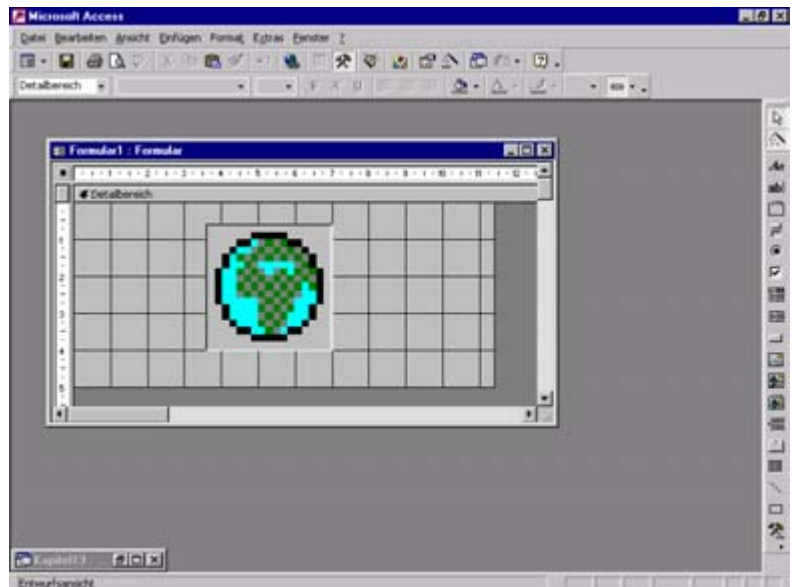
Prinzipiell gibt es drei verschiedene Möglichkeiten, ein OLE-Objekt in Access einzufügen. Der einfachste Weg führt über die Zwischenablage. Um beispielsweise eine Grafik einzufügen, die Sie mithilfe des zum Lieferumfang von Windows gehörenden Programms *Paint* erstellt haben, starten Sie das Zeichenprogramm aus dem Menü *Start > Programme > Zubehör* von Windows. Hier können Sie nun entweder etwas zeichnen oder eine fertige Grafik laden. Anschließend markieren Sie den gewünschten Teil der Grafik und kopieren ihn in die Zwischenablage. Nun wechseln Sie zu Access, öffnen ein neues Formular und wählen den Menüpunkt *Bearbeiten > Einfügen* aus. Kurze Zeit später erscheint das Objekt.

Der Vorteil dieser Vorgehensweise ist, dass Sie aus bestehenden Dateien bei Bedarf nur einen Ausschnitt als OLE-Objekt einbetten können. Auf die gleiche Weise können Sie natürlich auch ein Objekt in einen Bericht einfügen. Da Access in diesem Punkt keine großen Unterschiede zwischen Formularen und Berichten macht, werden im Folgenden nur Formulare beschrieben.

Einfügen eines OLE-Objekts

Abb. 13.3

Ein in Access eingefügtes *Paint*-Objekt



Die zweite Möglichkeit zum Einfügen eines OLE-Objekts ist nur dazu geeignet, neue leere Objekte einzufügen, die Sie selbst mit Inhalt füllen. Dazu stellt Access bei Formularen, die in der Entwurfsansicht geöffnet sind, den Menüpunkt *Einfügen > Objekt* zur Verfügung. Dieser aktiviert ein Dialogfenster, in dem Sie direkt einen

Leeres Objekt einfügen



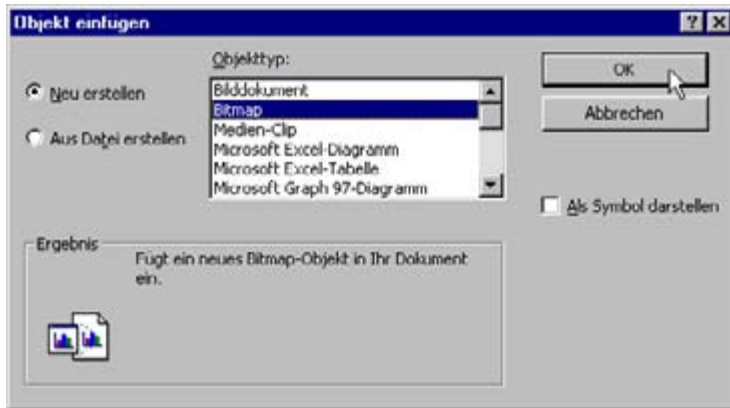
OLE-Server auswählen können. In der Liste der Server finden Sie beispielsweise den Eintrag *Bitmap*. Wenn Sie diesen Eintrag markieren und mit *OK* bestätigen, öffnet Access das Zeichenprogramm mit einer leeren Arbeitsfläche. Sie können nun mit dem Zeichnen beginnen. Ist dies erledigt, schließen Sie Paint oder wählen den Menüpunkt *Beenden und zurück zu ...* aus. Anschließend erscheint die neue Grafik in Access.

Objekt aus einer
Datei erstellen

Die dritte Variante wird dann verwendet, wenn Sie ein Objekt einfügen möchten, das bereits als Datei auf Ihrer Festplatte vorliegt. Dazu öffnen Sie wieder das Dialogfenster *Einfügen > Objekt* und markieren im Dialogfenster *Objekt einfügen* die Option *Aus Datei erstellen*. Es erscheint ein Textfeld, in das Sie nun den Namen der gewünschten Datei eingeben können. Durch einen Klick auf *Durchsuchen* lässt sich der Dateiname auch komfortabler über einen Dateidialog auswählen. Anhand der Dateierweiterung der ausgewählten Datei ermittelt Windows nun den passenden OLE-Server. Wenn Sie beispielsweise die Datei *Wolken.bmp* aus dem Windows-Installationsverzeichnis ausgewählt haben, erkennt Windows an der Endung *.bmp*, dass Sie ein *Bitmap*-Objekt einfügen möchten. Nach der Bestätigung mit *OK* erscheint das neue Objekt in dem Access-Formular.

Abb. 13.4

Das Dialogfenster
Objekt einfügen



Bearbeiten eines
Objekts

Dass Sie es wirklich mit einem waschechten Objekt und nicht etwa einer einfachen Grafik zu tun haben, erkennen Sie schnell an dem Menüpunkt *Bearbeiten > Bitmap-Objekt*. Dieser Menüpunkt ist nur dann vorhanden, wenn Sie im aktuellen Formular ein entsprechendes OLE-Objekt markiert haben. Der Menüpunkt enthält drei weitere Untermenüpunkte. Mit *Bearbeiten* und *Öffnen* können Sie die Grafik nachträglich verändern. Access startet dazu das Zeichenprogramm und zeigt die Grafik zur Bearbeitung an. Zum gleichen Ergebnis gelangen Sie, wenn Sie das Objekt doppelt mit der Maus anklicken. Die Funktion *Bearbeiten* wird auch als *Verb* des Objekts bezeichnet. In einigen Fällen besitzt ein Objekt mehrere Verben. Wenn Sie beispielsweise ein Klangobjekt (Wave-Audio) in Access einfügen, finden Sie hier zusätzlich den Menüpunkt *Wiedergeben*. Sie haben also zwei Möglichkeiten, das Objekt zu aktivieren: Entweder Sie öffnen es zur Bearbeitung oder zur reinen Wiedergabe.



Das erste Verb des Objekts wird auch als *Standard-Verb* bezeichnet. Wenn Sie ein Objekt doppelt anklicken, führt Access stets dieses Standard-Verb aus.



Hinweis

Der dritte Untermenüpunkt des Menüpunkts *Bearbeiten* > *Bitmap-Objekt* lautet *Konvertieren*. Hiermit wandeln Sie das OLE-Objekt in ein anderes Format um. Bei einigen Objekten hat dies den Vorteil, dass Speicherplatz eingespart wird. In der Regel geht hierbei allerdings die Möglichkeit verloren, das Objekt nachträglich mit dem ursprünglichen Objektservers zu bearbeiten.



Umwandlung von OLE-Objekten

Verknüpfen von Objekten

Das Verknüpfen von Objekten ist beispielsweise über die Zwischenablage möglich. Anstelle des Befehls *Bearbeiten* > *Einfügen* wählen Sie den Menüpunkt *Bearbeiten* > *Inhalte einfügen* aus, um ein Objekt in ein Formular einzufügen. Es erscheint das gleichnamige Dialogfenster, in dem Sie die Option *Verknüpfung einfügen* markieren. Mit OK wird die Verknüpfung erstellt und das Objekt aus der Zwischenablage eingefügt. Besonders wichtig beim Erstellen von Verknüpfungen ist, dass Sie das Dokument im OLE-Server vor dem Kopieren gespeichert haben, da Access die Verknüpfung nur zu einer Datei aufrechterhalten kann. Wenn Sie also einen Teil einer Excel-Tabelle verknüpfen möchten, sollten Sie diese auf jeden Fall vorher unter einem vernünftigen Dateinamen speichern.

Werden nun innerhalb von Excel Änderungen an dieser Datei vorgenommen und gespeichert, nimmt Access beim erneuten Öffnen des Formulars automatisch eine Aktualisierung der Tabelle vor. Sie können diese Aktualisierung aber auch "von Hand" einleiten. Dafür ist das Dialogfenster *Verknüpfungen* zuständig, das Sie über den Menüpunkt *OLE* > *DDE-Verknüpfungen* des Menüs *Bearbeiten* öffnen. Markieren Sie in der Liste einfach das gewünschte Objekt und klicken Sie auf *Anzeige aktualisieren*. Access startet nun im Hintergrund den passenden OLE-Server, lädt die Datei und nimmt die Aktualisierung vor.

Wenn Sie keine automatische Aktualisierung wünschen, wählen Sie einfach den betreffenden Listeneintrag aus und markieren die Option *Manuell*.



Hinweis

Durch einen Klick auf *Quelle öffnen* können Sie das OLE-Objekt direkt bearbeiten. Sie sparen sich damit die Arbeit, zuerst den passenden OLE-Server zu starten und anschließend die Datei zu öffnen. Mit *Quelle ändern* können Sie die Verknüpfung in eine andere Datei oder einen anderen Bereich umleiten. Möchten Sie die Verknüpfung komplett auflösen, klicken Sie auf *Verknüpfung lösen*. Access löst dadurch die Verknüpfung und wandelt die Tabelle in eine Grafik um, die von nun an in dem Formular dargestellt wird.

Im Dialogfenster *Verknüpfungen* wird zu jeder Verknüpfung nicht nur der Dateiname, sondern zusätzlich auch der genaue Bereich angegeben. Bei einer verknüpften Excel-Tabelle könnte beispielsweise der folgende Text angegeben sein:

C:\Tabelle.xls!Tabelle1!Z1S1:Z5S2

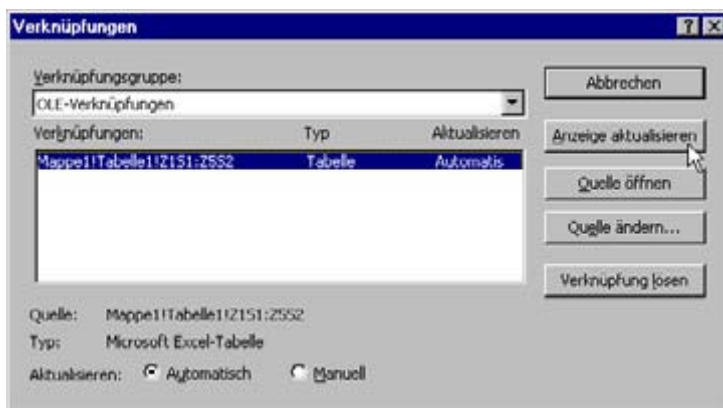


Verknüpfte Bereiche



Abb. 13.5

Das Dialogfenster
Verknüpfungen



Der Ausdruck besteht aus drei Bereichen, die jeweils durch ein Ausrufezeichen voneinander getrennt werden. Der erste Bereich `C:\Tabelle.xls` gibt die Datei an, mit der die Verknüpfung besteht. Der zweite Bereich bestimmt die Tabelle (in Excel können in einer Datei mehrere Tabelle vorhanden sein) und der dritte Bereich den genauen Zellbereich. In dem Beispiel wird der Bereich von Zeile 1 bis 5 und Spalte 1 bis 2 ausgewählt.

Hätten Sie anstelle der Excel-Tabelle einen Ausschnitt eines Word-Dokuments eingefügt, könnte folgende Verknüpfung angegeben werden:

`C:\Text.doc!OLE_LINK1`

Über den Ausdruck `OLE_LINK1` wird auf eine bestimmte Textmarke innerhalb des Word-Dokuments verwiesen. Diese Textmarke wird von Word automatisch angelegt, sobald Sie das OLE-Objekt mit Access verknüpfen. Innerhalb von Word können Sie die Textmarke mit *Einfügen > Textmarke* begutachten.

Abb. 13.6

Die Textmarke
`OLE_LINK1` innerhalb
von Word





Wie Sie eventuell bemerkt haben, können Sie die eingefügten Objekte nur dann zur Bearbeitung aktivieren, wenn das Formular in der Entwurfsansicht geöffnet ist. Der Grund dafür liegt darin, dass Access bei neu eingefügten Objektfeldern stets die *Aktiviert*-Eigenschaft auf *Nein* und die *Gesperrt*-Eigenschaft auf *Ja* einstellt. Wenn Sie diese Einstellungen in *Ja* bzw. *Nein* ändern, können Sie das Objekt auch dann aktivieren, wenn Sie das Formular in der Formularansicht öffnen.



Hinweis

Die Vor-Ort-Aktivierung

Wenn Sie bestimmte eingebettete Objekte (z. B. Objekte des Typs *Microsoft Word-Dokument*, *Microsoft Excel-Tabelle*, *Microsoft Graph* oder *Paint*) aktivieren, während sich das Formular in der Formularansicht befindet, werden Sie eine Überraschung erleben: In diesem Fall aktiviert Access das Objekt nicht in einem eigenen Anwendungsfenster, sondern per Vor-Ort-Aktivierung (Engl.: *In-Place-Activation*). Bei dieser Art der Aktivierung wird der OLE-Server nicht in einem eigenständigen Fenster gestartet, sondern blendet sich direkt innerhalb des Access-Formulars über dem OLE-Objekt ein. Zusätzlich werden einige Menüs von Access durch die entsprechenden Menüs des OLE-Servers ausgetauscht und – wenn verfügbar – die Symbolleisten des Servers eingeblendet. Das Ergebnis dieser Aktivierung ist, dass die Serveranwendung für den Benutzer nicht mehr als eigenständige Anwendung, sondern als Teil von Access erscheint.

Um die Bearbeitung eines per Vor-Ort-Aktivierung geöffneten Objekts zu beenden, klicken Sie einfach außerhalb des Objekts auf den Formularhintergrund.

Die Vor-Ort-Aktivierung steht nur bei Formularen zur Verfügung, die in der Formularansicht geöffnet sind. In der Entwurfsansicht werden OLE-Objekte grundsätzlich in einem eigenen Anwendungsfenster aktiviert. Außerdem werden verknüpfte Objekte, d. h. Objekte, die Sie als Verknüpfung eingefügt haben, grundsätzlich auch in einem eigenen Fenster geöffnet.



Hinweis

Wenn Sie ein Objekt einmal nicht per *Vor-Ort-Aktivierung* bearbeiten möchten, können Sie dies über den letzten Menüpunkt des Menüs *Bearbeiten* erledigen. Dieser Menüpunkt enthält ein Untermenü mit den verfügbaren Verben des Objekts. Das erste Verb *Bearbeiten* aktiviert das Objekt per Vor-Ort-Aktivierung; das zweite Verb mit dem Namen *Öffnen* in einem eigenen Anwendungsfenster. Da das erste Verb das Standard-Verb des Objekts ist, führt ein Doppelklick auf das Objekt in der Regel zur Vor-Ort-Aktivierung. Eine Ausnahme bilden lediglich OLE-Objekte, deren Server keine Vor-Ort-Aktivierung unterstützen.

Sie können das Standard-Verb mithilfe der *Verb*-Eigenschaft des gebundenen Objektfeldes ändern. Mehr dazu erfahren Sie im Abschnitt *Gebundene Objektfelder von VBA aus steuern*.



Hinweis

Da bei der Aktivierung eines OLE-Objekts – also auch bei Anwendung der Vor-Ort-Aktivierung – der komplette OLE-Server gestartet werden muss, kann die Aktivierung in einigen Fällen zur zeitraubenden Angelegenheit werden. Besonders auf Rechnern mit weniger als 32 MB Hauptspeicher lässt sich die Vor-Ort-Aktivierung



Tipp





nur mit sehr viel Geduld verwenden. Sie können den Vorgang allerdings etwas beschleunigen, indem Sie vorher die entsprechenden Server (z. B. Excel und Word) starten.

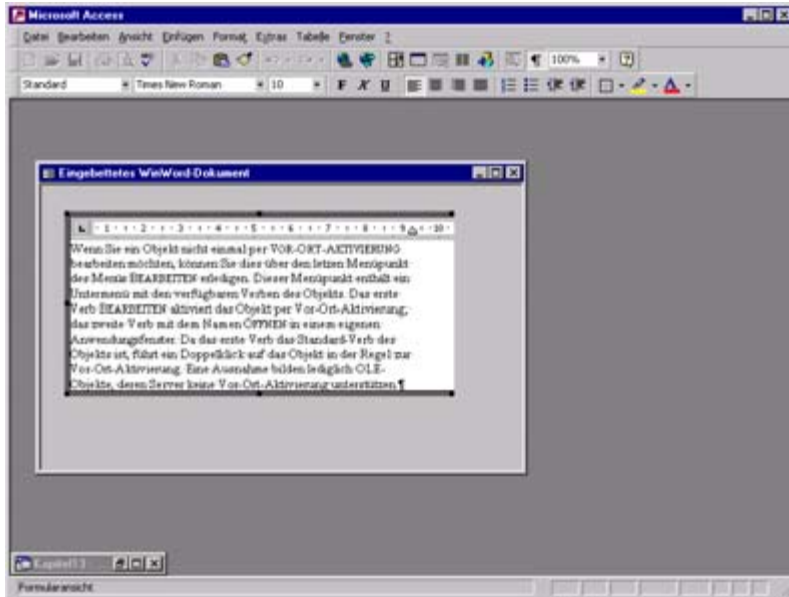


Abb. 13.7

Ein Vor-Ort-aktiviertes WinWord-Objekt

Außerdem sollten Sie bedenken, dass Sie nur solche Objekte bearbeiten können, deren OLE-Server auf Ihrem Rechner installiert sind. Konsequenzen hat dies besonders dann, wenn Sie eine Datenbank weitergeben.

Objektfelder

Alle OLE-Objekte, die Sie auf die vorher beschriebene Weise in ein Access-Formular einfügen, werden in einem Steuerelement des Typs *Objektfeld* gespeichert. Das Objektfeld lässt sich hierbei mit einem Behälter vergleichen, in dem sich das eigentliche OLE-Objekt befindet. Beim Einfügen eines OLE-Objekts wird dieses Steuerelement von Access automatisch angelegt, so dass Sie sich darum nicht zu kümmern brauchen.



Wenn Sie möchten, können Sie ein Objektfeld allerdings auch direkt über das *Objektfeld-Werkzeug* der Werkzeugleiste anlegen. Access öffnet daraufhin direkt den Dialog *Objekt einfügen* zum Einfügen eines OLE-Objekts in das Objektfeld.



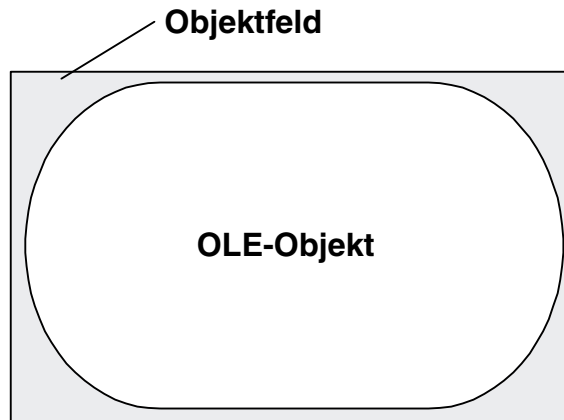


Abb. 13.8

Der Zusammenhang zwischen Objektfeld und OLE-Objekt

Wie bei Steuerelementen üblich, können Sie natürlich auch das Eigenschaftsfenster des Objektfeldes aktivieren. Hier finden Sie einige Eigenschaften, die speziell auf OLE-Objekte zugeschnitten sind. Die *Klasse*-Eigenschaft gibt an, welcher Objekttyp in dem *Objektfeld*-Steuerelement abgelegt ist. Der Text *Word.Document.10* weist beispielsweise auf ein Word-XP-Dokument hin. Mithilfe der Eigenschaft *OLE-Typ* können Sie zusätzlich ermitteln, ob es sich um ein eingebettetes oder verknüpftes Objekt handelt.

Eine wichtige Eigenschaft ist die *Größenanpassung*-Eigenschaft. Sie dient dazu, die Größe eines OLE-Objekts an die Größe des Objektfeldes anzupassen, in dem es enthalten ist. Die Standardeinstellung ist *Abschneiden*. Hierbei wird das OLE-Objekt in Originalgröße angezeigt. Sollte das Objektfeld zu klein sein, kann ein Teil des OLE-Objekts nicht dargestellt werden und wird daher abgeschnitten. Die Einstellung *Dehnen* bewirkt dagegen, dass das OLE-Objekt sowohl in der horizontalen als auch in der vertikalen Ausdehnung genau an die Größe des Objektfeldes angepasst wird. Hierbei werden allerdings meistens die Proportionen des OLE-Objekts verändert, was zu Verzerrungen führt. Abhilfe schafft die Einstellung *Zoomen*. Sie passt die Größe unter Beibehaltung der Proportionen an. Dies muss allerdings mit dem Nachteil erkauft werden, dass der Platz des Objektfeldes nicht immer vollständig ausgenutzt wird. In der folgenden Abbildung können Sie die Wirkung der drei Einstellungen begutachten.



Abb. 13.9

Die Einstellungen *Abschneiden*, *Dehnen* und *Zoomen* der Eigenschaft *Größenanpassung*



Gebundene Objektfelder

Arbeiten mit gebundenen Objektfeldern

Alle OLE-Objekte, die Sie bis jetzt kennen gelernt haben, wurden zusammen mit dem Formular gespeichert. Wenn Sie das Formular einmal in der Formularansicht öffnen und den Datensatz wechseln, zeigt Access unabhängig vom aktuellen Datensatz immer das gleiche OLE-Objekt an.

Was ist aber, wenn Sie Objekte während der Laufzeit einer Datenbank einfügen und verändern möchten und diese zusätzlich als Datensätze in einer Tabelle speichern möchten? Zu diesem Zweck stellt Access ein zweites Steuerelement mit dem Namen *gebundenes Objektfeld* zur Verfügung. Gebunden bedeutet in diesem Fall, dass das Steuerelement an ein Tabellenfeld gebunden werden kann, in dem Access die OLE-Objekte speichert.

Grundlage für das Arbeiten mit gebundenen Objektfeldern ist zunächst einmal eine Tabelle, die mindestens ein Datenfeld des Typs *OLE-Objekt* enthält. In der Beispieldatenbank *Kapitel13.mdb* finden Sie die Tabelle *Objekte*. Die Tabelle enthält die drei Felder *ObjektNr*, *Objektname* und *OLEObjekt*. Das letzte Feld mit dem Namen *OLEObjekt* hat den Felddatentyp *OLE-Objekt* und kann somit ein beliebiges OLE-Objekt (z. B. ein Word- oder ein Bitmap-Objekt) speichern.

In diese Tabelle können Sie nun Datensätze inklusive OLE-Objekten eingeben. Öffnen Sie die Tabelle dazu in der Datenblattansicht, und geben Sie in das Feld *Objektname* einen beliebigen Namen ein. Anschließend klicken Sie mit der Maus auf das Datenfeld *OLEObjekt*. Da es sich bei diesem Feld um ein Datenfeld des Typs *OLE-Objekt* handelt, zeigt Access hier keine Einfügemarke, sondern einen Rahmen an. In das so markierte Datenfeld können Sie nun mit den vorher beschriebenen Vorgehensweisen ein OLE-Objekt einbetten oder eine Verknüpfung definieren. Aus Platzgründen stellt Access innerhalb der Tabelle allerdings nicht das OLE-Objekt dar, sondern zeigt lediglich den Typ des Objekts als Text an (z. B. *Microsoft Word-Dokument*).

ObjektNr	Objektname	OLEObjekt
2	Ein Word-OLE-Objekt	Microsoft Word-Dokument
3	Ein Excel-Objekt	Microsoft Excel-Tabelle
14	Noch ein Excel-Objekt	Microsoft Excel-Tabelle

Abb. 13.10

Die Tabelle *Objekte* in der Datenblattansicht

Diese Darstellungsweise von OLE-Objekten ist für alle Datenblattansichten typisch. Um das OLE-Objekt vollständig anzeigen zu lassen, ist ein Formular oder ein Bericht erforderlich. Ein passendes Formular können Sie beispielsweise mithilfe des Formularassistenten erstellen lassen. Der Assistent erkennt automatisch, dass das Datenfeld *OLEObjekt* den Felddatentyp *OLE-Objekt* hat und legt in dem Formular ein gebundenes Objektfeld an. Natürlich können Sie das Formular auch "von Hand" erstellen. Das gebundene Objektfeld legen Sie in diesem Fall einfach mit dem Werk-



zeug *Gebundenes Objektfeld* an. Anschließend dürfen Sie nicht vergessen, das Feld per *Steuerelementinhalt*-Eigenschaft an das Datenfeld *OLEObjekt* der zugrunde liegenden Tabelle *Objekte* zu knüpfen.

In der Beispieldatenbank finden Sie das fertige Formular unter dem Namen *Objekte1*.

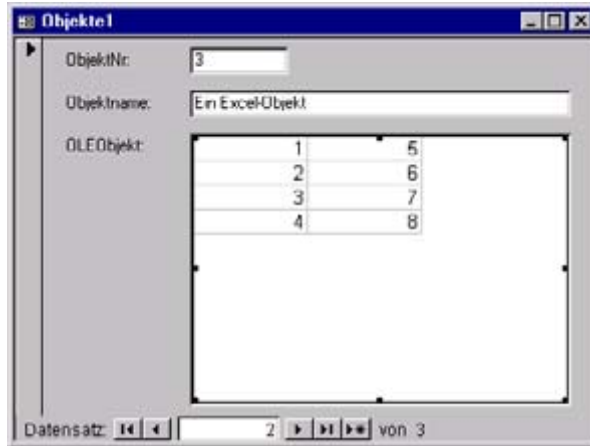


Abb. 13.11

Das Formular *Objekte1*

Das Formular können Sie nun zum Experimentieren in der Formularansicht öffnen. Zum Einfügen und Verknüpfen von Objekten stehen Ihnen jetzt die gleichen Möglichkeiten zur Verfügung, die Sie vorher bereits bei ungebundenen Objektfeldern kennen gelernt haben. Der einzige Unterschied besteht darin, dass Sie nun in der Formularansicht arbeiten und pro Datensatz ein neues Objekt einfügen können. Probieren Sie es einfach einmal aus und fügen Sie einige Objekte ein. Zusätzlich können Sie jedem Objekt über das Textfeld *Objektname* einen Namen geben.

Wenn Sie ein Objekt nachträglich bearbeiten möchten, wechseln Sie einfach über die Navigationsschaltflächen zu dem gewünschten Datensatz und klicken das Objekt doppelt an.

Ein vorhandenes Objekt können Sie problemlos löschen, indem Sie einfach den kompletten Datensatz löschen. Wenn Sie allerdings lediglich das OLE-Objekt, nicht aber den Inhalt der übrigen Datenfelder löschen möchten, gehen Sie folgendermaßen vor: Klicken Sie das Objekt einmal mit der Maus an, um es zu markieren. Sie erkennen die Markierung daran, dass das Objekt nun von einem Rahmen umgeben wird. Auf das markierte Objekt wenden Sie nun den Befehl *Löschen* des Menüs *Bearbeiten* an.

Löschen und Kopieren eines Objekts

Auch das Ausschneiden oder Kopieren des Objekts in die Zwischenablage ist über die entsprechenden Menüpunkte möglich. Von der Zwischenablage aus können Sie das Objekt in einen anderen Datensatz, ein anderes Formular oder sogar eine andere Windows-Anwendung einfügen.



Hinweis



Gebundene Objektfelder müssen nicht unbedingt an ein Tabellenfeld gebunden sein. Wenn keine Bindung vorliegt, wird ein eingefügtes OLE-Objekt allerdings weder in der Datenbank noch zusammen mit dem Formular gespeichert und geht damit beim Schließen des Formulars verloren. Darüber hinaus gibt es bei OLE-Objekten mit hohem Speicherbedarf Probleme, wenn diese nicht an ein Tabellenfeld gebunden sind.

Tipp



Beim Wechsel des Datensatzes benötigt Access einige Zeit, um das OLE-Objekt darzustellen. Dies können Sie beschleunigen, indem Sie die *Anzeigeart*-Eigenschaft von *Inhalt* in *Symbol* ändern. Access zeigt nun innerhalb des Objektfeldes nur noch ein Symbol an, das den Typ des OLE-Objekts andeutet. Beachten Sie, dass sich eine Veränderung dieser Eigenschaft nicht auf bereits vorhandene, sondern nur auf neu eingefügte OLE-Objekte auswirkt. Außerdem ist bei der Einstellung *Symbol* grundsätzlich keine Vor-Ort-Aktivierung möglich!

Gebundene Objektfelder von VBA aus steuern

Im vorhergehenden Abschnitt haben Sie mehrere Techniken zum Einfügen und Bearbeiten von OLE-Objekten kennen gelernt. Damit sich diese Funktionen auch von VBA aus steuern lassen, stellen alle gebundenen Objektfelder die *Action*-Eigenschaft zur Verfügung:

```
<Objektfeld>.Action = <Wert>
```

Das Besondere an der *Action*-Eigenschaft ist, dass sie sich wie eine Methode verhält: Sobald Sie ihr einen neuen Wert zuweisen, führt Access die betreffende Aktion aus. Welche Aktion ausgeführt wird, bestimmt der Wert, den Sie der Eigenschaft zuweisen. Die folgende Tabelle enthält eine Übersicht und eine kurze Beschreibung aller gültigen Werte. Eine genaue Erklärung finden Sie in den folgenden Abschnitten.

Gültige Werte der Action-Eigenschaft

Wert	Konstante	Aktion
0	<i>acOLECreateEmbed</i>	Bettet ein neues OLE-Objekt ein.
1	<i>acOLECreateLink</i>	Verknüpft ein OLE-Objekt.
4	<i>acOLECopy</i>	Kopiert das Objekt in die Zwischenablage.
5	<i>acOLEPaste</i>	Fügt ein Objekt aus der Zwischenablage in das Objektfeld ein.
6	<i>acOLEUpdate</i>	Aktualisiert ein verknüpftes Objekt.
7	<i>acOLEActivate</i>	Aktiviert das Objekt.
9	<i>acOLEClose</i>	Schließt ein aktiviertes Objekt.



Gültige Werte der Action-Eigenschaft (Forts.)

Wert	Konstante	Aktion
10	<i>acOLEDelete</i>	Löscht das OLE-Objekt.
14	<i>acOLEInsertObjDlg</i>	Zeigt den Dialog <i>Objekt einfügen</i> an.
15	<i>acOLEPasteSpecialDlg</i>	Zeigt den Dialog <i>Inhalte einfügen</i> an.
17	<i>acOLEFetchVerbs</i>	Fordert die unterstützten Verben eines OLE-Objekts erneut an.

Tabelle 13.1

Das Zuweisen einer der Werte an die *Action*-Eigenschaft wird im Folgenden als *Ausführen der betreffenden Aktion* bezeichnet. Wenn beispielsweise die Rede von der Ausführung der *acOLEDelete*-Aktion ist, bedeutet dies, dass Sie der *Action*-Eigenschaft den Wert 10 bzw. die Konstante *acOLEDelete* zuweisen.

Welche Aktionen bei einem Objekt verfügbar sind, hängt von den Einstellungen der Eigenschaften *Aktiviert* und *Gesperrt* ab. Ist ein Objekt aktiviert und nicht gesperrt, stehen alle Eigenschaften zur Verfügung. Sollte das betreffende Objekt deaktiviert sein – die *Aktiviert*-Eigenschaft besitzt in diesem Fall den Inhalt *False* (*Nein*) –, stehen ausschließlich die Aktionen *acOLECopy* und *acOLEActivate* zur Verfügung. Bei gesperrten Objekten sind neben den beiden Aktionen auch die Aktionen *acOLEClose* und *acOLEFetchVerbs* verfügbar.

Aktivieren von Objekten

Die wohl am häufigsten verwendete Aktion ist die Aktivierung eines OLE-Objekts, um dieses beispielsweise zu bearbeiten. Die Aktivierung nehmen Sie vor, indem Sie der *Action*-Eigenschaft den Wert 7 bzw. die Konstante *acOLEActivate* zuweisen. Um das Objekt aus dem Beispielformular *Objekte1* zu aktivieren, ist die folgende Programmzeile ausreichend:

```
Me!OLEObjekt.Action = acOLEActivate
```

Wie bereits vorher erwähnt, besitzt jedes Objekt verschiedene Möglichkeiten zur Aktivierung. Ein Objekt des Audiorecorders (Wave-Audio) kann beispielsweise zur Bearbeitung oder direkt zur Wiedergabe aktiviert werden. Auf welche Art das Objekt aktiviert wird, legen Sie über die *Verb*-Eigenschaft des Objekts fest. Die folgende Tabelle enthält alle gültigen Werte, die Sie der *Verb*-Eigenschaft eines Objektfeldes zuweisen können:

Die *Verb*-
Eigenschaft



Gültige Werte der *Verb*-Eigenschaft

Wert	Konstante	Aktion
0	<i>acOLEVerbPrimary</i>	Aktiviert das Objekt mit der Standardaktion.
-1	<i>acOLEVerbShow</i>	Aktiviert das Objekt in der Standardansicht zur Bearbeitung.
-2	<i>acOLEVerbOpen</i>	Aktiviert das Objekt in einem eigenen Anwendungsfenster.
-3	<i>acOLEVerbHide</i>	Aktiviert das Objekt, ohne dass dies für den Anwender sichtbar ist (nur bei eingebetteten Objekten). Diese Art der Aktivierung wird vornehmlich im Zusammenhang mit der OLE-Automation benötigt.
-4	<i>acOLEVerbInPlaceUIActivate</i>	Aktiviert das Objekt per Vor-Ort-Aktivierung.
-5	<i>acOLEVerbInPlaceActivate</i>	Aktiviert das Objekt per Vor-Ort-Aktivierung, ohne jedoch die Menüs und die Symbolleisten des Servers einzublenden.

Tabelle 13.2

Ob ein bestimmtes Verb unterstützt wird, ist vom jeweiligen OLE-Server abhängig. Das Verb *acOLEVerbHide* wird beispielsweise von Excel, nicht jedoch von Word unterstützt. Die Verben *acOLEVerbInPlaceUIActivate* und *acOLEVerbInPlaceActivate* stehen nur bei Objekten zur Verfügung, die Vor-Ort-Aktivierung unterstützen.

Sie können die *Verb*-Eigenschaft des gebundenen Objektfeldes bereits beim Entwurf des Formulars oder – von VBA aus – während der Laufzeit der Datenbank festlegen. Das folgende Beispiel aktiviert das Objekt beispielsweise stets in einem eigenen Anwendungsfenster:

```
Me!OLEObjekt.Verb = acOLEVerbOpen
Me!OLEObjekt.Action = acOLEActivate
```

In der Beispieldatenbank *Kapitel13.mdb* befindet sich im Formular *Objekte2* eine entsprechende Schaltfläche mit der Aufschrift *Im Fenster aktivieren*. Ein Klick auf diese Schaltfläche genügt, um das Objekt zu aktivieren. Vor der Aktivierung muss jedoch noch überprüft werden, ob das Objektfeld wirklich ein OLE-Objekt enthält oder eventuell leer ist. Wird die Aktivierung an einem leeren Objektfeld vorgenommen, tritt ein Laufzeitfehler auf.



Die Überprüfung auf einen Inhalt lässt sich mit Access wie gewohnt über die *IsNull*-Funktion durchführen. Die komplette Ereignisprozedur der Schaltfläche hat den folgenden Aufbau:

```
Private Sub btnAktivieren_Click()

    If IsNull(Me!OLEObjekt) Then

        Beep
        MsgBox "Es ist kein Objekt vorhanden!"

    Else

        Me!OLEObjekt.Verbo = acOLEVerboOpen
        Me!OLEObjekt.Action = acOLEActivate

    End If

End Sub
```

Die vorhergehende Tabelle enthält insgesamt sechs Verben, die über negative Werte bzw. die entsprechenden Konstanten ausgewählt werden. Neben den Verben der Tabelle unterstützen bestimmte Objekte allerdings noch einige zusätzliche Verben, die speziell auf die Funktion der Objekte zugeschnitten sind. Die Auswahl dieser Verben erfolgt grundsätzlich über positive Werte.

Ermitteln der Verben eines Objekts

Um eine vollständige Liste aller Verben anzufordern, stellen Objektfelder zwei Eigenschaften mit den Namen *ObjectVerbs* und *ObjectVerbsCount* zur Verfügung. Die *ObjectVerbsCount*-Eigenschaft enthält die Gesamtanzahl aller Verben, die ein Objekt unterstützt. Die *ObjectVerbs*-Eigenschaft gibt zu einer bestimmten Verbo-Nummer die entsprechende Aktion im Klartext zurück. Das folgende Beispiel listet alle Verben des aktuellen OLE-Objekts im Testfenster auf:

```
Dim i As Integer

Me!OLEObjekt.Action = acOLEFetchVerbs

For i = 0 To Me!OLEObjekt.ObjectVerbsCount - 1

    Debug.Print Me!OLEObjekt.ObjectVerbs(i)

Next i
```

Vor dem Auflisten der Verben wird über die Programmzeile *Me!OLEObjekt.Action = acOLEFetchVerbs* die interne Liste der Verben aktualisiert. Dies ist notwendig, damit ein Objekt – abhängig von seinem Zustand – unterschiedliche Verben unterstützen kann. Bei einem Word-Objekt führt die Beispielprozedur zu folgendem Ergebnis:

```
&Bearbeiten
&Bearbeiten
Ö&ffnen
```



Hinweis



Der erste Eintrag mit dem Index 0 entspricht dem Standard-Verb des Objekts. Ab dem Index 1 folgt dann die komplette Liste aller Verben, inklusive der Verben, die Sie eventuell auch über einen negativen Wert auswählen können. Wie Sie der Liste entnehmen können, enthält jeder Eintrag ein &-Zeichen, das bei Verwendung des Textes als Menübeschriftung oder Schaltflächenbeschriftung eine Schnellzugriffstaste definiert.

Das Beispielformular *Objekte2* enthält ein Listenfeld, in dem die verfügbaren Verben des aktuellen OLE-Objekts aufgelistet werden. Hier können Sie ein beliebiges Verb aktivieren und mittels der Schaltfläche *Mit ausgewähltem Verb aktivieren* ausführen.



Abb. 13.12

Das Formular *Objekte2*

Tip



Wenn Sie eine mögliche Aktivierung eines Objekts durch einen Doppelklick auf das OLE-Objekt verhindern möchten, stellen Sie einfach in der Entwurfsansicht die *Objektaktivierung*-Eigenschaft des Objektfeldes auf *Manuell* ein. Die Aktivierung kann nun ausschließlich von VBA erfolgen. Ungebundene Objektfelder stellen außerdem die Einstellung *Fokus* zur Verfügung. Bei dieser Einstellung wird das Objekt aktiviert, sobald es den Fokus erhält. Diese Einstellung ist allerdings nur bei sehr kleinen OLE-Servern praktikabel, bei denen die Aktivierung sehr schnell durchgeführt werden kann.

Wenn Sie grundsätzlich eine Aktivierung des Objekts verhindern möchten, stellen Sie einfach die *Aktiviert*-Eigenschaft (*Enabled*-Eigenschaft) auf *Nein* (*False*) und die *Gesperrt*-Eigenschaft (*Locked*-Eigenschaft) auf *Ja* (*True*) ein.

Deaktivieren eines Objekts

Besonders im Zusammenhang mit der OLE-Automation, die Sie im Abschnitt *OLE-Automation* noch kennen lernen, ist es notwendig, ein aktiviertes Objekt von VBA aus wieder zu deaktivieren. Dies erledigen Sie, indem Sie der *Action*-Eigenschaft den Wert 9 (*acOLEClose*) zuweisen. Beachten Sie, dass diese Aktion nur bei eingebetteten Objekten zur Verfügung steht!





Die Aktivierung eines Objekts über einen Doppelklick ist für den Anwender eine intuitive Vorgehensweise, die er von anderen Windows-Anwendungen kennt. Für den VBA-Programmierer besitzt diese Art der Aktivierung jedoch den Nachteil, dass er nichts davon erfährt, wenn ein Objekt aktiviert oder verändert wurde. Aus diesem Grund stellt Access bei Objektfeldern das *OLE-Aktualisierung-Ereignis* (*Updated-Ereignis*) zur Verfügung. Die entsprechende Ereignisseigenschaft dieses Ereignisses trägt den Namen *Bei OLE Aktualisierung*. Access löst dieses Ereignis immer dann aus, wenn Änderungen an einem OLE-Objekt vorgenommen wurden. Dazu zählt auch die Aktivierung eines Objekts. Die Auslösung des Ereignisses ist unabhängig davon, ob die Aktion vom Anwender per Doppelklick oder von VBA aus mithilfe der *Action*-Eigenschaft vorgenommen wurde.

Das *Update*-Ereignis

Die Ereignisprozedur hat einen Parameter mit dem Namen *Code*, der die Art der Änderung angibt, die an dem Objekt vorgenommen wurde. Der Kopf einer entsprechenden Ereignisprozedur könnte folgendermaßen aussehen:

```
Private Sub OLEObjekt_Updated (Code As Integer)
```

Beim Start der Prozedur weist Access der Variablen *Code* einen der folgenden Werte zu:

Werte für *Code*

Wert	Konstante	Beschreibung
0	<i>acOLEChanged</i>	Der Inhalt des Objekts wurde verändert.
1	<i>acOLESaved</i>	Die Änderungen an dem Objekt wurden an Access weitergegeben und hier gespeichert.
2	<i>acOLEClosed</i>	Das Objekt wurde geschlossen.
3	<i>acOLERenamed</i>	Das Herkunftsdocument eines verknüpften OLE-Objekts wurde umbenannt.

Tabelle 13.3

Das Ereignis wird grundsätzlich erst ausgelöst, nachdem ein aktiviertes Objekt wieder geschlossen wird. In einigen Fällen ruft Access die Prozedur mehrmals mit unterschiedlichen Werten für *Code* auf, um alle Änderungen an dem Objekt zu dokumentieren.

Das Objektfeld des Beispielformulars *Objekte2* enthält eine entsprechende Ereignisprozedur, die beim Auftreten des Ereignisses den Wert der Variablen *Code* in einem Meldungsfenster anzeigt.





Einfügen neuer Objekte

Eine weitere wichtige Aufgabe ist das Einfügen neuer Objekte von VBA aus. Am einfachsten erledigen Sie dies, indem Sie der *Action*-Eigenschaft des betreffenden Objektfeldes einfach den Wert 14 bzw. die Konstante *acOLEInsertObjDlg* zuweisen. Access öffnet dadurch automatisch den *Objekt einfügen*-Dialog, den Sie bereits von dem gleichnamigen Menüpunkt des Menüs *Bearbeiten* kennen. Hier kann der Anwender einen beliebigen OLE-Server auswählen.

Das Formular *Objekte3* der OLE-Beispieldatenbank enthält eine entsprechende Schaltfläche mit der Beschriftung *Objekt einfügen-Dialog*, mit der Sie ein neues Objekt einfügen können.

Während Access das Dialogfenster anzeigt, stoppt die Ausführung der VBA-Prozedur, bis das Objekt eingefügt bzw. die Schaltfläche *Abbrechen* betätigt wird. Im ersten Fall wird der Programmcode mit der nachfolgenden Anweisung fortgesetzt. Klickt der Anwender allerdings auf *Abbrechen*, tritt ein Laufzeitfehler mit der Fehlernummer 2001 auf.

Dieser Fehler lässt sich ohne Probleme mit Hilfe der Fehlerbehandlungsfunktionen von Access abfangen. Wenn Sie den Fehler beispielsweise komplett ignorieren möchten, können Sie Access anweisen, alle Laufzeitfehler zu ignorieren:

```
On Error Resume Next
Me!OLEObjekt.Action = acOLEInsertObjDlg
On Error Goto 0
```

Eleganter erledigen Sie dies allerdings, wenn Sie eine Fehlerbehandlungsroutine einsetzen. Das folgende Listing zeigt den Ereigniscode der *Beim Klicken*-Eigenschaft, wie er in dem Formular *Objekte3* verwendet wird. Die Prozedur überprüft zudem, ob bereits ein Objekt vorhanden ist. Ist dies der Fall, kann der Anwender bestimmen, ob das Objekt überschrieben werden soll.

```
Private Sub btn_Neu_Click()

On Error GoTo Err

Dim Meldung As String

If Not IsNull(Me!OLEObjekt) Then

    Beep
    Meldung = "Es ist bereits ein Objekt vorhanden."
    Meldung = Meldung & "Soll es gelöscht werden?"

    If MsgBox(Meldung, vbYesNo, "Objekt einfügen") _
        = vbNo Then
        Exit Sub
    End If

Err:
```





```
End If

Me!OLEObjekt.Action = acOLEInsertObjDlg

Exit Sub

Err:

If Err.Number = 2001 Then

    Beep
    MsgBox "Der Vorgang wurde abgebrochen!"

Else

    MsgBox Err.Description

End If

Exit Sub

End Sub
```

Das Einfügen eines Objekts über den *Objekt einfügen*-Dialog ist für den Anwender sehr flexibel, da er selbstständig den gewünschten Objekttyp auswählen kann. Wenn Sie den Objekttyp allerdings während der Entwicklung der Datenbank festlegen möchten, ist dieser Dialog nicht geeignet. In diesem Fall müssen Sie auf die Aktionen *acOLECreateEmbed* bzw. *acOLECreateLink* zurückgreifen, die Sie im Folgenden kennen lernen.

Bevor Sie ein neues Objekt mithilfe der beiden Aktionen einfügen, müssen Sie zunächst einmal über die *Klasse*-Eigenschaft (*Class*-Eigenschaft) des Objektfeldes festlegen, welcher Objekttyp eingefügt werden soll. Sie können diese Einstellung entweder beim Entwurf des Formulars oder während der Laufzeit von VBA aus vornehmen. Der *Klasse*-Eigenschaft muss dazu lediglich der genaue Klassenname des Objekttyps zugewiesen werden. Die folgende Tabelle enthält die Klassennamen der gängigsten Windows-Anwendungen:

Auswahl der Objektklasse

Objektklassen der gängigsten Windows-Anwendungen



Klassenname	Objekttyp
<i>Equation.3</i>	Microsoft Equation 3.0
<i>Excel.Chart.5</i>	Microsoft Excel 5.0/95-Diagramm
<i>Excel.Chart.8</i>	Microsoft Excel 97-Diagramm
<i>Excel.Chart.9</i>	Microsoft Excel 2000-Diagramm



Objektklassen der gängigsten Windows-Anwendungen (Forts.)

Klassenname	Objektyp
<i>Excel.Chart.10</i>	Microsoft Excel XP-Diagramm
<i>Excel.Sheet.5</i>	Microsoft Excel 5.0/95-Tabelle
<i>Excel.Sheet.8</i>	Microsoft Excel 97-Tabelle
<i>Excel.Sheet.9</i>	Microsoft Excel 2000-Tabelle
<i>Excel.Sheet.10</i>	Microsoft Excel XP-Tabelle
<i>MPlayer</i>	Medienwiedergabe-Objekt (z. B. AVI-Video)
<i>MS_ClipArt_Gallery.2</i>	Microsoft ClipArt Gallery 2.0
<i>MSDataMap.1</i>	Microsoft Data Map-Landkarte
<i>MSGraph.Chart.5</i>	Microsoft Graph 5.0
<i>MSGraph.Chart.8</i>	Microsoft Graph 97
<i>MSGraph.Chart.9</i>	Microsoft Graph 2000
<i>MSGraph.Chart.10</i>	Microsoft Graph XP
<i>MSWordArt.2</i>	Microsoft Wordart
<i>Paint.Picture</i>	Bitmap
<i>PowerPoint.Show.8</i>	MS PowerPoint 95-Präsentation
<i>PowerPoint.Show.9</i>	MS PowerPoint 2000-Präsentation
<i>PowerPoint.Slide.8</i>	MS PowerPoint 95-Folie
<i>PowerPoint.Slide.9</i>	MS PowerPoint 2000-Folie
<i>PowerPoint.Slide.10</i>	MS PowerPoint XP-Folie
<i>SoundRec</i>	WAV Audio
<i>Word.Document.6</i>	Microsoft Word 6.0/95-Dokument
<i>Word.Picture.6</i>	Microsoft Word 6.0/95-Grafik
<i>Word.Document.8</i>	Microsoft Word 97-Dokument
<i>Word.Document.9</i>	Microsoft Word 2000-Dokument
<i>Word.Document.10</i>	Microsoft Word XP-Dokument
<i>WordPad.Document.1</i>	WordPad-Dokument

Tabelle 13.4



Wenn Sie den Klassennamen eines bestimmten Objekttyps nicht kennen, fügen Sie einfach ein Objekt des gewünschten Typs über den Menüpunkt *Bearbeiten > Objekt einfügen* zu Testzwecken als ungebundenes Objektfeld in ein Formular ein. Bei diesem Vorgang trägt Access automatisch den richtigen Klassennamen in die *Klasse*-Eigenschaft ein.



Tipp

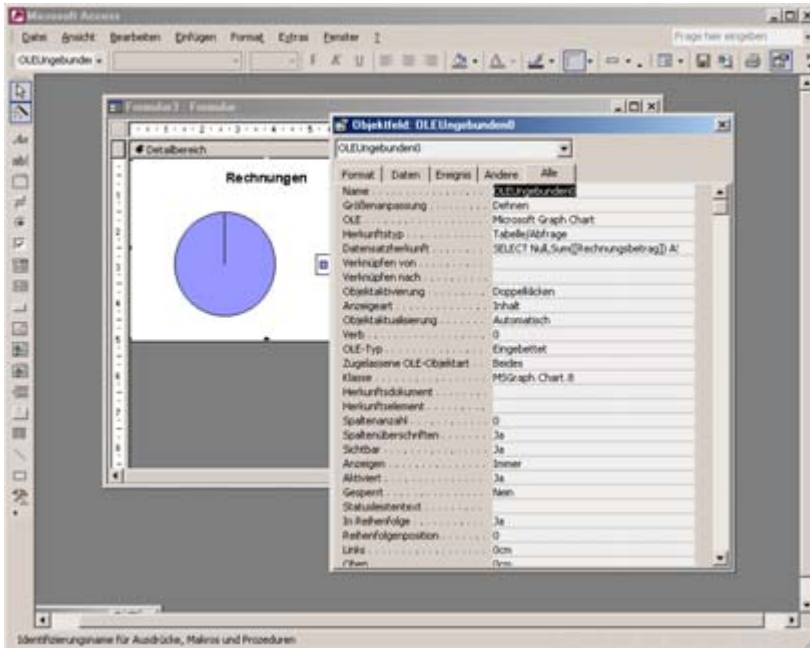


Abb. 13.13

Der Klassenname eines Objekts

Um ein Objekt von VBA aus einzubetten, legen Sie also zuerst die Objektklasse fest und führen dann die Aktion *acOLECreateEmbed* aus:

Objekt einbetten

```
Me!OLEObjekt.Class = "Paint.Picture"
Me!OLEObjekt.Action = acOLECreateEmbed
```

Durch die Aktion wird das neue Objekt lediglich eingebettet, aber nicht automatisch zur Bearbeitung aktiviert. Dies können Sie jedoch über die Aktion *acOLEActivate* nachholen:

```
Me!OLEObjekt.Class = "Paint.Picture"
Me!OLEObjekt.Action = acOLECreateEmbed
Me!OLEObjekt.Verb = acOLEVerbPrimary
Me!OLEObjekt.Action = acOLEActivate
```

Wenn Sie ein Objekt einfügen möchten, das bereits als Datei auf Ihrer Festplatte vorliegt, tragen Sie einfach den Namen dieser Datei in die *Herkunftsdokument*-Eigenschaft (*SourceDoc*-Eigenschaft) des Objektfeldes ein. Access ermittelt nun

Datei als Objekt einbetten





anhand der Dateieindung die passende OLE-Klasse und fügt eine Kopie der Datei als Objekt ein. Diese Kopie kann der Anwender dann beispielsweise als Vorlage verwenden und nach seinen Bedürfnissen verändern. Das Herkunftsdocument wird von diesen Änderungen nicht beeinflusst!

Das folgende Beispiel bettet ein Word-Dokument ein, das unter dem Namen *C:\Dokument1.doc* gespeichert wurde:

```
Me!OLEObjekt.SourceDoc = "C:\Dokument1.doc"  
Me!OLEObjekt.Action = acOLECreateEmbed
```

Zusätzlich zu dem Dateinamen des Herkunftsdocuments können Sie über die *Herkunftselement*-Eigenschaft (*SourceItem*-Eigenschaft) auf einen bestimmten Bereich des Dokuments verweisen, der dann als Objekt eingefügt wird (z.B. *Tabelle1!Z1S1:Z4S2*). Sie haben dies bereits im Abschnitt *Verknüpfen von Objekten* kennen gelernt. Allerdings müssen Sie hier stets den Dateinamen und die Bereichsangaben auf die beiden Eigenschaften *Herkunftsdocument* und *Herkunftselement* aufteilen, wie das folgende Beispiel zeigt:

```
Me!OLEObjekt.SourceDoc = "C:\Mappe1.XLS"  
Me!OLEObjekt.SourceItem = "Tabelle1!Z1S1:Z4S2"  
Me!OLEObjekt.Action = acOLECreateEmbed
```

Hinweis



Beim Einfügen eines neuen leeren Objekts sollten Sie darauf achten, dass die *Herkunftsdocument*-Eigenschaft keinen Inhalt hat. Im Zweifelsfall sollten Sie dieser Eigenschaft vor dem Einbetten also immer eine leere Zeichenkette zuweisen:

```
Me!OLEObjekt.SourceDoc = ""  
Me!OLEObjekt.Class = "Paint.Picture"  
Me!OLEObjekt.Action = acOLECreateEmbed
```

Wenn Sie sowohl ein Herkunftsdocument als auch eine – eventuell nicht zum Herkunftsdocument passende – Objektklasse angeben, ignoriert Access die Objektklasse und orientiert sich ausschließlich am Herkunftsdocument.

Das Formular *Objekt3* der Beispieldatenbank *Kapitel13.mdb* enthält mehrere Schaltflächen, die das Einfügen von Objekten auf die vorgestellten Weisen demonstriert.

Objekt verknüpfen

Das Verknüpfen eines Objekts erledigen Sie fast auf die gleiche Weise wie das Einbetten eines Objekts, das als Datei vorliegt. Anstelle der Aktion *acOLECreateEmbed* verwenden Sie lediglich die Aktion *acOLECreateLink*. Das folgende Beispiel stellt beispielsweise eine Verknüpfung zur dem Word-Dokument *C:\Dokument1.doc* her:

```
Me!OLEObjekt.SourceDoc = "C:\Dokument1.doc"  
Me!OLEObjekt.Action = acOLECreateLink
```



Über die *Herkunftselement*-Eigenschaft (*SourceItem*-Eigenschaft) können Sie natürlich auch wieder einen Bereich der Datei angeben, mit dem Sie eine Verknüpfung herstellen möchten. Das folgende Beispiel stellt eine Verknüpfung zu einem bestimmten Bereich der Word-Datei *C:\Dokument1.doc* her. Dem Bereich wurde vorher innerhalb von Word eine Textmarke mit dem Namen *Textmarke1* zugewiesen.

```
Me!OLEObjekt.SourceDoc = "C:\Dokument1.doc"
Me!OLEObjekt.SourceItem = "Textmarke1"
Me!OLEObjekt.Action = acOLECreateLink
```

Zur Aktualisierung des verknüpften Objekts verwenden Sie die *acOLEUpdate*-Aktion.



Hinweis

Arbeiten mit der Zwischenablage

Zum Austausch von Objekten mit der Zwischenablage stellt Access die zwei Aktionen *acOLECopy* und *acOLEPaste* zu Verfügung, die von der Funktion her den beiden Menüpunkten *Kopieren* und *Einfügen* des Menüs *Bearbeiten* entsprechen. Um ein Objekt der Beispieldatenbank in die Zwischenablage zu kopieren, geben Sie beispielsweise den folgenden Ausdruck an:

```
Me!OLEObjekt.Action = acOLECopy
```

Das Kopieren funktioniert sowohl bei eingebetteten als auch bei verknüpften Objekten. Dies gilt auch für den Fall, dass Sie ein Objekt mittels *acOLEPaste* aus der Zwischenablage in ein Objektfeld einfügen. Access erkennt hierbei automatisch, ob es sich um eine Einbettung oder eine Verknüpfung handelt, und setzt die Eigenschaft *OLEObjektart* (*OLEType*) dementsprechend auf den Wert 0 (verknüpft) oder 1 (eingebettet). War das Einfügen nicht erfolgreich, enthält die Eigenschaft den Inhalt 3.

Vor dem Einfügen eines Objekts müssen Sie sicherstellen, dass die Eigenschaft *Zugelassene OLE Objektart* die richtige Einstellung besitzt. Über diese Eigenschaft lässt sich bestimmen, ob ein Objektfeld nur verknüpfte (Wert 0), nur eingebettete (Wert 1) oder beide Typen (Wert 2) von OLE-Objekten aufnehmen kann.



Hinweis

Beim Einfügen eines Objekts aus der Zwischenablage in ein Objektfeld über die *acOLEPaste*-Aktion wird ein eventuell vorhandenes Objekt ohne Vorwarnung überschrieben!



Hinweis

Eine weitere Möglichkeit zum Einfügen eines Objekts stellt die Aktion *acOLEPasteSpecialDlg* dar. Sie zeigt den Dialog *Inhalte einfügen* an, über den der Anwender das Format des einzufügenden Objekts auswählen kann. Außerdem kann hier zwischen *Verknüpfung* und *Einfügen* ausgewählt werden. Wie vorher bereits im Zusammenhang mit der *acOLEInsertObjDlg*-Aktion beschrieben, tritt ein Laufzeitfehler auf, sobald der Anwender die *Abbrechen*-Schaltfläche betätigt. Dieser kann mit einer Fehlerbehandlungsroutine abgefangen werden, wie das folgende Beispiel demonstriert:





```
Private Sub btnInhalteEinfügen_Click()

    If Not IsNull(Me!OLEObjekt) Then

        Beep
        If MsgBox("Es ist bereits ein Objekt vorhanden. " _
            & "Soll es gelöscht werden?", _
            3 + 32, "Objekt einfügen") <> 6 Then

            Exit Sub

        End If

    End If

    On Error GoTo Err_InhalteEinfügen
    Me!OLEObjekt.Action = acOLEPasteSpecialDlg

    Exit Sub

Err_InhalteEinfügen:

    If Err.Number = 2001 Then

        Beep
        MsgBox "Der Vorgang wurde abgebrochen!"

    Else

        MsgBox Err.Description

    End If

End Sub
```

Den Programmcode zu diesem Beispiel finden Sie in dem Formular *Objekte4* der Beispieldatenbank. Die Prozedur ist mit dem *Beim Klicken*-Ereignis der Schaltfläche *"Inhalte einfügen"-Dialog* verknüpft.

Hinweis



Befindet sich in der Zwischenablage kein Objekt, das in das Objektfeld eingefügt werden kann, tritt bei Ausführung der Aktionen *acOLEPaste* und *acOLEPasteSpecialDlg* ein Laufzeitfehler mit dem Fehlercode 2234 auf.

Tipp



Wenn Sie ein Objekt von einem Datensatz in einen anderen Datensatz kopieren möchten, können Sie dies mithilfe der Zwischenablage bewerkstelligen. Dazu öffnen Sie zunächst das passende Formular und wechseln zu dem Datensatz, in dem sich das Objekt befindet. Anschließend kopieren Sie es mithilfe der Aktion *acOLECopy* in die Zwischenablage, wechseln zu dem zweiten Datensatz und fügen das Objekt hier wieder per *acOLEPaste* ein. Diese Vorgehensweise ist allerdings etwas umständlich, da unter anderem immer das betreffende Formular geöffnet werden muss.



OLE-Objekte werden innerhalb von Tabellenfeldern als lange binäre Byte-Folgen gespeichert. Diese werden auch als *BLOBs* (**B**inary **L**arge **O**bjects) bezeichnet. Solche Daten lassen sich mithilfe der beiden VBA-Funktionen *GetChunk* und *AppendChunk* kopieren.

In dem Modul *ObjektKopieren* der Beispieldatenbank *Kapitel13.mdb* finden Sie die Prozedur *ObjektKopieren*, mit der Sie OLE-Objekte innerhalb einer Tabelle von einem zum anderen Datensatz kopieren können. Die Funktion muss folgendermaßen aufgerufen werden:

```
ObjektKopieren <Tabellenname>,  
<Primärschlüsselfeldname>, <OLE-Feldname>, <Von>, <Nach>
```

Die einzelnen Parameter haben folgende Bedeutungen:

Parameter der Prozedur <i>ObjektKopieren</i>	
Parameter	Bedeutung
<i>Tabellenname</i>	Der Name der Tabelle, in der sich die OLE-Objekte befinden.
<i>Primärschlüsselfeldname</i>	Der Name des Primärschlüsselfeldes der Tabelle.
<i>OLE-Feldname</i>	Der Name des Feldes, in dem die OLE-Objekte gespeichert werden.
<i>Von</i>	Die Primärschlüsselnummer des Datensatzes, dessen Objekt kopiert werden soll.
<i>Nach</i>	Die Primärschlüsselnummer des Datensatzes, in den Sie das Objekt einfügen möchten.

Tabelle 13.5

Löschen von Objekten

Zum Löschen von Objekten stellt die *Action*-Eigenschaft die Aktion *acOLEDelete* zur Verfügung. Diese Aktion funktioniert allerdings nur bei Objektfeldern, die an kein Tabellenfeld gebunden sind. Bei gebundenen Objektfeldern weisen Sie stattdessen dem Feld einfach den Wert *NULL* zu.

Die folgende Beispielprozedur ist an die *Objekt löschen*-Schaltfläche des Beispielformulars *Objekte4* gebunden. Vor dem Löschen muss der Vorgang durch den Anwender bestätigt werden:

```
Private Sub btnLöschen_Click()  
  
Dim Meldung As String
```




```
If Not IsNull(Me!OLEObjekt) Then

    Beep
    Meldung = "Soll das Objekt wirklich gelöscht
werden?"
    If MsgBox(Meldung, vbYesNo, "Objekt Löschen") =
vbYes
        Then

            Me!OLEObjekt = Null

        End If

    End If

End Sub
```

Automation

Im dem vorhergehenden Abschnitt haben Sie das notwendige Rüstzeug kennen gelernt, um OLE-Objekte in Datenbanken einzusetzen. Die Integration lässt jedoch noch etwas zu wünschen übrig. Wenn Sie beispielsweise ein Word-Dokument auf den Drucker ausgeben möchten, müssen Sie es zuerst aktivieren und dann innerhalb von Word den Menüpunkt *Datei > Drucken* auswählen. Mithilfe der Automation (früher OLE-Automation) lassen sich solche und ähnliche Vorgänge automatisieren.

Automation

Die Automation ist Bestandteil von OLE in der Version 2.0. Allerdings unterstützen nicht alle OLE-2-konformen Windows-Anwendungen automatisch auch Automation. Typische Anwendungen, die Automation unterstützen, sind Word ab Version 6.0, Excel ab Version 5.0, PowerPoint ab Version 4.0 und Graph ab Version 5.0.

Was ist Automation?

Neben eingebetteten und verknüpften Objekten stellen automationsfähige OLE-Server noch eine Reihe weiterer Objekte zur Verfügung, die allerdings für den Anwender nicht sichtbar sind. Genauer gesagt sind diese Objekte nur von VBA aus nutzbar und verstecken sich hier hinter Variablennamen. Ein solches Objekt könnte beispielsweise ein bestimmter Zellbereich einer Excel-Tabelle sein. Mithilfe der Automation ist es beispielsweise möglich, auf diesen Zellbereich eine der vielzähligen Excel-Funktionen anzuwenden. So stellt es beispielsweise kein Problem dar, die Schriftart zu verändern oder das Ergebnis einer Kalkulation abzufragen. All dies können Sie problemlos direkt von Access aus steuern, ohne innerhalb von Excel auch nur eine einzelne Programmzeile geschrieben zu haben.



Automationsobjekte erstellen

Eine der wichtigsten Funktionen im Zusammenhang mit der Automation ist die *CreateObject*-Funktion. Sie ermöglicht es, von einer bestimmten Windows-Anwendung wie Excel oder Word ein neues Automationsobjekt anzufordern. Die Funktion erwartet beim Aufruf einen Parameter, der die Klasse des Automationsobjekts angibt. Im Abschnitt *Einfügen neuer Objekte* haben Sie bereits den Klassennamen *Excel.Sheet.10* kennen gelernt, der eine Excel-XP-Tabelle kennzeichnet. Der Name der Klasse besteht aus drei Teilen, wobei der erste mit dem Namen der Anwendung bzw. des OLE-Servers übereinstimmt, von dem Sie das Automationsobjekt anfordern möchten. Einige Anwendungen stellen mehrere Objektklassen zur Verfügung. Bei Excel sind dies unter anderem die Typen *Sheet* (für eine Excel-Tabelle) und *Chart* (für ein Excel-Diagramm). Die gewünschte Klasse geben Sie an zweiter Position an. Als Drittes folgt schließlich noch die Version der Serveranwendung. Diese dürfen Sie jedoch bei Bedarf auslassen (z. B. *Excel.Sheet*). In diesem Fall verwendet Access automatisch die aktuellste Version des Servers, die auf Ihrem Rechner installiert ist.

Um Automationsobjekte innerhalb von VBA zu handhaben, stellt Access einen speziellen VBA-Datentyp mit dem Namen *Object* zur Verfügung. Von der Handhabung lässt sich dieser Datentyp mit den Objektdatentypen vergleichen, die Ihnen bereits vom Zugriff auf Daten bekannt sind. Besonders wichtig ist in diesem Zusammenhang, dass eine Zuweisung an die Variable immer unter Zuhilfenahme der *Set*-Anweisung erfolgen muss. Der folgende Programmausschnitt fordert von Excel eine neue leere Tabelle an:

```
Dim Tabelle As Object
Set Tabelle = CreateObject("Excel.Sheet")
```

Wenn Sie die beiden Programmzeilen innerhalb einer Prozedur angeben und diese Prozedur starten, werden Sie bei der Ausführung der zweiten Zeile eine Verzögerung feststellen. Der Grund dafür ist, dass Excel nun im Hintergrund gestartet wird. Anschließend erstellt Excel hier eine neue Tabelle und stellt diese Access als OLE-Objekt zur Verfügung. Da dieser Vorgang im Hintergrund abläuft, werden Sie abgesehen vom Ladevorgang nichts von dem OLE-Server bemerken. Dies geht so weit, dass Excel noch nicht einmal in der Task-Leiste von Windows erscheint.

Obwohl Excel für den Anwender nicht sichtbar ist, verbraucht es natürlich die üblichen Systemressourcen wie beispielsweise den Hauptspeicher. Dies ist auch der Grund dafür, warum die Automation und OLE im Allgemeinen als "speicherfressend" bezeichnet werden. Neben der Client-Anwendung muss schließlich immer auch die Serveranwendung gestartet werden. Da Sie mehrere verschiedene Objekte gleichzeitig von Access aus anfordern können, kann dies schnell an die Grenzen der Systemressourcen führen. Es gehört daher zu einem guten Programmierstil, dass Sie ein nicht mehr benötigtes Objekt auch wieder schließen und damit den betreffenden OLE-Server aus dem Hauptspeicher entfernen. Dies erledigen Sie, indem Sie der Objektvariablen das Schlüsselwort *Nothing* zuweisen:

```
Dim Tabelle As Object
Set Tabelle = CreateObject("Excel.Sheet")
Set Tabelle = Nothing
```

Der *Object*-
Datentyp

Objekte schließen





Hinweis



Methoden und
Eigenschaften von
Automations-
objekten

Objektvariablen, die Sie auf Prozedurebene deklarieren, werden von Access bei Beendigung der Prozedur automatisch geschlossen.

Auf das Objekt können Sie nun alle von Excel bekannten Eigenschaften und Methoden anwenden. Auch der Verweis auf weitere Objekte, die in dem angeforderten Objekt enthalten sind, ist ohne Einschränkung möglich. Die folgende Beispielprozedur legt ein neues Excel-Objekt an, füllt die erste Spalte mit einigen Werten und speichert die Tabelle anschließend auf Ihrer Festplatte unter dem Namen *C:\Mappe1.xls*. Sie finden die fertige Prozedur im Modul *Automation* der Beispieldatenbank *Kapitel13.mdb*.

Hinweis



Um die folgenden Prozeduren kompilieren zu können, sollten Sie vorher die Objektbibliothek von Excel XP in Ihre Datenbank einbinden. Wählen Sie hierfür den Menübefehl *Extras > Verweise*, nachdem Sie ein Modul geöffnet haben. Im Dialog *Verweise* aktivieren Sie *Microsoft Excel 9.0 Object Library* und klicken auf *OK*.

```
Sub Excell1()  
  
Dim Tabelle As Object  
Dim i As Integer  
  
Set Tabelle = CreateObject("Excel.Sheet")  
Tabelle.Application.Visible = True  
  
For i = 1 To 10  
  
    Tabelle.ActiveSheet.Cells(i, 1).Value = i  
  
Next i  
  
Tabelle.ActiveSheet.SaveAs Filename:="C:\Mappe1.xls"  
  
Set Tabelle = Nothing  
  
End Sub
```

Nachdem Sie die Prozedur ausgeführt haben, befindet sich die neue Datei auf Ihrer Festplatte. Sie können diese nun zur Überprüfung auf die gewohnte Art und Weise mit Excel öffnen.



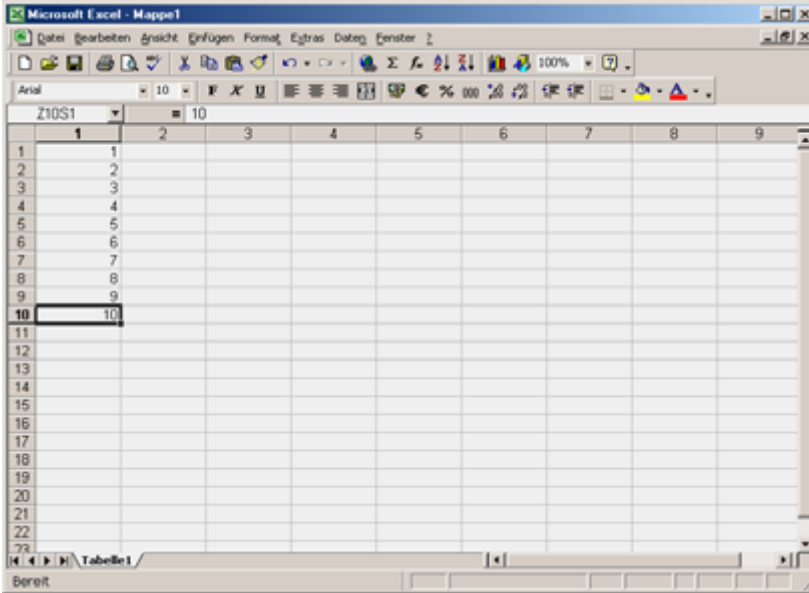


Abb. 13.14

Die Excel-Tabelle
C:\Mappel.xls

Innerhalb der *For...Next*-Schleife wird mithilfe des Objekts *Tabelle* auf ein weiteres Objekt mit dem Namen *ActiveSheet* verwiesen, da mehrere Arbeitsmappen für jede Excel-Datei angelegt werden können. Innerhalb der aktiven Tabelle wird wiederum auf ein Objekt namens *Cells* verwiesen. Dieses Objekt ist stellvertretend für eine bestimmte Zelle. Die Koordinaten der gewünschten Zelle müssen in der Form *<Spalte>*, *<Zeile>* direkt hinter dem Schlüsselwort *Cells* innerhalb der Klammern angegeben werden. Auf dieses *Cells*-Objekt können Sie nun wieder Methoden und Eigenschaften anwenden. In der Beispielprozedur wird z.B. die *Value*-Eigenschaft verwendet, um der Zelle einen Wert zuzuweisen. Dass es sich bei dem *Cells*-Objekt im Prinzip ebenso um ein Automationsobjekt handelt wie bei dem *Sheet*-Objekt, zeigt die folgende Prozedur, die im Gegensatz zur vorhergehenden um eine Variable *Zelle* erweitert wurde:

```
Sub Excellb()

Dim Tabelle As Object
Dim Zelle As Range

Dim i As Integer

Set Tabelle = CreateObject("Excel.Sheet")
Tabelle.Application.Visible = True

For i = 1 To 10

    Set Zelle = Tabelle.ActiveSheet.Cells(i, 1)
```





```
Zelle.Value = i * 2

Next i

Tabelle.SaveAs ("C:\Mappelb.xls")

Set Zelle = Nothing
Set Tabelle = Nothing

End Sub
```

Die Variable *Zelle* wird als Typ *Range* deklariert, der von der Excel-XP-Bibliothek zur Verfügung gestellt wird. Alle Variablen von diesem Typ definieren einen bestimmten Bereich innerhalb einer Excel-Mappe. In diesem Fall wird der Variablen *Zelle* eine bestimmte Excel-Zelle zugewiesen. Der Inhalt der Zelle wird anschließend nicht direkt, sondern mithilfe dieser Variablen verändert.

Das folgende Beispiel zeigt, wie Werte aus einer Excel-Tabelle ausgelesen werden. Die Prozedur füllt wieder die erste Spalte mit zehn Werten. In die elfte Zeile wird eine Formel zur Berechnung der Summe aller zehn Zahlen platziert. Anschließend wird diese Summe ausgelesen und innerhalb von Access in einem Meldungsfenster angezeigt. Die fertige Prozedur finden Sie wieder im Modul *Automation*:

```
Sub Excel2()

Dim Tabelle As Object
Dim i As Integer
Dim Wert As Variant

Set Tabelle = CreateObject("Excel.Sheet")

For i = 1 To 10

Tabelle.ActiveSheet.Cells(i, 1).Value = i

Next i

Tabelle.ActiveSheet.Cells(11, 1).Formula = _
    "=SUM(A1:A10) "

Wert = Tabelle.ActiveSheet.Cells(11, 1).Value

Set Tabelle = Nothing

MsgBox "Das Ergebnis ist " & Str(Wert)

End SubHinweis
```



Das Ergebnis der Berechnung wird der Variablen *Wert* zugewiesen, die den Datentyp *Variant* hat. Dies ist notwendig, da Zellen ohne Inhalt das Ergebnis *NULL* haben. Um Laufzeitfehler zu vermeiden, sollten Sie generell alle Variablen, denen Sie Ergebnisse von OLE-Servern zuweisen möchten, mit dem Datentyp *Variant* deklarieren!

Wie Sie an der Prozedur sehen, wird die Excel-Tabelle nach der Berechnung nicht gespeichert. Dies ist ein gutes Beispiel dafür, wie Sie OLE-Server von VBA aus zur Erledigung von Aufgaben einsetzen, die sich mit Access überhaupt nicht oder nur sehr umständlich erledigen lassen. In der Excel-Tabelle könnten Sie beispielsweise umfangreiche Berechnungen durchführen und anschließend die Ergebnisse abfragen.

Wenn Sie Excel bei der Arbeit beobachten möchten, können Sie es einblenden. Geben Sie dazu folgende Anweisung an:

```
Tabelle.Application.Visible = True
```



Hinweis

Jeder OLE-Server unterstützt eine andere Palette von Objekten, Methoden und Eigenschaften. Eine Auflistung aller Objekte, Methoden und Eigenschaften, die von Excel unterstützt werden, würde den Rahmen dieses Buches sprengen. Eine Übersicht finden Sie in der Regel in der Dokumentation der Serveranwendung.



Hinweis

Bei einigen Methoden, die Sie auf ein Automationsobjekt anwenden, kann es vorkommen, dass Access während der Kompilierung eine Fehlermeldung anzeigt. Dies ist vor allem bei Methoden und Funktionen der Fall, die sowohl Access als auch der OLE-Server unterstützt. In diesem Fall müssen Sie den Methodennamen des OLE-Servers in eckigen Klammern angeben! Das Gleiche gilt für Funktionen, die Sonderzeichen wie z. B. das \$-Zeichen enthalten.



**Eckige Klammern
bei Methoden und
Funktionen**

Ähnlich wie Excel können auch alle Word-Funktionen von Access aus genutzt werden. Mit der Einführung von Office 97 hat Microsoft das Objektmodell von Word grundlegend überarbeitet. Word -Dokumente werden nun beispielsweise als Objekte betrachtet. Gleiches gilt für Absätze, Wörter und markierte Bereiche (*Range*-Objekt).



**Automationsobjekte
mit Word erstellen**

Das folgende Beispiel fügt einen Text in ein neues Word-Dokument ein. Mithilfe der *With*-Anweisung werden die Ränder, Breite und Höhe des Dokuments vorher eingestellt. Danach wird der gesamte Text an Access zurückgegeben und in einem Meldungsfenster angezeigt. Letztendlich wird das Dokument unter dem Namen *C:\Dokument1.doc* gespeichert. Die fertige Prozedur finden Sie in der Beispieldatenbank:

```
Sub Word1()
```

```
Dim objNeuesDokument As Object
Dim objTextBereich As Word.Range
Dim i As Integer
Dim varDerText As Variant
```

```
Set objNeuesDokument = CreateObject("Word.Document")
```





```

objNeuesDokument.Application.Visible = True
objNeuesDokument.Activate

With objNeuesDokument.PageSetup
    .TopMargin = CentimetersToPoints(2)
    .BottomMargin = CentimetersToPoints(2)
    .RightMargin = CentimetersToPoints(2)
    .LeftMargin = CentimetersToPoints(2)
    .PageWidth = CentimetersToPoints(21.59)
    .PageHeight = CentimetersToPoints(27.94)
End With

Set objTextBereich = objNeuesDokument.Range

For i = 1 To 10
    objTextBereich.InsertAfter "Textzeile #" & i
    objTextBereich.InsertParagraphAfter
Next i

varDerText = objNeuesDokument.Range.Text
MsgBox varDerText

objNeuesDokument.SaveAs "C:\Dokument1.doc"

Set objTextBereich = Nothing
Set objNeuesDokument = Nothing

End SubHinweis

```

In dem obigen Beispiel ist die Variable *objTextBereich* mit dem Datentyp *Word.Range* deklariert. Dies ist notwendig, falls Sie im Verweisdialog neben Word auch Excel aktiviert haben. Da beide OLE-Server ein *Range*-Objekt zur Verfügung stellen, könnte VBA den Datentyp bei der einfachen Angabe *Range* nicht eindeutig zuordnen.

Tipp



Wenn Sie längere Textbereiche aus Word an Access übergeben möchten, kann es zu Problemen kommen. In diesem Fall sollten Sie den Text in die Zwischenablage kopieren und dann wieder innerhalb von VBA in ein Textfeld einfügen. Über das Textfeld können Sie dann auf den gesamten Text zugreifen.

Verweise zu OLE-Servern

Im Kapitel 3 *Die Programmiersprache VBA* im Abschnitt *Datenbankreferenzen* haben Sie bereits den *Verweise*-Dialog kennen gelernt, den Sie von einem beliebigen Modulfenster aus über den Menüpunkt *Extras > Verweise* öffnen. Neben Datenbanken können Sie über diesen Dialog auch OLE-Typenbibliotheken in die aktuelle Datenbank einbinden.



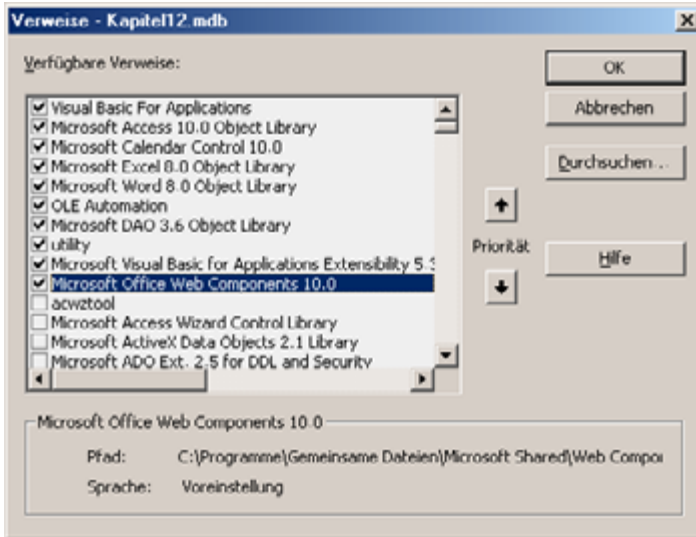


Abb. 13.15

Der *Verweise*-Dialog von Access

Der Vorteil ist hierbei, dass Sie die entsprechenden Objekte innerhalb von VBA mithilfe des *New*-Schlüsselworts der Anweisungen *Set* oder *Dim* anlegen. Dies funktioniert allerdings nur bei bestimmten OLE-Servern. Im Einzelfall müssen Sie durch Probieren herausfinden, ob das Anlegen neuer Objekte mithilfe des *New*-Schlüsselworts möglich ist oder ob Sie weiterhin auf die *CreateObject*-Anweisung zurückgreifen müssen. Ein Beispiel für ein Automationsobjekt, das mithilfe des *New*-Schlüsselworts erzeugt werden kann, ist ein Word-Dokument. Zur Anlage des Dokuments stehen Ihnen zwei Varianten zur Verfügung:

Variante 1:

```
Set objNeuesDokument = New Word.Document
```

Variante 2:

```
Dim objNeuesDokument As Word.Document
Set objNeuesDokument = New Word.Document
```

Auch bei Excel ist der Einsatz des *New*-Schlüsselworts zulässig. Genaue Informationen über seine Verwendung finden Sie im *Kapitel 9 Objektorientierte Programmierung*.

Ein weiterer Vorteil des Einbindens von OLE-Servern über den *Verweise*-Dialog ist die Möglichkeit, die Konstanten und Objektklassen, die von dem OLE-Server zur Verfügung gestellt werden, in VBA zu verwenden. Außerdem können Sie die einzelnen Elemente des referenzierten OLE-Servers mithilfe des Objektkatalogs begutachten und sich bei der Eingabe von Programmcode durch automatische Eingabehilfen unterstützen lassen.





Verweise via VBA-Code handhaben

Neu in Access ist die Möglichkeit, die Referenzen auf OLE-Servern der aktuellen Datenbank mit VBA zu manipulieren. Die Auflistung *References* enthält einige Eigenschaften, die Auskunft über jede Referenz bereitstellen. Die folgende Prozedur listet beispielsweise alle aktuellen Referenznamen und deren Typen im Testfenster auf.

```
Sub VerweiseAuflisten()  
  
Dim Verweis As Reference  
  
    For Each Verweis In References  
  
        Debug.Print Verweis.Name, IIf(Verweis.Kind =  
1, _  
        "VB-Projekt-Bibliothek", "Klassenbibliothek")  
  
    Next Verweis  
  
End Sub
```

Mit der Eigenschaft *IsBroken* prüfen Sie die Gültigkeit einer Referenz. Alle ungültigen Referenzen und deren Pfadnamen werden in der folgenden Prozedur im Testfenster aufgelistet. Sie könnten dann beispielsweise die fehlerhafte Referenz mittels *Remove*-Methode aus der *References*-Auflistung entfernen.

```
Sub UngültigeVerweiseEntfernen()  
  
Dim Verweis As Reference  
  
    For Each Verweis In References  
  
        If Verweis.IsBroken Then  
            With Verweis  
  
                Debug.Print "Der Verweis " & .Name _  
                & " ist ungültig. Pfad: " & .FullPath  
  
                References.Remove Verweis  
  
            End With  
        End If  
  
    Next Verweis  
  
End Sub
```





Automationsobjekte aus Dateien erstellen

Neben der Möglichkeit, neue leere Automationsobjekte von einer Serveranwendung anzufordern, können Sie auch Objekte anfordern, die bereits in Form einer Datei auf Ihrer Festplatte vorliegen. Dies ist beispielsweise dann sinnvoll, wenn Sie aus einer komplexen Excel-Tabelle einige Werte abfragen möchten, um diese in einer Access-Datenbank zu speichern.

Access stellt zu diesem Zweck eine Funktion mit dem Namen *GetObject* zur Verfügung, die Sie auf die gleiche Weise wie die *CreateObject*-Funktion anwenden können. Zusätzlich zum Namen der OLE-Klasse muss jedoch der Name der Datei angegeben werden, auf der das Objekt basiert. Die Funktion besitzt folgende Syntax:

```
<Objektvariable> = GetObject  
( <Dateiname> [, <Klassenname> ] )
```

Die Angabe des Klassennamens ist optional. Wenn Sie ihn auslassen, versucht Windows anhand der Dateierweiterung den passenden OLE-Server ausfindig zu machen.

Das folgende Beispiel finden Sie in der Beispieldatenbank im Modul *Automation*. Die Prozedur legt auf der Grundlage der im vorhergehenden Abschnitt gespeicherten Tabelle *C:\Mappel.xls* ein Automationsobjekt an und zeigt den Inhalt der Zelle A7 an.

```
Sub Excel3 ()  
  
Dim Tabelle As Object  
Dim Wert As Variant  
  
    Set Tabelle = GetObject("C:\mappel.xls",  
"Excel.Sheet")  
  
    Wert = Tabelle.ActiveSheet.Cells(7, 1).Value  
  
    Set Tabelle = Nothing  
  
    MsgBox "Inhalt der Zelle A7: " & Str(Wert) & "."  
  
End SubHinweis
```

Die Funktion *GetObject* ist mit Vorsicht zu genießen, da sie nicht von allen automationsfähigen OLE-Servern unterstützt wird.

Das folgende Beispiel zeigt, wie man mit *GetObject* den Text eines vorhandenen Word-Dokuments abfragen kann. Das Beispiel verwendet das im vorhergehenden Abschnitt angelegte Dokument *Dokument1.doc*:





```
Sub Word2()

Dim objDokument As Object
Dim objTextBereich As Object

Set objDokument = GetObject("C:\Dokument1.doc" _
    , "Word.Document")

Set objTextBereich = objDokument.Range

MsgBox objDokument.Range.Text

Set objTextBereich = Nothing
Set objDokument = Nothing

End Sub
```

Arbeiten mit eingebetteten und verknüpften Objekten

Besonders hilfreich ist die Arbeit mit der Automation, wenn Sie mit Objekten arbeiten, die sich in einem Objektfeld eines Formulars befinden. So ist es beispielsweise ohne Probleme möglich, ein Excel-Objekt direkt von Access aus auf dem Drucker auszugeben.

Die *Object*-
Eigenschaft

Um auf die Automationsschnittstelle eines eingebetteten oder verknüpften OLE-Objekts zugreifen zu können, stellen Objektfelder eine Eigenschaft mit dem Namen *Object* zu Verfügung, die nicht mit dem Datentyp *Object* verwechselt werden sollte. Das Ergebnis der Eigenschaft ist ein Automationsobjekt, das sich genau wie die Automationsobjekte der *CreateObject*- bzw. *GetObject*-Funktionen verhält.

Die folgende Beispielprozedur druckt ein Excel-Objekt aus, das sich in einem Objektfeld mit dem Namen *OLEObjekt* befindet. Sie finden das Beispiel in dem Formular *Automation1* der Datenbank *Kapitel3.mdb*:

```
Private Sub btnDrucken_Click()

Dim ExcelObjekt As Object

Me!OLEObjekt.Verb = acOLEVerbHide
Me!OLEObjekt.Action = acOLEActivate

Set ExcelObjekt = Me!OLEObjekt.Object

ExcelObjekt.ActiveSheet.PrintOut

Me!OLEObjekt.Action = acOLEClose

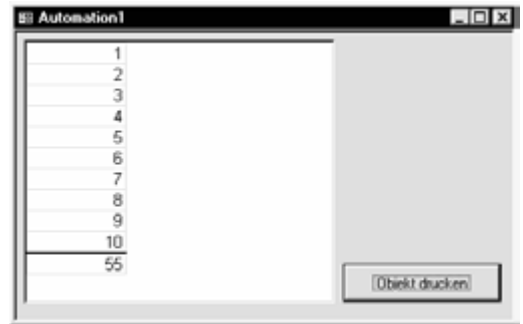
End Sub
```



Das OLE-Objekt wird in diesem Beispiel mithilfe des Verbs *acOLEVerbHide* aktiviert, wodurch Excel komplett im Hintergrund aktiviert wird.

Abb. 13.16 ○

Das Formular
Automation1



In den beiden Beispielen wurden jeweils ungebundene Objektfelder verwendet, bei denen die *Aktiviert*-Eigenschaft auf *Ja* und die *Gesperrt*-Eigenschaft auf *Nein* eingestellt ist. Stattdessen können Sie natürlich auch gebundene Objektfelder einsetzen, die an ein Tabellenfeld gebunden sind.



Hinweis

Access per Automation steuern

Access kann nicht nur als OLE-Client, sondern auch als OLE-Server arbeiten und damit seine Funktionen anderen automationsfähigen Windows-Anwendungen zur Verfügung stellen.

Der Klassenname von Access, der für den Zugriff benötigt wird, lautet *Access.Application*. Um beispielsweise von Visual Basic 4/5/6 aus auf Access zuzugreifen, können Sie folgende Programmzeilen verwenden:

```
Dim AccessObj As Object
Set AccessObj = CreateObject("Access.Application")
```

Das Beispiel definiert zunächst eine Objektvariable mit dem Namen *AccessObj* und weist dieser anschließend ein Objekt zu, das mithilfe der *CreateObject*-Anweisung neu erzeugt wurde. Bei diesem Vorgang wird Access minimiert gestartet.

Das von der *CreateObject*-Anweisung zurückgegebene Objekt ist mit dem *Application*-Objekt von Access identisch. Sie haben also die Möglichkeit, über dieses Objekt auf alle anderen Access-Objekte zuzugreifen.

Wenn Sie ein Access-Objekt mittels *CreateObject* erstellen, ist noch keine Datenbank geöffnet. Um dies nachzuholen, können Sie anschließend die Methoden *OpenCurrentDatabase* oder *NewCurrentDatabase* anwenden. Das Schließen der Datenbank bewerkstelligen Sie mittels *CloseCurrentDatabase*. Alle Methoden werden direkt von dem *Application*-Objekt unterstützt.





Das folgende VisualBasic-Beispiel erzeugt ein Access-Objekt und öffnet die Datenbank *C:\DB1.MDB*. Anschließend wird der Bericht mit dem Namen *Bericht1* auf dem Drucker ausgegeben und die Datenbank geschlossen.

```
Private Sub btnBerichtDrucken_Click()  
  
Dim AccessObj As Object  
  
    Set AccessObj = CreateObject("Access.Application")  
  
    With AccessObj  
  
        .OpenCurrentDatabase filepath:="C:\DB1.MDB" _  
            , Exclusive:=False  
  
        .DoCmd.OpenReport ReportName:="Bericht1" _  
            , View:=acViewNormal  
  
        .CloseCurrentDatabase  
  
    End With  
  
    Set AccessObj = Nothing  
  
End Sub
```

Der Ausdruck des Berichts erfolgt mithilfe der *OpenReport*-Methode, die von dem *DoCmd*-Objekt zur Verfügung gestellt wird. Als Parameter *View* wird hierbei die Konstante *acViewNormal* verwendet, die von Access definiert wird. Damit Visual Basic die Konstante bekannt ist, muss Access über den Dialog *Extras > Verweise* bzw. *Projekt > Verweise* von Visual Basic eingebunden werden. Dieser Dialog ist mit dem *Verweise*-Dialog von Access identisch. Markieren Sie hier den Eintrag *Microsoft Access 9.0 Object Library*.

Wenn Sie die Referenz zu Access hergestellt haben, können Sie das Access-Objekt auch mithilfe des *New*-Schlüsselworts anlegen:

```
Private Sub Command1_Click()  
  
Dim AccessObj As Access.Application  
  
    Set AccessObj = New Access.Application  
  
    AccessObj.OpenCurrentDatabase "C:\DB1.MDB"  
    AccessObj.DoCmd.OpenReport ReportName:="Bericht1", _  
        View:=acNormal  
    AccessObj.CloseCurrentDatabase  
  
End Sub
```



Der Ausdruck eines Berichts einer bestimmten Datenbank lässt sich mithilfe der *GetObject*-Anweisung weiter vereinfachen. Neben den Klassennamen wird hierbei einfach der Name einer Datenbankdatei angegeben, die dann von Access automatisch geöffnet wird:

```
Private Sub Command1_Click()

Dim AccessObj As Access.Application

    Set AccessObj = GetObject("C:\DB1.MDB", _
        "Access.Application")

    AccessObj.DoCmd.OpenReport ReportName:="Bericht1", _
        View:=acNormal

End Sub
```

ActiveX-Steuerelemente

Eng mit OLE und der Automation verwandt sind die so genannten *ActiveX-Steuerelemente*, die auch als *ActiveX-Controls* bezeichnet werden. ActiveX-Steuerelemente dienen dazu, die Palette der Access-Steuerelemente zu erweitern. Technisch gesehen sind ActiveX-Controls kleine OLE-Server. Durch den geringen Umfang der Programmdateien sind die Ladezeiten des Servers sehr niedrig, so dass die Aktivierung ohne Wartezeit erfolgt.

Zum Standardlieferumfang von Access gehört ein ActiveX-Steuerelement mit dem Namen *Kalender-Steuerelement*, über das sich die Eingabe eines Datums vereinfachen lässt. Das Steuerelement enthält zu diesem Zweck einen Monatskalender, über den ein bestimmtes Datum komfortabel mit der Maus ausgewählt werden kann.



Abb. 13.17

Das Kalender-Steuerelement



ActiveX-Steuerelemente registrieren

Bevor Sie ein ActiveX-Control nutzen können, müssen Sie es zunächst anmelden. In der Regel wird diese Aufgabe bei der Installation der Windows-Anwendung, mit der die Steuerelemente ausgeliefert wurden, automatisch durchgeführt.

In einigen Fällen kann aber eine manuelle Registrierung erforderlich sein. Dies ist beispielsweise dann der Fall, wenn Sie lediglich ein bestimmtes ActiveX-Steuerelement auf einen anderen Computer kopieren möchten.

ActiveX-Steuerelemente werden in der Regel im Windows-Systemverzeichnis gespeichert und tragen die Dateiendung `.ocx`. Unter Access erledigen Sie die Registrierung, indem Sie den Menüpunkt *Extras > ActiveX-Steuerelemente* auswählen. Access öffnet daraufhin das Dialogfenster *ActiveX-Steuerelemente*.

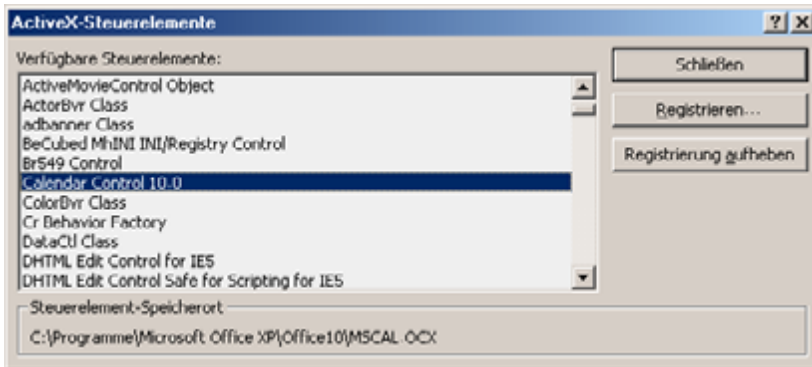


Abb. 13.18

Das Dialogfenster
ActiveX-Steuerelemente

Um ein neues Steuerelement zu registrieren, klicken Sie auf die Schaltfläche *Registrieren*. Es erscheint ein Dateiauswahldialog, in dem Sie nun die OCX-Datei des Steuerelements auswählen können. Nachdem Sie die Dateiauswahl bestätigt haben, erscheint das neue Steuerelement in der Liste.

Die Registrierung eines ActiveX-Steuerelements erfolgt in der Registrierungsdatenbank von Windows. Nach der Registrierung steht das Steuerelement allen Windows-Anwendungen zur Verfügung, die Steuerelemente nutzen können.

Mit der Schaltfläche *Registrierung aufheben* können Sie ein bereits registriertes ActiveX-Steuerelement entfernen. Auch dies wirkt sich systemweit aus.

Hinweis



Wenn Sie Datenbanken weitergeben, in denen Sie ActiveX-Steuerelemente verwenden, müssen Sie dafür sorgen, dass die Steuerelemente auch auf dem Zielrechner zur Verfügung stehen und registriert sind. Gegebenenfalls müssen Sie die entsprechenden OCX-Dateien in das Systemverzeichnis des Zielrechners kopieren und sie auf die oben beschriebene Weise registrieren.



ActiveX-Steuerelemente einfügen

Wenn Ihr ActiveX-Steuerelement erst einmal vorschriftsmäßig angemeldet ist, steht dem Einsatz in Formularen und Berichten nichts mehr im Wege. Öffnen Sie ein Formular oder einen Bericht in der Entwurfsansicht und wählen Sie den Menüpunkt *Einfügen > ActiveX-Steuerelement* aus, um das Dialogfenster *ActiveX-Steuerelement einfügen* zu aktivieren. Es genügt nun ein Doppelklick auf den gewünschten Steuerelementtyp, um das Steuerelement einzufügen.

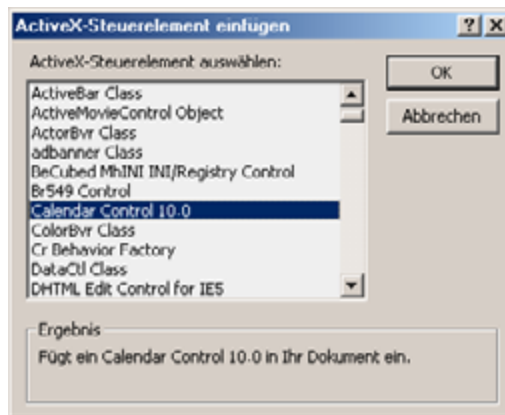


Abb. 13.19

Das Dialogfenster
*ActiveX-Steuerelement
einfügen*

Wenn Sie häufig ActiveX-Steuerelemente einsetzen, können Sie der Toolbox von Access neue Symbole hinzufügen, über die Sie schnell ein bestimmtes Steuerelement einfügen können. Dazu gehen Sie folgendermaßen vor:

- 1 Öffnen Sie ein beliebiges Formular in der Entwurfsansicht und öffnen Sie die Toolbox (z. B. über den Menüpunkt *Ansicht > Toolbox*).
- 2 Wählen Sie den Menüpunkt *Ansicht > Symbolleiste* aus und klicken Sie im Untermenü auf *Anpassen*.
- 3 Wählen Sie die Registerkarte *Befehle* aus.
- 4 Wählen Sie aus der Liste *Kategorien* den Eintrag *ActiveX-Steuerelemente* aus.
- 5 Ziehen Sie das gewünschte ActiveX-Steuerelement aus der Liste *Befehle* in die Toolbox.
- 6 Schließen Sie das Dialogfenster.

In Zukunft genügt ein Klick auf das neue Symbol, um das betreffende ActiveX-Steuerelement einzufügen.



Eigenschaften von ActiveX-Steuerelementen

Beim Entwurf von Formularen können Sie mit ActiveX-Steuerelementen auf die gleiche Weise hantieren wie mit den in Access eingebauten Steuerelementen.

Bearbeiten von
Eigenschaften

Natürlich können Sie für das ActiveX-Steuerelement auch das Eigenschaftenfenster aktivieren. Access zeigt hier neben einigen OLE-typischen Eigenschaften, die Sie bereits im Zusammenhang mit Objektfenstern kennen gelernt haben, auch Eigenschaften an, die speziell auf das Steuerelement zugeschnitten sind. Zur Begutachtung dieser Eigenschaften müssen Sie im Eigenschaftenfenster eines der Register *Andere* oder *Alle* auswählen.

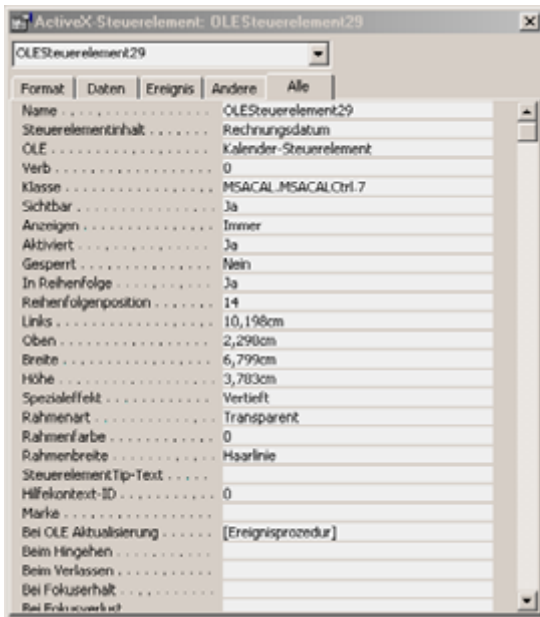


Abb. 13.20

Das Eigenschaftenfenster des *Kalender-*ActiveX-Steuerelements

In einigen Fällen unterstützt ein Steuerelement auch Eigenschaften, die Sie über das Eigenschaftenfenster von Access nicht bearbeiten können. In solch einem Fall klicken Sie doppelt auf das eingefügte Steuerelement. Access zeigt daraufhin ein Dialogfenster an, über das Sie alle Eigenschaften des Steuerelements bearbeiten können.

Das gleiche Fenster können Sie auch über das Kontextmenü oder das Menü *Bearbeiten* aktivieren. Hier finden Sie einen Menüpunkt, der mit dem Namen des ActiveX-Steuerelements übereinstimmt. Wenn Sie beispielsweise das *Kalender-*Steuerelement markieren, lautet der entsprechende Menüpunkt *Kalender-Steuerelement-Objekt*. Dieser Menüpunkt enthält einen Untermenüpunkt mit dem Namen *Eigenschaften*, der schließlich das Eigenschaftenfenster anzeigt.



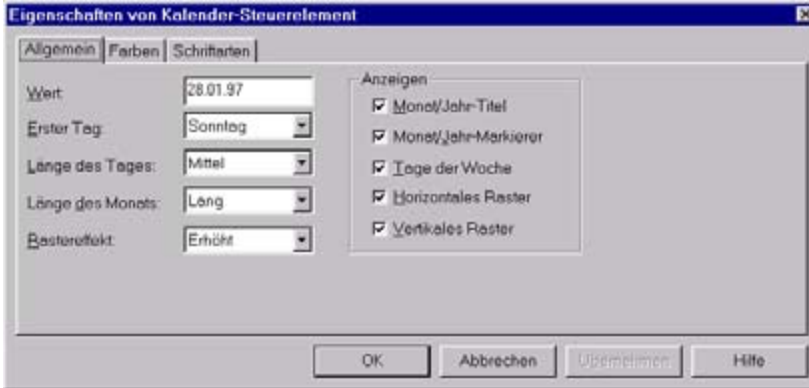


Abb. 13.21

Das spezielle Eigenschaftenfenster des *Kalender-Steuerelements*

Der Aufbau des Eigenschaftenfensters unterliegt keinen festen Bestimmungen und kann von Steuerelement zu Steuerelement variieren. In der Regel werden die Eigenschaften in mehrere Dialogseiten gegliedert.

Der Zugriff auf die Eigenschaften eines ActiveX-Steuerelements von VBA aus unterscheidet sich nicht von der üblichen Vorgehensweise. Das folgende Beispiel ändert die Hintergrundfarbe des *Kalender-Steuerelements*:

```
Private Sub btnWeiß_Click()

    Me!ocxKalender.BackColor = 16777215

End Sub
```

Sie finden die Prozedur im Formular *ActiveX-Steuerelement* der Beispieldatenbank. Die Prozedur ist an das *Click*-Ereignis der Schaltfläche *Hintergrund Weiß* geknüpft. Eine weitere Schaltfläche mit der Beschriftung *Hintergrund Rot* ist mit folgender Ereignisprozedur verknüpft:

```
Private Sub btnRot_Click()

    Dim OLESteuerelement As Control

    Set OLESteuerelement = Me!ocxKalender

    OLESteuerelement.BackColor = 255

End Sub
```

Diese zweite Ereignisprozedur ändert die Hintergrundfarbe in Rot. Die Besonderheit der Prozedur ist hierbei, dass ein Verweis auf das ActiveX-Steuerelement in einer Objektvariablen des Typs *Control* zwischengespeichert wird.





Ereigniseigenschaften von ActiveX-Steuerelementen

Im Bezug auf Ereignisseigenschaften gibt es einen wesentlichen Unterschied zwischen den in Access eingebauten Steuerelementen und ActiveX-Steuerelementen. Im Eigenschaftenfenster des Steuerelements werden nur die folgenden Ereignisseigenschaften aufgeführt:

- *Bei OLE-Aktualisierung*
- *Beim Hingehen*
- *Beim Verlassen*
- *Bei Fokuserhalt*
- *Bei Fokusverlust*

Neben diesen Ereignisseigenschaften definieren ActiveX-Steuerelemente in der Regel weitere Ereignisse, die speziell auf das Steuerelement zugeschnitten sind. Um auf diese Ereignisse eines Steuerelements reagieren zu können, kann nicht auf das Eigenschaftenfenster des Objektfeldes zurückgegriffen werden. Die Lösung besteht darin, dass Sie zunächst das Modulfenster des Formulars öffnen, in dem sich Ihr Steuerelement befindet. In der Symbolleiste wählen Sie anschließend über das linke Kombinationsfeld *Objekt* das Steuerelement aus. Das rechte Kombinationsfeld *Prozedur* enthält daraufhin eine komplette Liste aller Ereignisse, die das Steuerelement unterstützt. Nachdem Sie ein Ereignis ausgewählt haben, erstellt Access den Prozedurrumpf und Sie können wie gewohnt mit der Eingabe der Ereignisprozedur beginnen.

Gebundene ActiveX-Steuerelemente

Einige ActiveX-Steuerelemente können innerhalb eines Formulars oder eines Berichts an ein Datenfeld gebunden werden. Wie alle Steuerelemente stellen auch diese Steuerelemente zu diesem Zweck die Eigenschaft *Steuerelementinhalt* zur Verfügung, über das Sie ein Datenfeld der zugrunde liegenden Datenquelle auswählen können.

Das *Kalender*-Steuerelement ist ein Beispiel für ein solches Steuerelement. Mithilfe der *Steuerelementinhalt*-Eigenschaft können Sie es an ein Datenfeld des Typs *Datum/Zeit* binden. In der Beispieldatenbank zu diesem Kapitel finden Sie das Formular *Rechnungen*, in dem das Rechnungsdatum mithilfe des *Kalender*-Steuerelements begutachtet oder verändert werden kann.



Rechnungen

Rechnungs-Nr: 3
 Kunden-Code: 3
 Personal-Nr: 4
 Auslieferungs-Nr: 3
 Status: Erledigt
 Rechnungsdatum: 02.03.77
 Verkaufsbeizer: Herr Müller
 Versandsdatum: 01.12.95
 Versand an:
 Versandart: LIPS
 Versandkosten: 29.00 DM
 Anmerkung:
 Rechnungsbetrag: 435,00 DM
 MwSt: 15,00%

Calendar: März 1977

So	Mo	Di	Mi	Do	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Datensatz: 1 von 8

Abb. 13.22

Das Formular
Rechnungen

DDE

Während moderne Windows-Anwendungen per Automation miteinander kommunizieren, muss bei vielen älteren Windows-Anwendungen immer noch auf die veraltete DDE-Technik zurückgegriffen werden.

DDE ist eine Abkürzung für den englischen Begriff **D**ynamic **D**ata **E**xchange, was auf Deutsch so viel wie "dynamischer Datenaustausch" bedeutet. Das Prinzip von DDE basiert darauf, dass eine Windows-Anwendung zu einer anderen Windows-Anwendung einen Übertragungskanal – den *DDE-Kanal* – aufbaut. Über diesen Kanal können bestimmte Daten angefordert und übertragen werden. Diejenige Windows-Anwendung, die die Initiative zum Kanalaufbau ergreift, wird auch als DDE-Client bezeichnet. Die andere Windows-Anwendung arbeitet dagegen als DDE-Server.

Server und Client

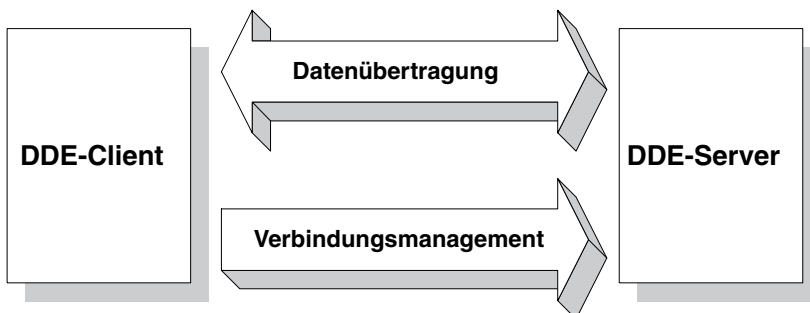


Abb. 13.23

DDE-Verbindung zwischen Client und Server



Viele Windows-Anwendungen können sowohl als Client als auch als Server arbeiten. Dies trifft natürlich auch auf Access zu. So ist es beispielsweise kein Problem, in Access Daten aus der Textverarbeitung Word anzufordern. Andersherum ist es aber auch kein Problem, von Word aus eine Verbindung zu Access aufzubauen, um hier beispielsweise eine Adresse abzufragen.

In den folgenden Abschnitten lernen Sie die Arbeit mit Access als DDE-Client und als DDE-Server kennen.

Access als DDE-Client

Wenn Access als DDE-Client arbeitet, übernimmt es damit den aktiven Teil beim Aufbau und bei der Abwicklung der Kommunikation. Sowohl der Aufbau einer DDE-Verbindung als auch der Datenaustausch lassen sich von VBA aus durchführen. Dazu stellt Access einige spezielle Funktionen und Anweisungen zur Verfügung.

Verbindungs-
management

Eine der wichtigsten DDE-Funktionen ist die *DDEInitiate*-Funktion. Mithilfe der Funktion kann ein Übertragungskanal zu einer anderen Windows-Anwendung aufgebaut werden. Die Funktion besitzt den folgenden Aufbau:

```
<KanalNummer> = DDEInitiate(<Application>, <Topic>)
```

Unter *<Application>* wird die Kennung der Windows-Anwendung angegeben, zu der Sie einen DDE-Kanal öffnen möchten. Diese Kennung stimmt in der Regel mit dem Namen der *.exe*-Datei überein, mit der die betreffende Windows-Anwendung gestartet wird. Die folgende Tabelle enthält die Kennungen einiger Windows-Anwendungen:

Kennungen einiger Windows-Anwendungen

Anwendung	Kennung
Microsoft Access	<i>MSAccess</i>
Microsoft Excel	<i>Excel</i>
Microsoft FoxPro	<i>FoxPro</i>
Microsoft Project	<i>WinProj</i>
Microsoft Word for Windows	<i>WinWord</i>
Explorer	<i>ProgMan</i>

Tabelle 13.6



<Topic> kennzeichnet ein Objekt der Serveranwendung, auf das Sie bestimmte DDE-Befehle ausführen möchten. Dies kann beispielsweise der Name einer bestimmten Excel-Tabelle oder eines Word-Dokuments sein. Viele Windows-Anwendungen unterstützen zudem das Topic *System*, das unabhängig von einer Datei die direkte Kommunikation mit der Anwendung erlaubt.

Wenn die Verbindung aufgebaut werden konnte, gibt die Funktion einen Wert vom Datentyp *LONG* zurück, der auch als *Kanalnummer* bezeichnet wird. Diese Nummer muss angegeben werden, wenn Sie Daten über den Kanal versenden möchten. Das folgende Beispiel stellt eine DDE-Verbindung zu Excel her und speichert die Nummer des Kanals in der Variablen *DDEKanal*:

```
Dim DDEExcelKanal As Long
DDEKanal = DDEInitiate("Excel", "System")
```

Damit die Verbindung hergestellt werden kann, muss die betreffende Windows-Anwendung – in diesem Fall also Excel – bereits gestartet sein. Ansonsten tritt ein Laufzeitfehler mit der Fehlernummer 282 auf. Diesen Fehler können Sie abfangen, um dann den Start der Anwendung mithilfe der *Shell*-Anweisung nachzuholen. Anschließend kann dann erneut versucht werden, den Kanal zu öffnen. Die Prozedur finden Sie in dem Modul *DDE* der Beispieldatenbank *Kapitel13.mdb*:

Verbindung
herstellen

```
Sub DDE1()

Dim DDEExcelKanal As Long
Dim TaskID As Long

KanalÖffnen:

    On Error GoTo Err

    DDEExcelKanal = DDEInitiate("Excel", "System")

    MsgBox "Der DDE-Kanal wurde erfolgreich aufgebaut!"

    DDETerminate DDEExcelKanal

Exit Sub

Err:

    If Err.Number = 282 Then

        TaskID = Shell("C:\Programme\Microsoft _
                        Office\Office\Excel.exe", 4)
        Resume KanalÖffnen

    Else

        MsgBox Err.Description
```



```
DDETerminateAll
Exit Sub
```

```
End If
```

```
End SubHinweis
```

Im Zusammenhang mit der *Shell*-Anweisung müssen Sie den kompletten Pfad zur Startdatei der Windows-Anwendung angeben. Da das Installationsverzeichnis von Anwender zu Anwender unterschiedlich sein kann, sollten Sie den kompletten Dateinamen inklusive Pfad einer Konstanten zuweisen, die Sie dann je nach Bedarf anpassen können. Ist der Pfadname des DDE-Servers in der Datei *AUTOEXEC.BAT* unter der *PATH*-Umgebungsvariablen angegeben, reicht in der Regel die Angabe der *.exe*-Datei ohne Pfad aus.

Nachdem in der Beispielprozedur *DDE1* erfolgreich ein DDE-Kanal aufgebaut wurde, gibt die Prozedur mithilfe der *MsgBox*-Anweisung eine Meldung aus. Anstelle dieser Meldung könnten Sie nun bereits Daten über den Kanal übertragen. Doch dazu später mehr.

Verbindung schließen

Zunächst ist die nachfolgende Anweisung *DDETerminate* interessant, die den DDE-Kanal nach erledigter Arbeit wieder schließt. Dies ist notwendig, da Windows nicht unbegrenzt viele DDE-Kanäle gleichzeitig öffnen kann. Wenn Sie also mehrmals hintereinander einen Kanal öffnen, aber nicht wieder schließen, kann Windows schnell an die Grenze der Ressourcen stoßen. Als Parameter erwartet die *DDETerminate*-Anweisung lediglich die Nummer des zu schließenden Kanals.

Alle Verbindungen schließen

Neben der *DDETerminate*-Anweisung stellt Access noch eine weitere Anweisung mit dem Namen *DDETerminateAll* zur Verfügung. Diese Anweisung schließt alle DDE-Kanäle, die von VBA aus geöffnet wurden. Eingesetzt wird diese Anweisung häufig in Fehlerbehandlungsroutinen zur Freigabe aller DDE-Ressourcen.

Daten übertragen

Ein geöffneter DDE-Kanal ist die wichtigste Voraussetzung zur Übertragung von Daten. Die eigentliche Übertragung wird mit vier speziellen VBA-Funktionen bzw. Anweisungen erledigt:

Übertragen von Daten über einen DDE-Kanal

Funktion/Anweisung	Beschreibung
<i>DDEExecute</i>	Führt in der Serveranwendung einen Befehl aus.
<i>DDERequest</i>	Fordert Daten von dem DDE-Server an.
<i>DDEPoke</i>	Sendet Daten an den DDE-Server.

Tabelle 13.7

Die *DDEExecute*-Anweisung hat die Aufgabe, gezielt einen bestimmten Befehl im DDE-Server auszuführen. Die Anweisung hat folgende Syntax:

```
DDEExecute <ChanNum>, <Command>
```

Mit *<ChanNum>* bestimmen Sie den Kanal, über den die Daten übertragen werden sollen. *<Command>* ist eine Zeichenkette, die den eigentlichen Befehl enthält. Wie diese Zeichenkette aufgebaut sein muss, hängt von dem jeweiligen DDE-Server ab. Um beispielsweise eine bestimmte Excel-XP-Tabelle zu öffnen, geben Sie folgende Anweisung an:

```
DDEExecute DDEExcelKanal, "[Open("C:\Mappel.xls")]"
```

Der DDE-Befehl wird in diesem Beispiel von eckigen Klammern eingeklammert. Dies ist eine Konvention, die speziell auf Microsoft-Anwendungen wie Excel oder Word zugeschnitten und bei der Kommunikation mit anderen Windows-Anwendungen eventuell nicht erforderlich ist. Die doppelten Anführungszeichen dienen dazu, ein Anführungszeichen in die Zeichenkette einzubetten, die Access normalerweise als Einleitung einer Zeichenkette interpretiert. Excel empfängt also letztendlich den folgenden Befehl:

```
[Open("C:\Mappel.xls")]
```

Berücksichtigen Sie, dass es sich bei dem *Open*-Befehl um einen Befehl der Excel-Makrosprache handelt. Auf die gleiche Weise können Sie auch andere Makrobefehle von Excel ausführen.

Bei einer DDE-Verbindung zu Word für Windows XP können Sie ebenfalls sämtliche Befehle ausführen. Das folgende Beispiel öffnet das Word-Dokument mit dem Namen *C:\Dokument1.doc* und fügt hier einen Text ein. Die vollständige Prozedur inklusive Fehlerbehandlungscode finden Sie in dem Modul *DDE* unter dem Namen *DDE2*.

```
DDEWordKanal = DDEInitiate("WinWord", "System")
DDEExecute DDEWordKanal, "[FileOpen _
    "C:\Dokument1.doc"]"
DDEExecute DDEWordKanal, "[Insert ""Dies ist ein _
    DDE-Test""]"
DDETerminate DDEWordKanal
```

Word und Excel erlauben die Ausführung mehrerer Befehle in einem Durchgang. Dazu geben Sie die einzelnen Befehle direkt hintereinander an, wobei Sie jeden Befehl wie in dem folgenden Beispiel in eckige Klammern einschließen:

```
DDEExecute DDEWordKanal, "[FileOpen _
    "C:\Dok1.doc"] [FileOpen "C:\Dok2.doc"]]"
```

Das Versenden von Daten kann ebenfalls mithilfe des *DDEExecute*-Befehls erledigt werden. So wird in dem vorhergehenden Word-Beispiel *DDE2* mithilfe des *EINFÜGEN*-Befehls ein Text in ein Dokument eingefügt. In Bezug auf Word ist das auch





die praktikabelste Lösung, da Sie mithilfe der anderen Word-Basic-Befehle auch die notwendige Kontrolle über die Position der Einfügemarke haben. Zum direkten Übertragen von Daten besitzt Access eine spezielle Anweisung mit dem Namen *DDEPoke*:

```
DDEPoke <ChanNum>, <Item>, <Data>
```

<ChanNum> ist Ihnen bereits von der *DDEExecute*-Anweisung her bekannt. *<Item>* gibt an, wohin die Daten geschrieben werden sollen. Die Angabe muss als Zeichenkette erfolgen, deren Aufbau von der jeweiligen Serveranwendung abhängig ist. Bei Excel können Sie über dieses Argument beispielsweise eine bestimmte Tabellenzelle angeben, in die Sie einen Wert schreiben möchten. Die Zeichenkette *Z1S2* verweist auf die erste Zeile der zweiten Spalte – also auf die Zelle B1. Welche Daten über den DDE-Kanal versendet werden sollen, bestimmt das letzte Argument mit dem Namen *<Data>*. Die Daten müssen ebenfalls in Form einer Zeichenkette angegeben werden.

Das folgende Beispiel fügt per *DDEPoke*-Anweisung einen Text in eine Tabelle einer Excel-Mappe ein. Die Prozedur befindet sich in dem Modul *DDE* der Beispieldatenbank:

```
Sub DDE3 ()

Dim DDEExcelKanal1 As Long
Dim DDEExcelKanal2 As Long
Dim Tabellename As String
Dim Themenliste As String
Dim TaskID As Long

Start_ExcelDDE:

    On Error GoTo Err_ExcelDDE

    DDEExcelKanal1 = DDEInitiate("Excel", "System")

    Themenliste = DDERequest(DDEExcelKanal1, "Topics")

    Tabellename = Left(Themenliste, InStr(1, _
        Themenliste, Chr(9)) - 1)

    DDEExcelKanal2 = DDEInitiate("Excel", Tabellename)

    DDEPoke DDEExcelKanal2, "Z1S2", "Dies ist ein _
        DDE-Test!"

    DDETerminate DDEExcelKanal2

    DDETerminate DDEExcelKanal1

Exit_ExcelDDE:
```



```

Exit Sub

Err_ExcelDDE:

If Err.Number = 282 Then

    " Hier evtl. den Pfad anpassen
    TaskID = Shell("C:\Programme\Microsoft _
                  Office\Office\Excel.exe", 4)
    Resume Start_ExcelDDE

Else

    MsgBox Err.Description
    DDETerminateAll
    Resume Exit_ExcelDDE

End If

End Sub

```

Die Prozedur baut zunächst eine DDE-Verbindung zu Excel auf. Hierbei wird als Thema *System* angegeben. Dieses Thema erlaubt es, eine neue Excel-Arbeitsmappe mit einer neuen Tabelle anzulegen. Um nun auf diese Tabelle zugreifen zu können, muss ein zweiter DDE speziell zu dieser Tabelle aufgebaut werden. Dies geschieht wieder mithilfe der *DDEInitiate*-Funktion, wobei als Thema diesmal der Name der neuen Arbeitsmappe und der Tabelle angegeben wird. Excel vergibt jedoch beim Anlegen einer neuen Arbeitsmappe immer neue Namen, so dass der aktuelle Name von Excel in Erfahrung gebracht werden muss.

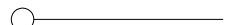
Das Thema *System* stellt zu diesem Zweck ein Element mit dem Namen *Topics* zur Verfügung, das die Namen aller geöffneten Arbeitsmappen als Zeichenkette zurückgibt. Die einzelnen Namen sind hierbei durch ein Tabulatorzeichen (*Chr(9)*) voneinander abgegrenzt. Das Abfragen dieser Topics erledigt die dritte DDE-Funktion mit dem Namen *DDERequest*:

```
<Daten> = DDERequest ( <ChanNum>, <Item> )
```

Die Parameter dieser Funktion stimmen voll mit denen der *DDEPoke*-Funktion überein. In der Beispielprozedur wird diese Funktion dazu verwendet, die Liste aller Themen von Excel abzurufen und in der Variablen *Themenliste* zu speichern. Im zweiten Schritt wird der erste Arbeitsmappenname der Liste mithilfe der *Left*-Funktion abgeschnitten und in der Variablen *Tabellenname* abgelegt. Der Inhalt der Variablen könnte wie folgt aussehen:

```
[Mappel.XLS]Tabelle1
```

Über die *DDEInitiate*-Funktion wird anschließend ein zweiter DDEKanal geöffnet, der sich auf die Tabelle bezieht. Über diesen Kanal können nun die Daten per *DDEPoke* übertragen werden.

 Daten anfordern





Das folgende Beispiel verwendet die *DDERequest*-Funktion dazu, Daten aus mehreren Excel-Zellen abzufragen und in einem Meldungsfenster darzustellen. Damit Sie dieses Beispiel nachvollziehen können, sollten Sie vorher die Excel-Datei *C:\Mappe1.xls* anlegen und die erste Spalte der Tabelle *Tabelle1* mit einigen Werten füllen.

```
Sub DDE4()  
  
Dim DDEExcelKanal1 As Long  
Dim DDEExcelKanal2 As Long  
Dim TaskID As Long  
Dim i As Integer  
Dim Inhalt As Variant  
Dim Zelle As String  
Dim Meldung As String  
  
Start_ExcelDDE:  
  
On Error GoTo Err_ExcelDDE  
  
DDEExcelKanal1 = DDEInitiate("Excel", "System")  
  
DDEExecute DDEExcelKanal1, "[Open(\"C:\Mappe1.XLS\")]"  
  
DDEExcelKanal2 = DDEInitiate("Excel", _  
    "[Mappe1.XLS]Tabelle1")  
  
For i = 1 To 10  
  
    Zelle = "Z" & Trim(Str(i)) & "S1"  
  
    Inhalt = DDERequest(DDEExcelKanal2, Zelle)  
  
    If Not IsNull(Inhalt) Then  
  
        Meldung = Meldung & Zelle & ": " & Inhalt  
  
    End If  
  
Next i  
  
MsgBox Meldung, vbInformation, "Tabelleninhalt"  
  
DDETerminate DDEExcelKanal2  
DDETerminate DDEExcelKanal1
```



```

Exit_ExcelDDE:
    Exit Sub

Err_ExcelDDE:

    If Err.Number = 282 Then

        " Hier evtl. den Pfad anpassen
        TaskID = Shell("C:\Programme\Microsoft _
                        Office\Office\Excel.exe", 4)
        Resume Start_ExcelDDE

    Else

        MsgBox Err.Description
        DDETerminateAll
        Resume Exit_ExcelDDE

    End If

End SubHinweis

```

Da in diesem Beispiel eine vorhandene Tabelle geöffnet wird, ist der Name der Arbeitsmappe bekannt und muss nicht extra von Excel angefordert werden!

Access als DDE-Server

Neben der Möglichkeit, Access als DDE-Client einzusetzen, kann das Programm auch von anderen Windows-Anwendungen aus per DDE als Server angesprochen werden. Im Folgenden wird dies anhand von Beispielen erläutert, die mit der Programmiersprache *VisualBasic* und mit *WordBasic* entwickelt wurden.

Access unterstützt folgende Themen (*Topics*), die von anderen Anwendungen beim Aufbau einer DDE-Verbindung genutzt werden können: *System*, den Namen einer Datenbank, einer Tabelle oder einer Abfrage oder einen beliebigen SQL-Text.

Das Thema *System* ist ein allgemeines Thema, das bei allen Windows-Anwendungen zur Verfügung steht. Es stellt eine Reihe von Elementen zur Verfügung, die von anderen Anwendungen genutzt werden können. Ein Beispiel ist das *Systems*-Element. Wenn ein DDE-Server über eine *DDERequest*-Funktion auf das *Systems*-Element Bezug nimmt, gibt Access eine Liste aller unterstützten Themen zurück. Die Liste wird als Zeichenkette zurückgegeben, wobei die einzelnen Listeneinträge durch Ausrufezeichen (!) voneinander abgegrenzt werden.

Das Thema *System*



Die folgende Tabelle enthält alle unterstützten Elemente des Themas *System*:

Elemente des Themas *System*

Element	Beschreibung
<i>Systems</i>	Eine Liste aller von Access unterstützten Themen.
<i>Formats</i>	Eine Liste aller Formate, in denen Access Daten in der Zwischenablage ablegen kann.
<i>Status</i>	Gibt an, ob Access weitere DDE-Befehle entgegennehmen kann. Das Ergebnis ist eine der beiden Zeichenketten <i>Busy</i> oder <i>Ready</i> .
<i>Topics</i>	Eine Liste aller geöffneten Datenbanken.

Tabelle 13.8

Sobald eine Verbindung zu dem Thema *System* aufgebaut ist, können Sie mithilfe der speziell für DDE vorgesehenen Anweisungen *OpenDatabase* und *CloseDatabase* eine Datenbank öffnen bzw. schließen. Der Name der Datenbank muss hierbei direkt und ohne Anführungszeichen hinter dem Befehl angegeben werden. Die folgenden WinWord-Befehle bauen eine Verbindung zu Access auf und öffnen die Datenbank *C:\DB1.mdb*. Damit es funktioniert, muss Access bereits gestartet sein.

```
DDEKanal1 = DDEInitiate("MSAccess", "System")
DDEExecute DDEKanal1, "[OpenDatabase C:\DB1.mdb]"
```

Auf die gleiche Weise wie im vorhergehenden Abschnitt bei der Verbindungsaufnahmen mit Excel beschrieben, können Sie auch bei Access eine direkte Verbindung zu einer bereits geöffneten Datenbank aufbauen. Anstelle des Themas *System* geben Sie dazu einfach den Namen der Datenbank an, so wie er im Datenbankfenster von Access erscheint.

Wenn Sie auf eine Datenbank zugreifen möchten, die noch nicht geöffnet ist, stellen Sie zuerst eine Verbindung zum *System*-Thema her und öffnen die Datenbank mittels *OpenDatabase*-Befehl. Anschließend bauen Sie einen zweiten Kanal zu der nun geöffneten Datenbank auf. Das folgende Beispiel ist die Ereignisprozedur einer Schaltfläche namens *btnAccessDDE*, die sich auf einem WinWord-Benutzerformular befindet:

```
Private Sub btnAccessDDE_Click()

    DDEAccessKanal1 = DDEInitiate("MSAccess", "System")
    DDEExecute DDEAccessKanal1, "[OpenDatabase _
        C:\db1.mdb]"

    DDEAccessKanal2 = DDEInitiate("MSAccess", "DB1")
    DDEExecute DDEAccessKanal2, "[Testmakro]"
```

```
DDEExecute DDEAccessKanal2, "[OpenForm Formular1]"
```

```
DDETerminateAll
```

```
End Sub
```

Der DDE-Kanal zu der geöffneten Datenbank wird in diesem Beispiel dazu genutzt, ein Makro mit dem Namen *Testmakro* auszuführen. Eine weitere Möglichkeit besteht darin, über den Kanal eine beliebige Makroaktion direkt auszuführen. Dazu müssen Sie die Aktion auf die gleiche Weise wie innerhalb von VBA – allerdings ohne die *DoCmd*-Anweisung – angeben. In dem obigen Beispiel wird zusätzlich noch das Formular *Formular1* geöffnet.

Hinter dem Namen der Aktion können bei Bedarf die Aktionsargumente angegeben werden. Das folgende Beispiel öffnet das *Formular1* in der Entwurfsansicht:

```
DDEExecute DDEAccessKanal2, "[OpenForm Formular1, 1]"
```

Neben der Möglichkeit, über den DDE-Kanal zu einer Datenbank Makros und Aktionen auszuführen, stellt er folgende Elemente zur Verfügung:

Elemente einer Datenbank

Element	Beschreibung
<i>TableList</i>	Eine Liste aller Tabellen
<i>QueryList</i>	Eine Liste aller Abfragen
<i>FormList</i>	Eine Liste aller Formulare
<i>ReportList</i>	Eine Liste aller Berichte
<i>MacroList</i>	Eine Liste aller Makros
<i>ModuleList</i>	Eine Liste aller Module

Tabelle 13.9

Über die einzelnen Elemente können Informationen über die Objekte der Datenbank angefordert werden. Das Element *TableList* fordert beispielsweise eine Zeichenkette an, die die Namen aller Tabellen der Datenbank enthält. Die einzelnen Tabellennamen sind hierbei durch Tabulatorzeichen voneinander getrennt.

Das folgende Visual-Basic-Beispiel öffnet die *Nordwind*-Datenbank und gibt eine Liste aller Tabellen in einem Meldungsfenster aus. Vor der Ausführung der Prozedur sollten Sie sicherstellen, dass Access gestartet ist.



```

Private Sub Command1_Click()

Dim DDEKanal1 As Long
Dim DDEKanal2 As Long
Dim Tabellenliste As Variant

DDEKanal1 = DDEInitiate("MSACCESS", "System")
DDEExecute DDEKanal1, "[OpenDatabase _
    \"C:\Programme\Microsoft _
    Office\Office\Beispiele\NORDWIND.MDB\"]"

DDEKanal2 = DDEInitiate("MSACCESS", "Nordwind")

Tabelleliste = DDERequest(DDEKanal2, "TableList")

MsgBox Tabelleliste

DDETerminate DDEKanal2
DDETerminate DDEKanal1

End Sub

```

Zugriff auf Daten

Um den Zugriff auf Daten zu ermöglichen, stellt Access die Möglichkeit zur Verfügung, zusätzlich zum Datenbanknamen eine Tabelle, eine Abfrage oder einen SQL-Text als Thema anzugeben. Allgemein erfolgen die Angaben gemäß folgender Syntax:

```

<Datenbankname>; TABLE <Tabellenname>
<Datenbankname>; QUERY <Abfragenname>
<Datenbankname>; SQL <SQL-Text>

```

Der Zugriff auf die Daten erfolgt mithilfe der folgenden Elemente:

Elemente der Themen

Element	Beschreibung
<i>All</i>	Liefert alle Daten der Datenquelle (inklusive Feldnamen).
<i>Data</i>	Liefert alle Daten der Datenquellen, jedoch ohne Feldnamen.
<i>FieldNames</i>	Eine einspaltige Liste aller Feldnamen.
<i>FieldNames;T</i>	Eine zweispaltige Liste aller Feldnamen. Die zweite Spalte enthält einen Wert, der den Datentyp des Feldes angibt (siehe Tabelle).
<i>NextRow</i>	Liefert die Daten des nächsten Datensatzes. Beim ersten Öffnen der Datenquelle wird der Inhalt des ersten Datensatzes zurückgegeben.



Elemente der Themen (Forts.)

Element	Beschreibung
<i>PrevRow</i>	Liefert die Daten des vorhergehenden Datensatzes.
<i>FirstRow</i>	Liefert die Daten des ersten Datensatzes.
<i>LastRow</i>	Liefert die Daten des letzten Datensatzes.
<i>FieldCount</i>	Gibt die Anzahl der Datenfelder zurück.
<i>SQL Text</i>	Liefert den SQL-Text, der einer Datenquelle zugrunde liegt.
<i>SQL Text;<n></i>	Liefert den n-ten Teil eines SQL-Textes. <n> muss zwischen 1 und 255 liegen. Dieses Element wird für die Übergabe von SQL-Texten benötigt, die länger als 255 Zeichen sind.

Tabelle 13.10

Folgende Werte sind im Zusammenhang mit dem Element *FieldNames;T* stellvertretend für die VBA-Datentypen:

Datentypen des Elements *FieldNames;T*

Wert	Datentyp
1	Ja/Nein (Boolean)
2	Byte
3	Integer
4	Long
5	Currency
6	Single
7	Double
8	Datum/Zeit
9	Binäre Daten mit maximal 255 Bytes
10	Text
11	OLE-Objekt
12	Memo

Tabelle 13.11



Die einzelnen Daten werden grundsätzlich in Form einer Zeichenkette zurückgegeben, wobei die Feldinhalte durch Tabulatorzeichen und die Datensätze durch Absatzmarken voneinander getrennt werden.

Die folgende WinWord-Prozedur fragt die Artikelnamen und -preise in der Tabelle *Artikel* aus der *Nordwind*-Datenbank ab. Das Ergebnis wird in das aktuelle Dokument eingefügt:

```
Private Sub btnArtikeldaten_Click()  
  
    DDEAccessKanal1 = DDEInitiate("MSAccess", "System")  
  
    " Hier evtl. Pfad anpassen  
    DDEExecute DDEAccessKanal1, "[OpenDatabase _  
    "C:\Programme\Microsoft _  
    Office\Office\Beispiel\Nordwind.mdb"]"  
  
    DDEAccessKanal2 = DDEInitiate("MSAccess", _  
    "Nordwind;SQL SELECT  
    Artikelname, Einzelpreis FROM Artikel")  
  
    On Error Resume Next  
    Do While Err.Number <> 4599  
        ActiveDocument.Content.InsertAfter _  
            DDERequest(DDEAccessKanal2, "NextRow")  
        ActiveDocument.Content.InsertAfter vbCrLf  
    Loop  
  
    DDEExecute DDEAccessKanal1, "[CloseDatabase]"  
    DDETerminateAll  
  
End Sub
```



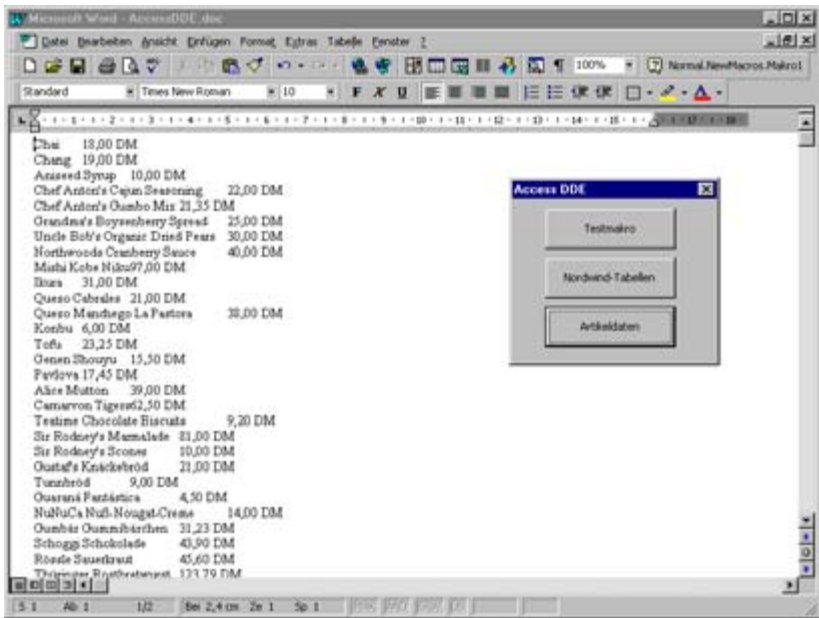


Abb. 13.24

Das Word-Dokument mit den Artikeldaten der Nordwind-Datenbank





Den Funktionsumfang von Access erweitern



Die Windows-Komponenten	690
Das Windows-API unter Access nutzen	694
Parameterübergabe an API-Funktionen	700
Das API unter Access nutzen	710





Microsoft Access eignet sich hervorragend zur Erstellung Windows-kompatibler Datenbankapplikationen. Access erlaubt dem Programmierer die Konzentration auf die zugrunde liegende Problemstellung, ohne sich um Windows bzw. die Programmierung von Fenstern, Eingabefeldern etc. kümmern zu müssen. All das, was ein C- oder PASCAL-Programmierer mühsam "von Hand" kodieren muss, steht dem Access-Entwickler "frei Haus" zur Verfügung. Schon die Erzeugung eines einfachen Fensters verlangt einem C-Programmierer die Tipparbeit von mindestens 50 Zeilen Code ab – verteilt auf mindestens zwei Dateien – während unter Access ein neues Formular mit nur wenigen Mausklicks erstellt wird.

Die Komplexität des *Windows-API*, des **A**pplication **P**rogramming **I**nterface, bleibt somit vor dem Access-Entwickler verborgen. Aber wie so oft, steckt der Teufel im Detail, und selbst die Leistungsfähigkeit von Access reicht manchmal nicht mehr aus. Probleme, die sich unter Access gar nicht oder nur mit ungerechtfertigt hohem Aufwand lösen lassen, können aber in vielen Fällen durch den Aufruf einiger Funktionen des Windows-API elegant, schnell und sicher gelöst werden. Durch Rückgriff auf das Windows-API tritt der ambitionierte Access-Entwickler in die Fußstapfen der Systementwickler und nutzt Systemfunktionen in eigenen Anwendungen.

Allerdings hat der Rückgriff auf das Windows-API seinen Preis: Zum einen lassen sich Systemfunktionen nur von VBA aus aufrufen, zum anderen müssen einige Vorkehrungen getroffen werden, um eine Systemfunktion anzusprechen: Der Vorgang, der eine Systemfunktion unter VBA nutzbar macht, nennt sich *Deklaration*. Ähnlich der Deklaration einer Variablen wird hier eine externe Systemfunktion in die Sprache VBA eingebettet. Wie Systemfunktionen deklariert werden, was dabei beachtet werden muss, ist Thema dieses Kapitels. Aus zahlreichen Beispielen können Sie erfahren, wie Systemfunktionen eingesetzt werden und – was noch viel wichtiger ist – wie Sie die Systemfunktion finden, die zur Lösung eines anstehenden Problems benötigt werden.

Die Windows-Komponenten

Hinweis



Sind Ihnen Begriffe wie User, Kernel, GDI und API schon bekannt, können Sie diesen Abschnitt überspringen und beim Abschnitt *Das Windows-API unter Access nutzen* fortfahren.

Dynamic-Link-
Library



Bevor Sie mit dem Einsatz von Systemfunktionen beginnen, müssen Sie sich leider ein wenig mit dem Betriebssystem Windows beschäftigen. Im Gegensatz zu Betriebssystemen älteren Baujahrs ist Windows kein monolithischer Block mehr, sondern besteht aus einer ganzen Reihe von Komponenten. Diese Komponenten werden **D**ynamic **L**ink **L**ibraries (*DLL*) genannt und enthalten – ähnlich einer öffentlichen Bibliothek – Funktionen, die sich akkreditierte Benutzer ausleihen können. Die gesamte Betriebssystemfunktionalität wird – bis auf einige Ausnahmen zur Ansteuerung der Hardware – in solchen DLLs bereitgestellt.



DLLs haben zwei Vorteile: Erstens werden nur die wirklich benötigten DLLs in den Speicher Ihres Rechner geladen, zweitens können fehlerhafte DLLs durch einfaches Kopieren der neuen, fehlerfreien DLL-Version ersetzt werden. Nutzt z. B. keine der laufenden Anwendungen die Multimediafähigkeiten von Windows, werden die entsprechenden DLLs – obwohl sie zum Betriebssystem gehören – nicht geladen und schonen so den wertvollen Speicherplatz.

Die Gesamtheit der mit Windows ausgelieferten DLLs stellt das so genannte *Windows-API* dar. Bevor visuelle Programmiersysteme wie Visual Basic oder Access auf den Markt kamen, konnten nur die Programmierer auf diese *DLLs* zugreifen, die sich im Besitz eines **Software Development Kits (SDK)** befanden. Dies bestand meist aus einem C-Compiler und einigen benötigten Deklarationsdateien, um sofort die Systemdienste nutzen zu können.



Klassische
Windows-Program-
miersysteme

Um heute anspruchsvolle Windows-Anwendungen zu entwickeln, müssen diese "rudimentären" Programmierwerkzeuge nicht mehr eingesetzt werden. Access erlaubt die Entwicklung komplexer Anwendungen, ohne den Rahmen von Access verlassen zu müssen. Sollte es dennoch der Fall sein, dass die von Access gebotenen Möglichkeiten nicht ausreichen, können Sie selektiv einzelne API-Funktionen nutzen – genau dort, wo es nötig ist. Sie müssen sich also nicht erst durch den Dschungel der über 1.000 API-Funktionen kämpfen, um Systemfunktionen unter Access zu nutzen.

Was ist das Windows-API?

Unter dem Begriff *Windows-API* (**A**pplication **P**rogramming **I**nterface) werden alle Funktionen zusammengefasst, die es Windows erlauben, sich selbst als grafische Benutzeroberfläche zu präsentieren. Darüber hinaus gehören zum API auch Funktionen für die Aufgaben hinter den Kulissen (z. B. Speichermanagement, Multitasking).

Wie oben schon erwähnt, stehen diese Funktionen in so genannten *DLLs* bereit. Aber auch *.exe*-Dateien können die in ihnen enthaltenen Funktionen anderen Anwendungen bereitstellen. (Der Ausdruck *Modul* statt DLL-oder *.exe*-Datei ist in diesem Zusammenhang gebräuchlicher und umfassender – wenn auch ein wenig ungenauer.)

Solange eine DLL nicht erst in den Speicher geladen werden muss, muss bei der Deklaration einer externen Funktion unter Access nicht der Name der DLL- bzw. *.exe*-Datei angegeben werden, sondern es kann der interne Modulname verwendet werden. Zu den DLLs, die sofort nach dem Start von Windows zur Verfügung stehen und in denen der größte Teil des Windows-API definiert ist, gehören folgende drei Hauptmodule:



Hauptmodule

- *Kernel32.dll* (*Kernel32*)
- *User32.dll* (*User32*)
- *GDI32.dll* (*GDI32*)





Diese Module enthalten Hunderte von Funktionen. Die Aufgabe für Sie als Programmierer besteht nun darin, die für Ihre Anwendung geeigneten Funktionen herauszufinden, um sie anschließend einzusetzen. Einige API-Funktionen sind sehr leicht zu erlernen und benötigen wenig Windows-Hintergrundwissen. Um andere Funktionen einsetzen zu können, werden jedoch Detailkenntnisse benötigt: z. B. in welcher Abfolge die Funktionen eingesetzt werden, auf welchen Datentypen die Funktionen operieren, wie diese Datentypen definiert werden etc. Um Ihnen bei der späteren Funktionssuche ein wenig behilflich zu sein, erfahren Sie in den folgenden drei Abschnitten einiges über die von diesen Modulen bereitgestellten Funktionen.

Der Windows-Kern (*Kernel32*)

Kernel32

In der *Kernel32*-Bibliothek sind alle Funktionen enthalten, die zum eigentlichen Kern eines Betriebssystems gehören. Der Begriff "Kern" stammt noch aus den Tagen, als ein Betriebssystem nicht "mehr" leistete, als neben einfachen Tastatureingaben und Bildschirmausgaben die Prozessorzeit zwischen gleichzeitig ablaufenden Anwendungen zu verteilen und die Festplatte mit einem Dateisystem zu versehen. Auch unter Windows werden diese Aufgaben von den Funktionen des *Kernel32* behandelt. Unter anderem werden hier aber auch Funktionen für die Ermittlung der aktuellen Windows-Version und für den Zugriff auf *.ini*-Dateien bereitgestellt. Von den Funktionen des *Kernel32*-Moduls kommen allerdings nur wenige für die Nutzung innerhalb einer Access-Anwendung in Betracht.

Die Benutzerschnittstelle (*User32*)

User32

Die Funktionen der Bibliothek *User32* beschäftigen sich mit der Benutzerschnittstelle – insbesondere mit den eigentlichen Fenstern. Hier werden Mittel und Wege zur Verfügung gestellt, im Fenster zu erzeugen, zu verschieben oder zu entfernen. Auch der Nachrichtenfluss, Menüs, der Mauszeiger und der Zeitgeber (*Timer*) werden von *User32*-Funktionen betreut.

Das Graphics Device Interface (*GDI32*)

GDI und
Gerätekontexte

Das **Graphics Device Interface** (*GDI32*) stellt Funktionen für grafische Ausgaben bereit. So können Sie mit *GDI32*-Funktionen Linien ziehen, Kreise zeichnen, Texte ausgeben und vieles mehr. Die Funktionen des *GDI32*-Moduls sind so vielfältig und umfangreich, dass eine Abbildung aller Funktionen auf Methoden eines Access-Formulars oder Reports den Rahmen sprengen würde. Daher wird in diesem Kapitel insbesondere auf die Grafikprogrammierung unter Access mithilfe der *GDI32*-Funktionen eingegangen.

Das GDI ist eine geräteunabhängige Programmierschnittstelle – egal, ob die Ausgabe auf einem Bildschirm oder einem Papierblatt erscheinen soll – programmiert werden Grafiken in beiden Fällen mit den gleichen Funktionen. Das GDI setzt dazu so genannte *Gerätekontexte* ein, die beim Aufruf eines GDI-Befehls übergeben werden und damit letztendlich das Ausgabegerät bestimmen.



Für die Druckausgabe werden GDI-Funktionen selten verwendet, da die Berichtsfunktionen von Access nicht mit grafischen Gestaltungsmöglichkeiten geizen – wem diese jedoch nicht ausreichen, der kann mit einigen Tricks und Kniffen den noch seine eigenen Grafiken in einem Report ausgeben.

Druckausgaben

Bezeichner und Handles

Eine Besonderheit, die Ihnen auf Systemebene immer wieder begegnen wird, sind so genannte *Bezeichner* oder *Handles*. Diese erlauben den Zugriff auf Objekte unterschiedlichster Art. So werden Fenster, Zeichenstifte, Mauszeiger, Bitmaps usw. durch Handles repräsentiert.

Handles wurden eingeführt, um auf die Probleme des Windows-Speichermanagements einzugehen. Wie Sie wissen, lagert Windows bei Bedarf Speicherbereiche auf die Festplatte aus. Das bedeutet, dass Speicheradressen u.U. nicht mehr gültig sind, weil der Speicherinhalt an diesen Adressen gar nicht mehr das enthält, was Sie erwarten. Durch Handles, die von Windows intern immer im Speicher gehalten werden, kann der Speichermanager feststellen, wo der tatsächliche Aufenthaltsort eines Objekts ist (RAM oder Festplatte) und kann so bei Bedarf den dem Handle zugeordneten Speicherbereich wieder ins RAM laden – unter Umständen an eine andere Adresse als beim ersten Zugriff.

Eine weitere Eigenschaft dieser Handles ist nicht weniger bemerkenswert: Es gibt niemals zwei Handles eines Objekttyps mit gleichem Wert. Durch diese Eindeutigkeit fällt es Ihnen und auch Windows leicht, einzelne Objekte systemweit zu unterscheiden. Während unter Windows 3.x noch eine 16-Bit-Integer-Variable zur Aufnahme eines Handles ausreichte, müssen unter Windows 95/98/Me (und Windows NT) schon 32 Bit breite *Long*-Variablen eingesetzt werden. Dies gilt für alle Handle-Typen.

Handles sind eindeutig

Die in diesem Kapitel vorgestellten Verfahren beziehen sich ausschließlich auf Windows und Access ab Version 95. Da Access nun eine vollständig in der 32-Bit-Technologie entwickelte Anwendung ist, wird nicht weiter auf die 16-Bit-Programmierung – wie etwa unter Access 2.0 und Win3.x – eingegangen.



Hinweis

Während unter Access jedes Formular den Zugriff auf das Handle des zugeordneten Formularfensters erlaubt (*hWnd*-Eigenschaft), können die Bezeichner der in einem Formular platzierten Steuerelemente nicht ermittelt werden. Erstens besitzen Access-Steuerelemente keine *hWnd*-Eigenschaft, und zweitens werden viele Access-Steuerelemente anders als z. B. Steuerelemente anderer Anwendungen und Programmiersysteme nicht über eigene Fenster realisiert, sondern von internen Access-Datenstrukturen kontrolliert. So lassen sich Endlosformulare leichter erzeugen, da für die Bereitstellung einer neuen Zeile nicht erst ein neuer Satz von Steuerelementen erzeugt werden muss.

Steuerelemente besitzen keine Handles

Dieser Umstand schränkt die Nutzung der API-Funktionen des *User*-Moduls auf Formulare – bzw. deren Fenster – und das Access-Hauptfenster ein.





Das Windows-API unter Access nutzen

Nach diesen generellen Bemerkungen zum Windows-API, zu DLLs und zu Handles nun zum Einsatz von Funktionen einer DLL. Folgende Schritte sind zu befolgen, wenn der Aufruf einer DLL-Funktion gelingen soll:

- Finden Sie eine nützliche API-Funktion.
- Lernen Sie, wie diese Funktion verwendet wird. Mit der entsprechenden Dokumentation ist dies nicht schwierig.
- Deklarieren Sie die API-Funktion. Einmal gelernt, vergisst man es nie mehr.
- Setzen Sie die API-Funktion wie eine gewöhnliche VBA-Funktion ein.
- Speichern Sie Ihre Module und Formulare. Eine Selbstverständlichkeit, die jeder ernsthafte Programmierer ohne großes Nachdenken erledigt.
- Testen Sie Ihre Anwendung. Auch ein sehr einfach zu befolgender, in der Praxis leider nicht oft befolgter Ratschlag. Besonders unter Zeitdruck wird dieser Schritt arg vernachlässigt. Planen Sie deshalb immer eine entsprechende Testphase in Ihre Terminkalkulation mit ein!

Tipps zur Erleichterung der Programmierung

Sicherheits-
vorkehrungen

- Der Einsatz der API-Funktionen geschieht unter Access ausschließlich durch VBA-Code. Hierzu können unter Access mehrere Codemodule eingesetzt werden, die die Gruppierung und Kapselung des Codes erleichtern. Es empfiehlt sich, jedem Modul die Zeile

`Option Explicit`

voranzustellen. So teilen Sie Access mit, dass alle verwendeten Variablen durch ein *Private*-, *Public*-, *Dim*-, *Global*-, *Const*- oder *Static*-Statement definiert werden müssen. Dies verhindert oft schon von vornherein die Übergabe von uninitialisierten Variablen an API-Funktionen, was zu Systemabstürzen führen kann. Auch in den "hinter" einem Formular verborgenen Codemodulen darf *Option Explicit* nicht fehlen. Dieser **Code Behind Forms (CBF)** ist VBA-Code, der direkt mit den Ereignisseigenschaften von Formularen oder Steuerelementen verknüpft werden kann und mit dem Formular gespeichert wird (siehe Ereignisprozeduren).

Kompilationszeit
verkürzen

- Darüber hinaus sollten alle API-Funktions- und Datentypdeklarationen in ein eigenes Modul eingebettet werden. So behalten Sie die Übersicht und können Access außerdem dadurch entlasten, dass Sie nur die Funktionen deklarieren, die tatsächlich eingesetzt werden. Bevor die erste VBA-Prozedur ausgeführt wird, kompiliert Access das entsprechende Modul. Sind VBA-Module überladen mit Prozeduren, dauert diese Kompilation unnötig lange.



Statt alle API-Deklarationen in ein eigenes Modul zu packen, besteht auch die Möglichkeit, private Deklarationen vorzunehmen. Wenn eine API-Funktion innerhalb eines Moduls mit dem Schlüsselwort *Private* gekennzeichnet ist, kann diese Funktion nur innerhalb des entsprechenden Moduls aufgerufen werden. Dieses Vorgehen hat den Vorteil, dass Module "self-contained" sind und somit ohne Probleme in eine andere Datenbank übernommen werden können – Namenskonflikte mit bestehenden VBA-Modulen werden so verhindert.

Private
Deklarationen

Nützliche API-Funktionen finden

Der erste Schritt für den erfolgreichen Einsatz einer API-Funktion ist gleichzeitig der schwerste: das Auffinden einer API-Funktion. Woher sollen Sie wissen, welche der über 1.000 API-Funktionen Ihnen bei einem aktuellen Problem weiterhilft? Leider gibt es auf diese Frage keine einfache Antwort – selbst C/Windows-Profis kennen nicht alle API-Funktionen. Eine Hilfequelle stellen die Hilfedateien des Windows-SDK dar. Diese befinden sich jedoch im Allgemeinen nicht im Besitz des Access-Anwenders. Microsoft vertreibt allerdings das so genannte *Access Developer Toolkit*, in dessen Umfang auch die SDK-Hilfedatei zu finden ist.

Wie für das Programmieren im Allgemeinen gilt auch für das Auffinden einer API-Funktion: Übung macht den Meister. Um die API-Programmierung zu üben, sollten Sie in Zeitschriften und Büchern so viel wie möglich über API-Tipps & Tricks lesen. Auch wenn der Artikel, den Sie in der Hand halten, nicht unmittelbar eine Lösung für Ihr Problem bietet, können Sie auf die richtige Fährte gebracht werden. Außerdem wird durch ständige Lektüre das Windows-API-Puzzle von Artikel zu Artikel vervollständigt. Eine wertvolle Quelle für API-Funktionen stellen insbesondere Visual-Basic- bzw. VBA-Artikel dar. Viele der für Visual Basic gültigen Techniken lassen sich auch unter Access nutzen.

Informationsquellen
nutzen

Verwendung der Funktion erlernen

Wenn Sie eine API-Funktion ausgemacht haben, die für Ihre Anwendung nützlich sein könnte, müssen Sie herausfinden, wie sie eingesetzt wird. Oftmals wird nicht nur eine einzige Funktion benötigt, sondern eine ganze Gruppe von Funktionen, Konstanten und Datentypen. Sie benötigen also eine Dokumentation der API-Funktionen wie sie beispielsweise in den Hilfedateien zu jedem Windows-C-Compiler enthalten ist. Im **Access Developer Toolkit (ADT)** finden Sie lediglich die API-Datenbank, in der alle Deklarationen von API-Funktionen und API-Konstanten enthalten sind. Diese Deklarationen können Sie über die Zwischenablage in Ihre Access-Datenbank übernehmen.

Das Access Distribution Toolkit (ADT)



Abb. 14.1

Ein Eintrag der
Windows-SDK-
Hilfedatei

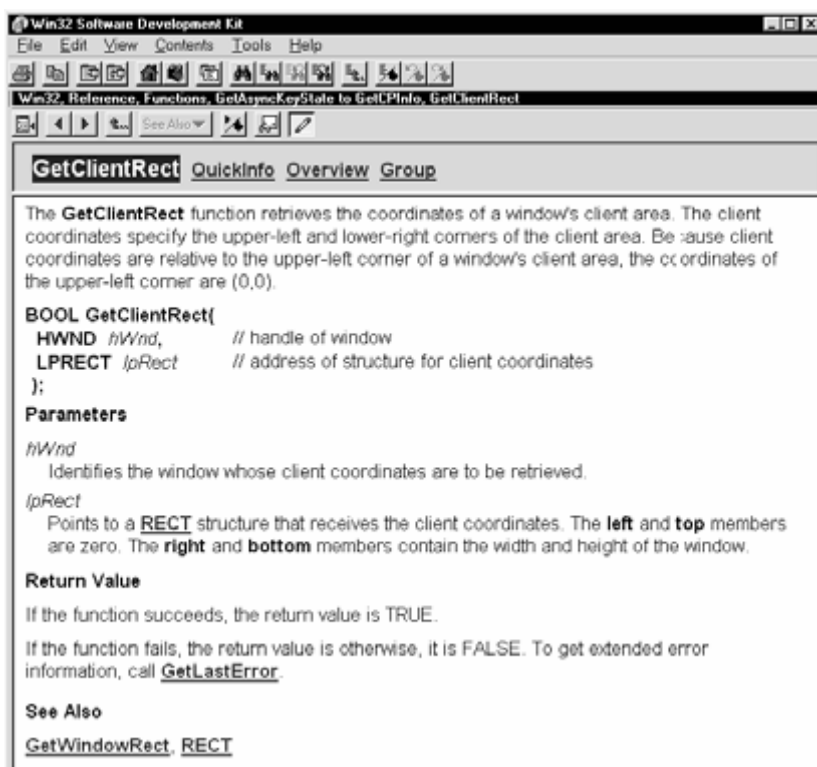
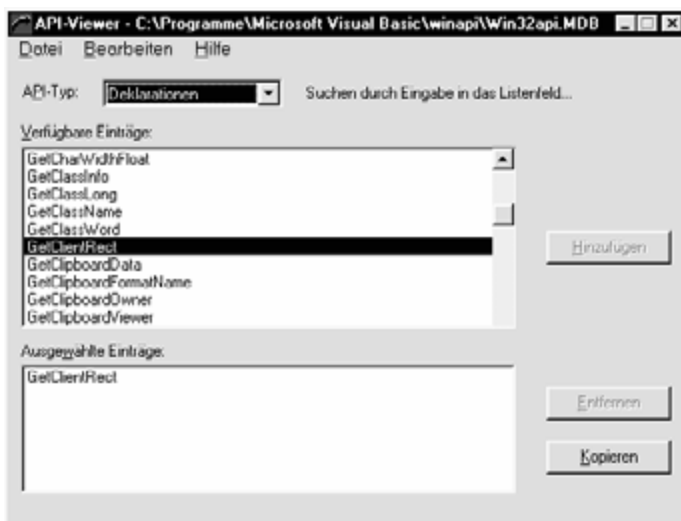


Abb. 14.2

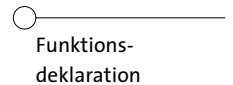
Der entsprechende
Eintrag in der
API-Datenbank



Während die API-Datenbank mit den Funktions-, Konstanten- und Datentypdeklarationen in erster Linie Tipparbeit erspart, dient die SDK-Hilfedatei zum Auffinden und Erlernen einzelner Funktionen und Funktionsgruppen. Hier können Sie im API-Revier herumstöbern. Sie finden z.B. Hilfeseiten, in denen API-Funktionen nach Gruppen geordnet aufgelistet werden. Ausgehend von einer Funktionsgruppe können Sie oft ein Problem so weit einkreisen, dass das Auffinden der letztendlich passenden API-Funktion eine Sache von wenigen Minuten ist.

API-Funktion deklarieren

Wenn die gefundene API-Funktion Ihren Ansprüchen entspricht bzw. für Ihre Aufgabenstellung geeignet ist, müssen Sie eine Verbindung zwischen Access und dem Windows-API herstellen. Dieser Vorgang wird *Deklaration* genannt. Mithilfe einer Funktionsdeklaration teilen Sie Access mit, in welcher DLL eine API-Funktion zu finden ist. Außerdem erfährt Access so, welche Parameter der Funktion übergeben werden müssen und von welchem Typ der Funktionsrückgabewert ist.



So einfach sich das in der Theorie anhört – in der Praxis kann es eine wahre Herausforderung sein, eine korrekte API-Deklaration anzugeben. Erstens muss das Problem der Datentypen geklärt werden. Access verfügt über die Datentypen *Byte*, *Integer* (%), *Long* (&), *String* (\$), *Single* (!), *Double* (#), *Date*, *Currency* (@) und *Variant*. Wenn Sie die SDK-Hilfedateien zu Rate ziehen, werden Sie die meisten dieser Datentypen jedoch vergeblich suchen. Die SDK-Hilfedateien wurden für C-Programmierer geschrieben, und die Sprache C kennt die oben aufgeführten Datentypen nicht – zumindest heißen sie in C anders als unter VBA. Sie müssen also erst die C-Parametertypen in ihre Access-Äquivalente überführen.

Dabei kommt noch erschwerend hinzu, dass zwischen *Referenz-* und *Werteparameter* (*ByVal*) unterschieden werden muss. Referenzparameter werden in der C-Terminologie als *Zeiger* bzw. *Pointer* bezeichnet – allerdings dürften Programmierer, die nur wenig von der Windows-Programmierung und den dort verwendeten Datentypen verstehen, arge Probleme damit haben, herauszufinden, ob sich hinter einem Parameter nun ein Zeiger oder ein Werteparameter verbirgt.

Steht in C dem Variablentyp das Präfix *LP* oder *P* vor, handelt es sich meist um einen Pointer (z.B. *LPRECT lpRect*).



Glücklicherweise können wir Ihnen jedoch die Datei *Win32API.txt* mitliefern, in der fast alle API-Funktionen der Bibliotheken *User32*, *Kernel32* und *GDI32* deklariert wurden. Diese Datei finden Sie eingebettet in ein eigenes Modul innerhalb der Datenbank *Kapitel4.mdb*.

Bei der Arbeit mit der Datei *Win32API.txt* sind uns einige Fehler aufgefallen, die wir natürlich sofort korrigiert haben. Dennoch kann keine Gewähr für die Richtigkeit aller Deklarationen in dieser Datei übernommen werden. In einigen Fällen wurde eine falsche DLL angegeben (z.B. *User32* statt *GDI32*), in anderen Fällen wurden einfach falsche Parameter deklariert. Auch fehlten einige der API-Konstanten. Selbst mit dieser Datei an der Hand werden Sie u.U. auf weitere Hilfe nicht verzichten können.





Beschränkung auf
das Wesentliche

In der Entwicklungsphase Ihrer Anwendung können Sie die Datei *Win32API.txt* in ein eigenes VBA-Modul laden (vorher muss sie aus der Datenbank *Kapitel14.mdb* in eine Textdatei exportiert werden). In der auslieferungsfähigen bzw. endgültigen Version Ihrer Datenbankanwendung sollten aber nur die Funktionsdeklarationen zu den Funktionen enthalten sein, die tatsächlich benötigt werden. Da Access beim Laden einer Datenbank alle Module auf syntaktische Korrektheit überprüft (kompiliert), wird die Geduld des Benutzers weniger strapaziert, wenn VBA-Module nur den Code bzw. nur die Deklarationen enthalten, die tatsächlich gebraucht werden. Die Überprüfung des gesamten Umfangs der Datei *Win32API.txt* nimmt je nach System einige Sekunden in Anspruch.

Bei der Beschränkung der Module auf das Wesentliche stehen Ihnen zwei Wege zur Verfügung: Entweder kopieren Sie die benötigten Funktionsdeklarationen direkt aus dem Modul mit der Datei *Win32API.txt* in ein eigenständiges Modul, um danach das *Win32API*-Modul aus der Datenbank zu entfernen, oder Sie greifen von Anfang an auf die API-Datenbank (enthalten im ADT) zurück, um von dort über die Zwischenablage nur die tatsächlich benötigten Funktionen in ein Access-Modul einzufügen.

API-Funktionen einsetzen

API-Funktionen wie
VBA-Funktionen
aufrufen

Unter der Voraussetzung, dass eine API-Funktion ordnungsgemäß deklariert wurde, kann diese in Ihrem VBA-Code eingesetzt werden. Prinzipiell können Sie eine API-Funktion auch innerhalb einer Ereignisseigenschaft eines Access-Objekts (Formular oder Steuerelement) aufrufen. Meist sind jedoch mehrere Schritte vor dem Aufruf der API-Funktion zu erledigen, so dass ein direkter API-Aufruf aus einer Ereignisseigenschaft eher selten ist.

In der Nutzung unterscheiden sich API-Funktionen nicht sehr von gewöhnlichen VBA-Funktionen oder Prozeduren. Das Einzige, worauf Sie achten müssen, ist die korrekte Übergabe der erforderlichen Parameter. Anders als Access verfügt das Windows-API nicht über Mechanismen zum Abfangen von Fehlern (z. B. *On Error...*). Wenn Sie die Programmkontrolle an eine API-Funktion abgegeben haben, arbeiten Sie ohne Netz und doppelten Boden. Selbst wenn die übergebenen Parameter vom Typ her korrekt sind, können fehlerhafte Parameterwerte mitunter zu einem Systemabsturz führen (z. B. wegen uninitialisierter Variablen).

Änderungen sichern

Speichern Sie Ihre
Module!

Bevor Sie die Wirkungsweise einer API-Funktion austesten, sollten Sie alle Formulare und Module erst einmal speichern und eine Sicherheitskopie der Datenbank anfertigen. Leider bietet Access keine Funktion zum automatischen Speichern eines Formulars beim Umschalten von der Entwurfs- in die Formularansicht. Gleiches gilt für Access-Module. Hier müssen Sie selbst Hand anlegen.

Das Speichern ist deshalb so wichtig, weil fehlerhafte Funktionsargumente Ihrer Anwendung Programmabstürze beschern können. Dies sind Fehler, die Windows nicht beheben kann und mit dem Schließen der Anwendung quittiert. Muss Access



infolge eines solchen Absturzes geschlossen werden, gehen alle Änderungen an nicht gespeicherten Formularen und Modulen verloren.

Anwendung testen

Der letzte Schritt für den Einsatz von API-Funktionen liegt auf der Hand: Erst durch sorgfältiges Austesten können Sie gewährleisten, dass ein ahnungsloser Anwender nicht mit Programmabstürzen oder anderen merkwürdigen Verhaltensweisen konfrontiert wird.

Unterschiede zwischen Windows-Versionen

Das Win32-API kommt prinzipiell in zwei Ausprägungen vor: einmal als ANSI-Variante und einmal als UNICODE-Variante. Während beide Varianten unter Windows NT vereint werden, verfügt Windows 95/98/Me – bis auf wenige Ausnahmen – nur über die Funktionen der ANSI-Variante.

Diese Dualität des Betriebssystems führt dazu, dass zur Unterscheidung der jeweiligen Variante der Buchstabe *A* (ANSI) oder *W* (WIDE Character, Unicode) an den Namen einer Funktion angehängt wird. So heißt die Funktion *SendMessage()* tatsächlich *SendMessageA()* oder *SendMessageW()* (nur Windows NT). Unter Verwendung des Deklarationsalias kann die Funktion *SendMessage()* jedoch ohne dieses störende Postfix eingesetzt werden:

```
Declare Function SendMessage Lib "user32" Alias  
"SendMessageA" (ByVal hwnd As Long, ByVal wParam As Long,  
Long, ByVal lParam As Long, lParam As Long) As Long
```

Bei der Portierung bestehenden Codes können Sie also durch Vergabe der richtigen Aliasnamen dafür sorgen, dass Access eine API-Funktion wiederfindet.

Eine Änderung, die weniger mit Windows in Zusammenhang steht, sondern mit VBA zu tun hat, erleichtert das Programmieren und erlaubt den "leichtfertigen" Umgang mit den Rückgabewerten einer API-Funktion: Obwohl eine API-Funktion als *Function* und nicht als *Sub* deklariert wurde, kann man die API-Funktion wie eine Prozedur aufrufen und muss sie nicht innerhalb eines Zuweisungsausdrucks verwenden. Dies verhindert die Deklaration von Dummy-Variablen zur Entgegennahme von API-Rückgabewerten, die sowieso nicht weiter benötigt werden.

Ein Beispiel

Um Ihnen einen Eindruck von der Leistungsfähigkeit der API-Funktionen zu geben, wird im folgenden Beispiel die API-Funktion aufgerufen, die ein Formularfenster zu einem Vordergrundfenster macht. Sie kennen diesen Fenstertyp z.B. von der Windows-Hilfe. Damit die Hilfeseiten auch während der Arbeit mit einer Applikation immer angezeigt werden, können Sie das Hilfefenster zu einem Vordergrundfenster machen (Menüpunkt *Optionen/Hilfe immer im Vordergrund*). Egal, welches





Fenster nun aktiviert wird, das Vordergrundfenster ist immer das oberste Fenster und somit immer sichtbar. Fenster anderer Applikationen (z.B. Explorer) können dieses Fenster nicht verdecken.

Die hier zum Einsatz gekommene Funktion heißt *SetWindowPos()* und ist im Modul *User32* angesiedelt. Diese Funktion erwartet als ersten Parameter den Bezeichner des Fensters (Handle), das in ein Vordergrundfenster umgewandelt werden soll. Die Funktion *SetWindowPos()* beeinflusst nicht nur den Überlappungsstatus (Z-Order) eines Fensters, sondern auch dessen Position (X- und Y-Koordinate) und Dimension (Breite, Höhe). Durch Angabe eines *Flags* – eines numerischen Wertes, der die Arbeitsweise einer Funktion steuert – legen Sie fest, was die Funktion *SetWindowPos()* tun soll. Um *SetWindowPos()* mitzuteilen, dass nur die Art der Überlappung verändert, Position und Dimension jedoch beibehalten werden sollen, wird die Funktion über das *BeimÖffnen*-Ereignis eines Formulars wie folgt aufgerufen:

```
Sub Form_Open(Cancel As Integer)
    SetWindowPos Me.Hwnd, HWND_TOPMOST, 0, 0, 0, 0,
    SWP_NOSIZE Or SWP_NOMOVE
End Sub
```

Normalerweise ist ein Formular ein Kindfenster des Access-Hauptfensters. Es besteht keine Möglichkeit für ein geöffnetes Formularfenster, aus diesem Rahmen "herauszuspringen". Das gilt jedoch nicht für Popup-Formulare. Diese werden nicht im Access-Hauptfenster abgelegt, sondern erscheinen direkt auf dem Desktop. Ein auf dem Desktop liegendes Fenster kann jedoch von anderen Fenstern auf dem Desktop wie z.B. dem Fenster des Explorers verdeckt werden. Durch den Aufruf von *SetWindowPos()* in der oben gezeigten Art und Weise wird das verhindert – allerdings nur dann, wenn das Formular tatsächlich ein Popup-Formular ist. Dazu müssen Sie die Formulareigenschaft *PopUp* auf *Ja* setzen und das Formular schließen, um dann beim erneuten Öffnen des Formulars in der Ereignisseigenschaft *BeimÖffnen* den Aufruf der Funktion *SetWindowPos()* durchführen zu lassen.

Parameterübergabe an API-Funktionen

Wenn Sie das obere Beispiel abgetippt haben, werden Sie feststellen, dass Access eine Fehlermeldung liefert – denn bis jetzt weiß Access noch gar nichts von der Funktion *SetWindowPos()*. Erst durch die Deklaration der Funktion kann Access eine API-Funktion in den richtigen Kontext einordnen. In der Deklarationszeile einer Funktion wird Access mitgeteilt, wo sich eine Funktion befindet und welche Parameter ihr übergeben werden:

```
Declare Function SetWindowPos Lib "user32" (ByVal hwnd
As Long, _
    ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal
y As Long, _
    ByVal cx As Long, ByVal cy As Long, ByVal wFlags As
Long) As Long
```



Die Syntax des *Declare*-Statements entnehmen Sie bitte der Access-Dokumentation oder den Access-Hilfedateien.



Hinweis

Wenn Sie wissen, dass eine DLL immer zur Verfügung steht (*Kernel32*, *User32* oder *GDI32*), können Sie bei der Funktionsdeklaration auf die Endung *.dll* bei der Angabe des Aufenthaltsortes der Funktion (*Lib*) verzichten. Muss die DLL hingegen erst geladen werden, muss auch die Endung *.dll* angegeben werden. Befindet sich die DLL-Datei nicht im aktuellen Verzeichnis, im *Windows*-Verzeichnis oder im *System*-Verzeichnis, benötigt das *Declare*-Statement zudem den gesamten Pfadnamen der DLL. Von dieser Möglichkeit sollten Sie allerdings nur in Notfällen Gebrauch machen, da ein anderer Rechner u.U. die hier angegebenen Verzeichnisse nicht aufweist und Ihre Datenbank somit nur auf Ihrem eigenen System ordnungsgemäß läuft.



Hinweis

Obwohl die API-Funktion tatsächlich als Funktion deklariert wurde, wird sie in unserem ersten Beispiel wie eine Prozedur aufgerufen. Dies ist besonders dann sehr praktisch, wenn man am Rückgabewert einer API-Funktion nicht interessiert ist. Man muss den Rückgabewert einer API-Funktion nicht unbedingt an eine Variable zuweisen. Hier kommt Access endlich der in C üblichen Programmierpraxis entgegen.



Funktionen und Prozeduren

Referenz- und Werteparameter

Schon bei der Deklaration einer Funktion wird angegeben, auf welche Art die Parameter übergeben werden sollen. Entweder wird der Inhalt bzw. Wert einer Variablen an die Funktion übergeben (*ByVal*), oder es wird die Adresse des Speicherplatzes übergeben, an dem der Wert der Variablen gespeichert wurde (*ByRef*). Diese Schlüsselwörter kennen Sie vielleicht schon aus der Programmierung mit VBA. Wenn der Parameter einer benutzerdefinierten Funktion oder Prozedur per Referenz übergeben wird, werden alle Änderungen am Parameter selbst durchgeführt. Liest der Aufrufer nach Beendigung der Funktion den Inhalt der übergebenen Parametervariablen aus, steht dort der in der Funktion zugewiesene Wert. So können Sie z. B. eine Parametervariable als Zählvariable innerhalb einer *For...Next*-Schleife verwenden:



Werteparameter werden mit *ByVal* gekennzeichnet

```
Sub Countdown( i As Integer ) "i ist ein
Referenzparameter
  for i = i to 0 Step - 1 "Zählt von i bis 0
  ...
  Next i
End Sub
```

Alle Änderungen an einem als Referenzparameter deklarierten Parameter sind für den Aufrufer sichtbar. Das heißt: Nachdem die Programmkontrolle von der Funktion an den Aufrufer zurückgegeben wurde, kann dieser durch Auslesen der übergebenen Variablen alle Änderungen sehen, die innerhalb der Funktion an der Variablen durchgeführt wurden.





```
Dim i
i = 100
CountDown i
" i hat jetzt den Wert 0 (geändert durch CountDown)
```

Auch API-Funktionen können sowohl Wert- als auch Referenzparameter erwarten. Referenzparameter werden häufig dazu eingesetzt, Informationen vom System abzufragen. Sie definieren z.B. eine Struktur und übergeben die Adresse dieser Struktur an eine API-Funktion, damit sie die Strukturelemente mit den angeforderten Informationen auffüllen kann.

String-Parameter

Tatsächlich gibt es eine Reihe von Funktionen innerhalb des Windows-API, die Referenzparameter als so genannte *Ausgabeparameter* verwenden. Besonders wenn es um Zeichenketten geht, wird auf Referenzparameter zurückgegriffen. So stellt der Aufrufer den Speicherplatz für eine Zeichenkette bereit, übergibt diese an eine API-Funktion, woraufhin die Funktion im übergebenen String z.B. den Pfad des Windows-Verzeichnisses ablegt. Als Rückgabewert einer Funktion werden Sie den Typ *String* niemals antreffen.

Dies hat seine Ursache darin, dass die Zeichenketten von Access anders verwaltet werden als die integralen Datentypen *Long*, *Integer*, *Single* und *Double*. Während Variablen dieser Typen in einer endlichen Anzahl von Bytes gespeichert werden können (max. 8 Bytes), können Strings ihre Länge während der Laufzeit ändern. Das bedeutet, dass sich auch die Adresse, an der ein String gespeichert wurde, während der Laufzeit ändern kann. Wenn ein String verlängert wird, der im Speicher z.B. zwischen zwei anderen Strings "eingekeilt" ist, findet er nun zwischen den beiden einkeilenden Strings nicht mehr genug Platz und muss in seiner Gesamtheit verschoben werden.

Access verwaltet daher für jeden VBA-String eine Art Handle, über das der dem String zugewiesene Speicherplatz jederzeit aufgefunden werden kann – auch wenn dieser durch die Bearbeitung des Strings verschoben wurde. Dieses Handle ist jedoch Access-typisch und wird nicht vom Windows-API unterstützt. Daher liefert keine API-Funktion einen Access-String als Rückgabewert.

String-Handles und
Stringadressen

Dass Access Strings über eine Art Handle-Mechanismus realisiert, hat allerdings noch eine weitere Konsequenz: Um die tatsächliche Speicheradresse ein Access-Strings an eine API-Funktion zu übergeben, muss der Parameter als *ByVal*-Parameter deklariert werden – obwohl sie eine Referenz auf den "Stringinhalt" übergeben wollen. Nur so übergibt Access die Adresse des ersten String-Zeichens an eine API-Funktion. Deklariert man einen Stringparameter als *ByRef*-Argument, übergibt Access das String-Handle an die API-Funktion. Diese kann damit (im Allgemeinen) nichts anfangen und provoziert oftmals einen Systemabsturz bzw. einen unfreiwilligen Applikationsabbruch.



Im folgenden Beispiel wird der Umgang mit Strings beim Aufruf einer API-Funktion deutlich. Die aufgelistete Codesequenz erzeugt einen neuen Schlüssel innerhalb der Registratur und legt einen String dort ab. Dazu wird auf die API-Funktionen *RegCreateKey()* und *RegSetValue()* zurückgegriffen:

```
Dim NewKey As Long
Dim ValueName as String
Dim Value As String
Dim KeyName As String

KeyName = "Software\Hersteller\Produkt\Version"
_ & Chr$(0)
ValueName = "Daten1" & Chr$(0)
Value = "DatenWert" & Chr$(0)

If RegCreateKey(HKEY_CURRENT_USER, KeyName, NewKey) = _
    ERROR_SUCCESS Then

    If RegSetValue(NewKey, ValueName, REG_SZ, ByVal
Value,
        _Len(Value)) = ERROR_SUCCESS Then
        \6Daten gespeichert
    End If

    RegCloseKey NewKey

End If
```

Über die Registratur haben Sie die Möglichkeit, sitzungsübergreifend Daten zu speichern. So kann z.B. die Position des Access-Hauptfensters beim Beenden einer Datenbankanwendung niedergeschrieben werden, um beim erneuten Start der Datenbankanwendung das Fenster an der Position anzuzeigen, an der der Benutzer es zuletzt geschlossen hat.

Was unter Windows 3.x in so genannten *.ini-Dateien* gespeichert wurde, wird unter Windows ab Version 95– bis auf wenige Ausnahmen – in der Registratur verwahrt.

Wie Sie aus dem obigen Codefragment ersehen können, wird an übergebene Strings das *NUL*-Zeichen (*Chr\$(0)*) angehängt. Für dieses Zeichen stellt VBA die Konstante *vbNullChar* zur Verfügung, die Sie anstelle von *Chr\$(0)* verwenden können. Dieses Zeichen wird von den API-Funktionen benötigt, um das Ende eines Strings zu erkennen. Hier wird deutlich, dass das gesamte Windows-API in C entwickelt wurde. Während Access über den eigenen Handle-Mechanismus neben der Adresse der Stringdaten auch die Länge des Strings abspeichert, muss die Länge eines C-Strings immer durch explizites Durchlaufen aller Zeichen – bis zum Auftauchen des *NUL*-Zeichens – ermittelt werden. Das wiederum bedeutet, dass ein Access-String das *NUL*-Zeichen durchaus enthalten kann, aber an diesem Zeichen noch längst nicht zu Ende ist:

Strings als Ausgabeparameter



Hinweis

C-Strings und Access-Strings





```
Dim s$  
s$ = "Dies ist ein API-String!" & vbNullChar & "Obwohl  
für Access diese Zeichen auch noch zum String gehören!"
```

In einem solchen Fall könnte es passieren, dass die API-Funktion nicht das gewünschte Ergebnis liefert – weil eben noch ein eingebettetes *NUL*-Zeichen im Access-String enthalten ist.

Allokation von Stringspeicher

Soll eine API-Funktion einen Namen, ein Verzeichnis oder sonstige Informationen in einem String ablegen, müssen Sie zunächst die dafür benötigten Zeichen allokalieren. Die einfache Deklaration einer Stringvariablen hat nämlich nicht die gleichzeitige Allokation von Speicherplätzen zur Folge:

```
Dim s As String  
DoSomeApiStuff s
```

Sollte die Funktion *DoSomeApiStuff* versuchen, einige Zeichen in dem mit *s* übergebenen String zu speichern, werden Sie sicher Ihr blaues Wunder erleben! Denn *s* stellt noch kein einziges Zeichen bereit, das von *DoSomeApiStuff* zur Ablage von Informationen genutzt werden könnte.

Sie müssen *s* erst einige Zeichen zuweisen, um den String gefahrlos von einer API-Funktion beschreiben zu lassen. Dies geschieht am besten durch Aufruf der *Space\$()*- oder *String\$()*-Funktionen von VBA:

```
Dim s As String  
s = Space( 255 ) \6255 Leer-Zeichen allokalieren  
DoSomeApiStuff s
```

oder

```
Dim s As String  
s = String$( 255, 0 ) \6255 NUL-Zeichen allokalieren  
DoSomeApiStuff s
```

Den so allokierten Stringspeicher kann eine API-Funktion beschreiben, ohne dass eine Schutzverletzung (kurz *UAE* oder *GPF*) stattfindet. Schreibt die API-Funktion aber über den zur Verfügung stehenden Speicherbereich hinaus, wird auch hier ein *UAE* ausgelöst. Allokieren Sie daher für einen String immer die von der API-Funktion maximal geforderte Anzahl von Zeichen.

Ein gängiges Maß für "die maximal geforderte Anzahl von Zeichen" ist der Wert 255. Darüber hinaus erwarten einige API-Funktionen die Übergabe der Anzahl der im übergebenen String (Buffer) enthaltenen Zeichen. In solch einem Fall brauchen Sie nicht die maximal mögliche Anzahl von Zeichen zu allokalieren (falls eine solche Größe überhaupt existiert), sondern müssen nur darauf achten, dass die übergebene Zeichenzahl nicht größer als der tatsächlich bereitgestellte Speicher ist (siehe den obigen Aufruf von *RegSetValue()*).



Die meisten Funktionen, die einen von Ihnen übergebenen String beschreiben, geben als Rückgabewert an, wie viele Zeichen ihres Buffers tatsächlich genutzt wurden (z.B. bei der Ermittlung von Verzeichnissen, oder der Titelzeile eines Fensters). Diesen Wert können Sie verwenden, um den Access-String so weit zu kürzen, dass dieser nur noch das tatsächliche Ergebnis enthält – ohne die ungenutzten Zeichen am Ende:

```
Dim Caption As String
Caption = String$(255, 0)
Caption = Left$(Caption, GetWindowText( hWnd, _
    Caption, Len(Caption)))
```

In diesem Beispielaufruf wird der Titelzeilentext innerhalb des Aufrufs der VBA-Funktion *Left\$()* durchgeführt. Dieser verschachtelte Aufruf ist deshalb erlaubt, weil zuerst die einer Funktion als Parameter übergebenen Funktionen ausgeführt (*GetWindowText()*) werden, bevor die eigentliche Funktion (*Left\$()*) aufgerufen wird.

Strings fester Länge

Neben den Strings mit variabler Länge kennt Access auch Strings mit einer fest vorgegebenen Zeichenzahl. Diesen Strings muss nicht durch *Space\$()* oder *String\$()* Speicherplatz zugewiesen werden. Vielmehr wird der Speicherplatz schon bei der Definition (bzw. Dimensionierung) der Variablen angelegt:

```
Dim FixedString As String * 255
```

So einfach die Allokation von Speicherplätzen mit dieser Art von Strings auch sein mag, einen großen Nachteil bringt sie mit sich: Strings fester Länge sind resistent gegen alle String-Manipulationsfunktionen. Im *GetWindowText()-*Beispiel wurde nach Ermittlung des Titelzeilentextes die Länge des ermittelten Eintrags auf die tatsächlich übergebene Zeichenzahl beschränkt. Wenn Sie dies mit Strings fester Länge versuchen, wird Ihnen leider kein Erfolg beschert:

```
Dim FixedString As String * 255
"API-Aufruf..."
FixedString = Left$(FixedString, nZeichen)
```

Da die Anzahl der einem festen String zur Verfügung stehenden Zeichen während der Laufzeit nicht mehr verändert werden kann, müssen Sie eine weitere (variable) Stringvariable anlegen, in die nur die gültigen Zeichen des festen Strings kopiert werden:

```
Dim FixedString As String * 255
Dim VariableString As String
"API-Aufruf..."
VariableString = Left$(FixedString, nZeichen)
```

Dieses Vorgehen ist jedoch wenig elegant und außerdem fehlerträchtig. Sie müssen für einen Zweck gleich zwei Variablen verwenden, das die Wartung Ihrer Software nur unnötig verkompliziert.



Odyssee eines
festen Strings bei
Übergabe an eine
API-Funktion

Neben der Verwendung von zwei Variablen für einen API-Aufruf haben feste Strings noch einen weiteren Nachteil: Access erzeugt bei der Übergabe des festen Strings an eine API-Funktion zuerst einen temporären String variabler Länge, kopiert die Daten des festen Strings in den temporären String, übergibt die Adresse des temporären Strings an die API-Funktion, um nach Beenden der API-Funktion den Inhalt des temporären Strings wieder in den festen String zu kopieren. Dies ist ein unnötiges Vorgehen, das einen Geschwindigkeitsnachteil gegenüber einer einfacheren Variablendeklaration nach sich zieht. Wenn Sie von vornherein mit Strings variabler Länge arbeiten, müssen Sie zwar zunächst den benötigten Speicher allokalieren, ersparen Access dadurch aber das doppelte Kopieren zwischen festem und temporärem String.

Typfreie Parameter

Wenn Sie die Funktion *RegSetValue* einer genauen Betrachtung unterziehen, werden Sie feststellen, dass in der Deklarationszeile der Ausdruck *lpData As Any* auftaucht.

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

Der hier verwendete Parametertyp *Any* wird nur im Zusammenhang mit der Deklaration eines API-Parameters verwendet. Er kommt immer dann zum Einsatz, wenn der Parameter einer API-Funktion aus unterschiedlichen Datentypen gebildet werden kann. Kann eine API-Funktion in einem Parameter unterschiedliche Datentypen entgegennehmen, wird meist auch ein *Flags*-Parameter übergeben, der der Funktion mitteilt, um welchen Datentyp es sich beim übergebenen Parameter handelt (z. B. *REG_SZ*).

Ein anderes Einsatzgebiet des Parametertyps *Any* liegt in der Übergabe von NULL-Zeigern. Wie schon erwähnt, sind Referenzparameter eng verwandt mit den in C bekannten Zeigern. Viele API-Funktionen ignorieren eine übergebene Variablenreferenz, wenn an ihrer Stelle der Wert *NULL* (0) übergeben wird. Um jegliche Missverständnisse auszuschließen, sollte ein NULL-Zeiger immer durch *ByVal 0* übergeben werden.

Any verhindert die
Syntaxprüfung

Durch Verwendung des Parametertyps *Any* wird die Syntaxprüfung für einen Parameter abgeschaltet. Klar, dass dieser Parametertyp mit Vorsicht zu genießen ist. Insbesondere die *ByVal/ByRef*-Deklaration ist zu beachten. Aber auch die Breite eines Parameters muss beim Aufruf berücksichtigt werden. Übergeben Sie anstelle einer geforderten Integervariablen (2 Byte) einen *Long*-Wert (4 Byte), wird die API-Funktion nicht ordnungsgemäß funktionieren. Bei der Übergabe von Parametern mit unkorrekten Bytelängen meldet Access den Fehler "Falsche DLL Aufrufkonventi-



on". Dies ist ein Zeichen dafür, dass der Stack nach dem Aufruf der Funktion nicht ordnungsgemäß aufgeräumt wurde – weil z. B. ein *Long*- statt eines *Integer*-Wertes übergeben wurde.

Wie kommt eine API-Funktion aber dazu, sowohl eine *String*- als auch eine *Long*-Variable als gültigen Parameter zuzulassen? Im Gegensatz zu VBA wird beim Aufruf einer API-Funktion zur Laufzeit keine Überprüfung des Typs einer übergebenen Variablen durchgeführt (dies übernimmt im Allgemeinen der C-Compiler). Der Aufrufer ist dafür verantwortlich, dass die übergebenen Parameter sinnvolle Werte und Referenzen enthalten. Dieser kleine, aber feine Unterschied erklärt, warum viele API-Funktionen bei der Übergabe einer Adresse bzw. Referenz auf einen String oder eine Datenstruktur auch den Wert *o&* akzeptieren. Das erklärt auch, warum eine API-Funktion bei Übergabe der Adresse *o&* einen leeren String erkennt und nicht auf einen String mit dem *NUL*-Zeichen als einzigem Zeichen rechnet. Erstens werden durch diese NULL-Zeiger-Konvention sowohl Strings als auch Datenstrukturen abgedeckt, zweitens dauert die Überprüfung eines übergebenen Strings länger (Durchlaufen der Zeichen und Suche nach dem *NUL*-Zeichen) als der sofortige Test des übergebenen Zeigers (wurde eine Speicheradresse *<> o* übergeben?).

Inkonsistenzen des APIs

Doch zurück zu Referenz- und Werteparametern im Zusammenhang mit *Any*. Im Fall von *RegSetValue()* wurde der Parameter *lpData* als Referenzparameter deklariert (*lpData as Any* und nicht *ByVal lpData As Any*). Wenn Sie beim Aufruf eine beliebige Variable übergeben, erzeugt Access eine Referenz auf diese Variable und übergibt somit die Adresse der Variablen an die API-Funktion. Das bedeutet, dass Sie bei der Übergabe eines NULL-Zeigers einen so genannten *Type-Override* durchführen müssen. In der Praxis heißt das nichts anderes, als dass Sie beim Aufruf der API-Funktion das Schlüsselwort *ByVal* dem Wert *o&* voranstellen, um trotz geforderter Referenz den angegebenen Wert zu übergeben.

Explizite Wertübergabe

Übergeben Sie einer API-Funktion in einem *ByRef*-Parameter eine Konstante, legt Access zunächst eine temporäre Variable an, speichert den Wert der zu übergebenen Konstanten in dieser temporären Variablen ab und übergibt der API-Funktion eine Referenz auf diese Variable.

Referenzen auf Konstanten vermeiden!

Typsichere API-Funktionen

Die Verwendung des Parametertyps *Any* birgt aus den oben geschilderten Gründen einige Gefahren: Sie können einer API-Funktion beliebige Variablen als Parameter übergeben, ohne dass Access eine Typprüfung vornimmt und Sie so schon bei der Syntaxprüfung – also noch vor der Ausführung Ihrer Anwendung – über falsche Parameter informiert. Schlimmstenfalls wird Ihr System beim Aufruf der API-Funktion einen Totalabsturz erleiden. Es kann auch passieren, dass Sie nur Ihre aktuelle Access-Sitzung schließen müssen. In beiden Fällen bekommen Sie aber zumindest einen Hinweis darauf, dass etwas nicht in Ordnung ist.

Es kann aber auch vorkommen, dass eine API-Funktion nicht mit spektakulären Aktionen aufwartet, wenn ihr unkorrekte Parameter übergeben wurden. Vielmehr liefert die API-Funktion z. B. Rückgabewerte, die Sie nicht erwartet haben. Um dies





von vornherein auszuschließen und die Syntaxprüfung von Access als nützliches Debugging-Werkzeug einzusetzen, können Sie durch Verwendung von Aliasen eine API-Funktion in allen benötigten Ausprägungen deklarieren.

Folgendes Beispiel zeigt die Deklaration der schon bekannten Funktion *RegSetValueEx* zum Setzen eines Strings bzw. einer *Long*-Variablen in der Registratur:

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

Schade ist nur, dass bei der Deklaration der unterschiedlichen Ausprägungen einer DLL-Funktion keine Konstanten als Parameter vorgegeben werden können. So hat der Parameter *cbData* beim Setzen eines Langworts in der Registratur immer den Wert 4. Es wäre schön, wenn dies schon bei der Funktionsdeklaration angegeben werden könnte – leider ist solch ein Makromechanismus unter Access nicht vorgesehen, könnte jedoch durch eine dezidierte VBA-Prozedur nachgebildet werden.

Ein Trick hilft unter bestimmten Voraussetzungen, diesen Mechanismus zu realisieren. Ist der Parameter wie in diesem Beispiel der letzte der Aufzählung, können Sie mit dem Schlüsselwort *Optional* diesen Parameter als optional deklarieren. Hierbei haben Sie die Möglichkeit, den Wert des Parameters vorzugeben, falls er nicht beim Aufruf angegeben wird. Die obige Deklaration würde daher folgendermaßen lauten:

```
Declare Function RegSetValueExLong Lib "advapi32.dll"
Alias _
"RegSetValueExA" (ByVal hKey As Long, _
ByVal lpValueName As String, ByVal Reserved As Long, _
ByVal dwType As Long, lpData As Long, _
Optional ByVal cbData As Long = 4) As Long
```

Beim Aufruf der Funktion brauchen Sie dann den letzten Parameter nicht mehr anzugeben.

Wenn der Name der deklarierten Funktion mit dem in der DLL angegebenen Namen übereinstimmt, kann auf die Angabe des Schlüsselworts *Alias* verzichtet werden. Wenn der unter Access "sichtbare" Name einer API-Funktion jedoch von dem in der DLL vermerkten Namen abweichen soll, muss hinter dem Schlüsselwort *Alias* der in der DLL vermerkte Name angegeben werden.



Benutzerdefinierte Datentypen (Strukturen) und Arrays

Neben numerischen und *String*-Variablen verlangen einige API-Funktionen auch den Einsatz von benutzerdefinierten Datentypen. Diese werden verwendet, wenn die benötigten Informationen nicht mehr in einem einzelnen Datentyp (*Integer* oder *Long*) übergeben bzw. ausgelesen werden können. Am Beispiel des Typs *Rect* sehen Sie die Definition solcher benutzerdefinierten Datentypen und eine der API-Funktionen, die diesen Datentyp verwendet:

```
Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type

Declare Function GetWindowRect Lib "user32" (ByVal
hwnd As Long, lpRect As RECT) As Long
```

Der Datentyp *Rect* wird vom Windows-API immer dann eingesetzt, wenn es darum geht, die Dimension eines Objekts (z. B. eines Fensters) zu beschreiben. Die im *Rect*-Typ enthaltenen Variablen geben die Koordinaten der linken oberen und rechten unteren Ecke eines Objekts an. Soll solch ein benutzerdefinierter Datentyp (in C wird dies *Struktur* genannt) an eine Windows-API-Funktion übergeben werden, müssen Sie in der Deklarationszeile als Parameter eine Referenz auf den Benutzertyp angeben.

Den Einsatz der Funktion *GetWindowRect* lernen Sie in einem späteren Abschnitt kennen.

Beim Aufruf einer so deklarierten Funktion wird ihr die Adresse der Struktur übergeben. Die Elemente eines Benutzertyps werden im Speicher an aufeinander folgenden Speicheradressen abgelegt. So begnügt sich eine API-Funktion mit der Angabe der Adresse des ersten Elements, um auf alle weiteren Elemente des Benutzertyps zuzugreifen. Dazu müssen die Definitionen dieser Struktur sowohl unter VBA als auch im Windows-API übereinstimmen. Hier hilft die Datei *Win32API.txt* weiter, in der nicht nur alle Funktionsdeklarationen enthalten sind, sondern auch alle API-Strukturen und Konstanten.

Bei der Entwicklung eigener DLL-Funktionen in C, die von VBA aus aufgerufen werden und Strukturen als Parameter erwarten, müssen Sie das so genannte *Byte Alignment* Ihres Compilers auf 1 *Byte* stellen. So ist gewährleistet, dass der C-Compiler die Elemente einer in VBA definierten Struktur an den erforderlichen Offset findet.

Oftmals wird verlangt, dass das erste Byte, Wort oder Langwort eines Benutzertyps mit der Länge der Struktur belegt wird. Anhand dieser Längeninformation kann die API-Funktion dann entscheiden, wie groß die übergebene Struktur ist. Diese Länge kann mithilfe der *Len*-Funktion – genau wie bei Strings – ermittelt werden:



Hinweis



Die Adresse des Benutzertyps reicht aus, um diesen vollständig zu definieren



Hinweis





Arrays an API-Funktionen übergeben

```
Dim s As Struktur  
s.cbLen = Len(s)
```

Die Übergabe eines Arrays an eine API-Funktion geschieht analog zu der Übergabe einer benutzerdefinierten Variable. In der Funktionsdeklaration geben Sie an, dass als Parameter die Referenz auf ein Array eines bestimmten Variablen- oder Strukturtyps erwartet wird. Beim Aufruf der Funktion übergeben Sie dann das erste Element des Arrays.

Wollen Sie statt der *Rect*-Struktur ein 4-elementiges Array des Typs *Long* einsetzen, können Sie sich folgender Deklaration bedienen:

```
Declare Sub GetWindowRectLong Lib "User32" Alias  
"GetWindowRect" (ByVal hWnd As Long, Array() As Long)  
...  
Dim DimPos(4) as Long  
GetWindowRect Forms("Mein Formular"), DimPos(0)
```

Zusammenfassung

Wie Sie sehen gibt es mehrere Wege, eine API-Funktion zu deklarieren. Um Fehler von vornherein auszuschließen, sollten Sie sich die Mühe machen und eine API-Funktion mithilfe von Funktionsaliasen in allen benötigten Ausprägungen deklarieren. Verzichten Sie weitgehend auf *Any*-Parameter, um sich die Typprüfung von Access zunutze zu machen. Achten Sie bei der Übergabe von Strings darauf, dass der entsprechende Parameter als *ByVal*-Parameter deklariert wurde, um die Adresse des ersten Zeichens und nicht das String-Handle zu übergeben. Hängen Sie an Zeichenketten, die dem API übergeben werden, immer das String-Ende Zeichen (*vbNullChar*) an, falls einer API-Funktion nicht die Länge des übergebenen Strings in einem weiteren Parameter übergeben werden kann. Vermeiden Sie die Verwendung fester Strings und die Übergabe von Referenzen auf Konstanten (temporäre Variablen).

Das API unter Access nutzen

Lasst Taten sprechen! Unter diesem Motto steht nun der Rest dieses Kapitels. Sie finden hier Beispiele für die effektive Nutzung des Windows-API unter Access. Aber keine Angst, Sie erhalten keine stupide Liste mit Funktionsaufrufen und Codefragmenten. Vielmehr wird jedes Beispiel eingehend erläutert, um Sie so ein wenig in die Windows-Programmierung einzuführen. Dieses Wissen lässt sich als Ausgangsbasis für eigene API-Exkursionen nutzen. Nehmen Sie sich die Zeit, die folgenden Seiten durcharbeiten.

Neben einigen API-Funktionen, die die Manipulation der Formularfenster und des Access-Hauptfensters ermöglichen, werden Sie anhand einer Art Projekt erfahren, wie Sie die grafischen Fähigkeiten von Access durch Einsatz von API-Funktionen erweitern.

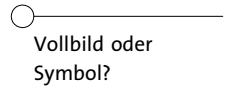


Fenstermanipulationen

Obwohl ein Access-Formular zahlreiche Eigenschaften bietet, lassen sich nicht alle möglichen Zustände eines Formulars über diese Eigenschaften ermitteln und einstellen. Entweder wurden spezielle Eigenschaften einfach vergessen, oder die Access-Entwickler waren sich von vornherein im Klaren darüber, dass weniger oft benötigte Funktionen bei Bedarf durch API-Funktionen nachgebildet werden können.

Vollbild oder Symbol?

So können Sie ein Formular zwar zu einem Symbol verkleinern oder als Vollbildfenster bildschirmfüllend (bzw. das Access-Hauptfenster ausfüllend) anzeigen – ermitteln, in welchem Zustand sich ein Formular befindet, können Sie über eine entsprechende Formulareigenschaft jedoch nicht. Abhilfe schaffen hier zwei API-Funktionen:



```
Declare Function IsIconic Lib "user32" Alias  
"IsIconic" _  
    (ByVal hwnd As Long) As Long
```

```
Declare Function IsZoomed Lib "user32" Alias  
"IsZoomed" _  
    (ByVal hwnd As Long) As Long
```

Mit *IsZoomed()* können Sie feststellen, ob ein Fenster im Vollbildmodus angezeigt wird. In diesem Fall liefert diese Funktion einen Wert ungleich 0. *IsIconic()* hingegen gibt Aufschluss darüber, ob ein Fenster zum Symbol verkleinert wurde. Auch hier ist dann der Rückgabewert ungleich 0. Folgende VBA-Funktion greift auf diese Funktionen zurück, um den aktuellen Formularzustand eindeutig zu bestimmen:

```
Global Const NORMAL = 0  
Global Const MAXIMIZED = 1  
Global Const MINIMIZED = 2  
...  
Function WindowState( ByVal hWnd& )  
    If IsIconic( hWnd ) then WindowState = MINIMIZED:  
Exit Function  
    If IsZoomed( hWnd ) then WindowState = MAXIMIZED:  
Exit Function  
    WindowState = NORMAL  
End Function
```

Sie übergeben der VBA-Funktion *WindowState()* das Handle eines Formularfensters und erhalten im Gegenzug den Zustand des Fensters zurück:

```
Zst = WindowState( Forms("Mein Formular").hWnd )
```





Das Handle des Hauptfensters

Absicht oder Ignoranz?

Allerdings hat diese Funktion noch einen Haken: Da ein Formularfenster innerhalb des Hauptfensters angezeigt wird, wissen Sie noch nichts darüber, ob das Formularfenster tatsächlich den gesamten Bildschirm einnimmt. Sie müssen zudem wissen, wie das Hauptfenster dargestellt wird. Dazu benötigen Sie jedoch das Fenster-Handle des Hauptfensters und können über dieses Handle den aktuellen Zustand des Hauptfensters ermitteln.

Access XP bietet eine Eigenschaft, über die das Hauptfenster-Handle ausgelesen werden kann:

```
Application.hwndAccessApp
```

Positionieren eines Fensters

Während ein Formularfenster über Eigenschaften zur Positionierung und Dimensionierung verfügt (Links, Rechts, Höhe, Breite), suchen Sie ähnliche Hauptfenster-Eigenschaften vergeblich. Hier müssen Sie wieder auf API-Funktionen zurückgreifen, um die Position und Dimension des Hauptfensters z. B. auf die Werte zu setzen, die sie hatten, als der Benutzer eine Datenbankanwendung geschlossen hat. Beim erneuten Laden der Datenbank werden diese gespeicherten Werte aus der Registrierung ausgelesen und dem Hauptfenster über die API-Funktion *SetWindowPos()* mitgeteilt:

```
Declare Function SetWindowPos Lib "user32" _
    (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, _
    ByVal x As Long, ByVal y As Long, ByVal cx As Long, _
    ByVal cy As Long, ByVal wFlags As Long) As Long
...
Dim r As RECT
... \6RECT-Werte einlesen
SetWindowPos Application.hwndAccessApp, 0, r.Left,
r.Top, r.Right-r.Left, t.Bottom - r.Top, SWP_NOZORDER
```

Überlappungszustand beibehalten

Dieser Aufruf positioniert Ihr Hauptfenster auf die in der *RECT*-Struktur angegebenen Daten. Wie Sie aber schon zu Beginn dieses Kapitels erfahren haben, kann die Funktionalität von *SetWindowPos()* durch Angabe von Flags dazu benutzt werden, den Überlappungszustand eines Fensters zu ändern. Da genau dies bei der Repositionierung des Hauptfensters nicht geschehen soll, wird als einziges Flag die Konstante *SWP_NOZORDER* angegeben, die dafür sorgt, dass sich am Hauptfenster nur die Position und Dimension ändert – Überlappungszustand, Aktivierungsstatus etc. bleiben erhalten.

Pixel versus cm

Beachten Sie, dass die Position und Dimension beim Aufruf von *SetWindowPos()* in der Einheit Pixel angegeben werden muss. Im Allgemeinen wird für Access-Objekte die Maßeinheit cm verwendet. Wenn Sie jedoch ein Formularfenster pixelgenau positionieren möchten, können Sie *SetWindowPos()* auch auf ein Formularfenster anwenden – ohne auf die Formulareigenschaften *Links*, *Rechts*, *Höhe* und *Breite* zurückzugreifen. Eine Funktion zur Umwandlung verschiedener Maßeinheiten finden Sie weiter unten in diesem Kapitel.



Formulare ohne Titel

Manchmal soll ein Formular angezeigt werden, das keine Titelzeile besitzt. Dazu können Sie zwar die Formulareigenschaft *Rahmenart* auf *Keine* setzen, müssen aber auch die Eigenschaften *Mit Systemmenüfeld*, *Mit Min Schaltfläche* und *Mit Max Schaltfläche* auf *Nein* setzen. Das daraus resultierende Fenster besitzt nun keine Titelzeile mehr – allerdings fehlt auch der Rahmen. So fällt vor allen Dingen die Abgrenzung zu anderen Fenstern sehr schwer.

Fenster mit einem Rahmen, aber ohne Titelzeile, erhalten Sie mit einigen API-Aufrufen. Sie müssen dazu erstens einige Daten der internen Fensterstruktur ändern und zweitens das Fenster nach dieser Manipulation um die Höhe der entfernten Titelzeile verkleinern.

Um beide Aufgaben zu erledigen, müssen Sie allerdings schon ein wenig tiefer in die Windows-Interns einsteigen. Vorab schon einmal die Funktion zur Entfernung der Titelzeile:

```
"RemoveCaptionBar -- Entfernen der Titelzeile eines
"                      Fensters und Darstellung
"                      desselben im Vordergrund
"F                      - zu bearbeitendes Formular
Sub RemoveCaptionBar(F As Form)
    Dim OldStyle          As Long
    Dim NewStyle          As Long
    Dim W                 As Long
    Dim H                 As Long
    Dim TwipsPerPixelX   As Long
    Dim TwipsPerPixelY   As Long
    Dim hdc               As Long
    "TwipsPerPixel auf Desktop-DC ermitteln
    hdc = GetDC(GetDesktopWindow())
    TwipsPerPixelX = 1440 / GetDeviceCaps(hdc, _
        LOGPIXELSX)
    TwipsPerPixelY = 1440 / GetDeviceCaps(hdc, _
        LOGPIXELSY)
    ReleaseDC GetDesktopWindow(), hdc

    "Aktuellen Fensterstil ermitteln
    OldStyle = GetWindowLong&(F.hWnd, GWL_STYLE)

    "Stile für Titel entfernen
    NewStyle = (OldStyle And Not WS_CAPTION)
    NewStyle = NewStyle Or WS_BORDER

    "Neuen Stil setzen
    SetWindowLong F.hWnd, GWL_STYLE%, NewStyle&

    "Breite des Formulars ermitteln
    W = F.Width \ TwipsPerPixelX
```





```

"Höhe des Detailbereichs ermitteln
H = F.Section(acDetail).Height \ TwipsPerPixelY

" Fenster an neue Stile anpassen, aber nicht
" neu zeichnen
SetWindowPos F.hWnd, 0, 0, 0, W, H, SWP_NOMOVE Or
SWP_NOZORDER Or SWP_FRAMECHANGED
End Sub

```

In dieser Funktion kommen einige API-Aufrufe zum Einsatz: Zunächst einmal werden mit *GetWindowLong()* und *SetWindowLong()* die internen Datenstrukturen eines Fensters bearbeitet. Sie erlauben das Lesen und Schreiben eines einzelnen *Long*-Wertes innerhalb dieser Struktur. Diese Fensterstruktur ist für den Programmierer nur über die Funktionen *GetWindowLong()* und *SetWindowLong()* erreichbar. Einen anderen Zugriff auf die Fensterstrukturen erlaubt Windows nicht.

In der oben abgedruckten Funktion *RemoveCaption()* wird mit *GetWindowLong()* zunächst das Langwort ermittelt, das die Fensterstile bestimmt (*GWL_STYLE*). Dieses wird so geändert, dass die Titelzeile verschwindet (*And Not WS_CAPTION*) und ein dünner Rahmen angezeigt wird (*Or WS_BORDER*). Nach dem Zurückschreiben des neuen Fensterstils mit *SetWindowLong()* muss jedoch dafür gesorgt werden, dass das Fenster neu gezeichnet wird. Dies geschieht durch den Aufruf von *SetWindowPos()*. Vorher muss das Fenster allerdings noch um die Höhe der Titelzeile verkleinert werden.

Dazu wird zunächst die Höhe des Detailbereichs ermittelt ($H = F.Section(acDetail).Height$). Diese Höhe wird über die DC-Umrechnungsfunktionen von der Einheit Twips in die Einheit Pixel umgerechnet, um sie dann via *SetWindowPos()* an das Fenster zu übergeben (siehe Abschnitt *Maßeinheiten*). Da Access eine Fehlermeldung erzeugt, wenn die Kopf- oder Fußzeile eines Formulars nicht eingeblendet wurde, aber mittels *F.Section(acHeader)* oder *F.Section(acFooter)* auf diese Bereiche zugegriffen wird, müssen Sie die Funktion *RemoveCaption* von Hand anpassen, wenn auch Kopf- oder Fußzeile mit in die Höhenberechnung eingehen sollen.

Mit *SetWindowPos()* wird einem Fenster eine neue Position und Dimension zugewiesen. Durch Angabe des Flags *SWP_FRAMECHANGED* veranlassen Sie Windows zur Neuberechnung des Fensterrahmens – basierend auf den vorher geänderten Fensterstilen.

Die hier vorgestellte Prozedur *RemoveCaption()* sollte in der Formular-Ereigniseigenschaft *Beim Öffnen* aufgerufen werden, um schon vor dem ersten Anzeigen eines Fensters die Titelzeile zu entfernen. Dieser Aufruf erfolgt entweder durch Angabe von $= RemoveCaption(Forms![Name des Fensters])$ im Eigenschaftsfenster oder innerhalb der *Open*-Ereignisprozedur von VBA:

```

Sub Form_Open( Cancel As Integer )
    RemoveCaption Me
End Sub

```





Mit dieser Funktion lässt sich die Titelzeile sowohl von Popup-Formularen als auch von gewöhnlichen Formularen entfernen. Doch Vorsicht: Ein Formular ohne Titelzeile kann nicht mehr mit der Maus bewegt werden. Da das Fenster außerdem über einen dünnen Rahmen verfügt (*WS_BORDER*), kann es zudem nicht mehr in seiner Größe verändert werden.

Obwohl solche Fenster vornehmlich für so genannte *ToolTips* oder *QuickInfos* eingesetzt wurden, die mittlerweile zum Standardrepertoire von Access bzw. den Access-Steuerelementen gehören, können diese Fenster dennoch sehr nützlich zur Anzeige von aktuellen Zustandsinformationen sein – z. B. zur Anzeige der aktuellen Mausposition oder – wie etwa in *Schedule+* – zur Anzeige von Zusatzinformationen bei der Auswahl eines Termins.

In der Datenbank *Kapitel14.MDB* finden Sie ein Formular (*Hinweis*), das später genutzt wird, um eben solche Zusatzinformationen zu einem anderen Formular anzuzeigen (siehe *GDI-SubClass*).

Fenster ohne gleichzeitige Aktivierung einblenden

Wenn Sie ein Fenster über die *OpenForm*-Methode von Access anzeigen wollen, werden Sie feststellen, dass Access dies nur bei gleichzeitiger Aktivierung des Fensters erlaubt. Auch wenn ein schon längst geöffnetes, aber verborgenes Fenster angezeigt werden soll, erlaubt Access dies nur bei gleichzeitiger Aktivierung desselben.

Wenn Sie ein Statusfenster anzeigen wollen, soll dieses jedoch meist nicht aktiviert werden. Entweder weil das vormalig aktuelle Fenster den Eingabefokus nicht verlieren soll, da somit alle Tastatureingaben an das Statusfenster gehen, oder weil ein störendes Geflicker entsteht, da zuerst das Statusfenster und sofort danach das alte Fenster durch einen Mausklick aktiviert wird. Diesem Umstand kann jedoch durch die Verwendung der bekannten Funktion *SetWindowPos()* abgeholfen werden.

Sie haben diese Funktion schon im Zusammenhang mit dem Bewegen bzw. Vergrößern eines Fensters kennen gelernt und auch schon als Werkzeug zur Erzeugung von Vordergrundfenstern erlebt. Diese Funktion ist der Dreh- und Angelpunkt des gesamten Fenstermanagements von Windows. Durch Angabe geeigneter Flags im letzten Parameter können Sie *SetWindowPos()* dazu bringen, ein Fenster anzuzeigen, ohne es zu aktivieren. Allerdings müssen Sie durch Angabe geeigneter Flags dafür sorgen, dass die aktuelle Position und Größe des Fensters nicht verändert werden:

```
SetWindowPos Me.Hwnd, 0, 0, 0, 0, 0, SWP_NOSIZE Or  
SWP_NOMOVE Or SWP_NOZORDER Or SWP_SHOWWINDOW Or  
SWP_NOACTIVATE
```

Durch die Angabe der Flags *SWP_SHOWWINDOW* und *SWP_NOACTIVATE* wird ein ehemals verborgenes Fenster angezeigt, ohne gleichzeitig den Eingabefokus zu erhalten.





Maßeinheiten

Deutsche und
englische
Eigenschaftsnamen

Ein Problem, das Access noch immer nicht befriedigend löst, ist das Umrechnen von Maßeinheiten. Access versteht neben der Einheit Pixel auch die Einheiten cm und Twips. Nur leider wird keine Funktion geboten, die diese Einheiten ineinander überführt. Hinzu kommt, dass bei der interaktiven Programmierung – also der Eingabe von Formulareigenschaften im Eigenschaftsfenster – alles in der Einheit cm erledigt wird, während Sie beim Zugriff auf diese Eigenschaften mit VBA plötzlich Twips geliefert bekommen. (Twip = twentieth of a point = 1/20 pt, pt = 1/72 Inch, Inch = Zoll, Zoll = 2,54 cm). Weiterhin kommt erschwerend hinzu, dass beim Zugriff auf Position und Dimension eines Fenster mittels VBA englische Eigenschaftsnamen (*Left, Top, Width, Height*) verwendet werden.

Wenn Sie später in diesem Kapitel die Grafikfähigkeiten von Windows ausloten, werden Sie feststellen, dass alle Grafikbefehle mit der Einheit Pixel arbeiten. Wenn Sie also neben einem Steuerelement eine eigene Grafik zeichnen wollen, müssen Sie die rechte Kante dieses Steuerelements ermitteln und die Position der Kante von Twips in Pixel umwandeln um dann Ihre Grafikfunktion mit dieser Koordinate zu füttern.

Wie Twips mit anderen Maßeinheiten zusammenhängen, erkennen Sie aus der folgenden Beziehung:

$$1 \text{ TWIP} = 1/20 \text{ pt} = 1 / 20 * (1/70 \text{ INCH}) = 1/1440 \text{ INCH} \text{ (Zoll)}$$

Diese voneinander abhängigen Maßeinheiten erlauben eine Abbildung von Twips auf die Einheit Zoll. Und dies ist die Einheit, über die das System mehr weiß: Windows kann Ihnen nämlich sagen, wie viele Bildschirmpunkte in einem Zoll dargestellt werden können. Mit diesem Wissen können Sie eine recht einfache Funktion entwickeln, die Twips in Pixel umrechnet.

Nur, wie erhalten Sie die Anzahl von Punkten innerhalb eines Zolls? Windows stellt hierzu eine Funktion bereit, die ausgehend von einem Gerätekontext die Anzahl der Pixel innerhalb eines Zolls sowohl in vertikaler als auch in horizontaler Richtung ermittelt:

```
Declare Function GetDeviceCaps% Lib "GDI" (ByVal hDC%,  
ByVal caps%)
```

hDC eines Fensters

Das einzige Problem dieser Funktion besteht darin, dass ihr ein *hDC* übergeben werden muss. *hDC* steht für *Handle auf einen Device-Kontext*. Solche Device- oder Gerätekontexte sind eine Einrichtung von Windows, die es erlauben, verschiedene Ausgabegeräte weitgehend gleich zu behandeln. Sie erlauben die Programmierung einer Grafik ohne Rücksicht darauf, ob diese auf dem Bildschirm angezeigt oder auf einem Drucker ausgedruckt werden soll. Wo die Grafik letztendlich angezeigt wird, entscheidet allein der *hDC*.



Allerdings besitzen Access-Formulare keine Eigenschaft, die den Zugriff auf den DC eines Formulars erlaubt. Mit den Funktionen *GetDC()* und *ReleaseDC()* können Sie sich aber kurzfristigen Zugang zu solch einem DC verschaffen und diesen nach Gebrauch auch wieder freigeben. Besonders das Freigeben ist sehr wichtig: Da DCs eine knappe Ressource sind, treten bei gleichzeitiger Allokation mehrerer DCs ziemliche Probleme beim Bildschirmaufbau auf.

Wenn Sie mit *GetDC()* einen Device-Kontext allokalieren, müssen Sie angeben, von welchem Fenster der DC ermittelt werden soll. Um möglichst unabhängig von geöffneten Formularfenstern zu bleiben, bietet es sich an, hierzu das Access-Hauptfenster heranzuziehen. Dies gilt allerdings nur in dem Fall, dass Sie Maßeinheiten über den DC ermitteln wollen. Soll eine Grafik in einem Fenster dargestellt werden, muss der DC des entsprechenden Fensters – durch Übergabe des Fenster-Handles – ermittelt werden.

```
Dim hDC%
hDC = GetDC( Application.hWndAccessApp ) "DC ermitteln
...
dummy = ReleaseDC( Application.hWndAccessApp, hDC )
"sehr wichtig!
```

Für unsere Zwecke – nämlich der Umrechnung von Twips in Pixel – kann der DC eines jeden Fensters verwendet werden, weil die eingeforderten Maßeinheiten für alle Bildschirmkontexte identisch sind. Dazu wird die Funktion aufgerufen, der ein DC übergeben werden muss, um die Anzahl der Bildschirmpunkte in einem Inch zu ermitteln. Dividiert man 1440 durch die so ermittelten Werte, erhält man die Anzahl der Twips, die durch einen Bildschirmpunkt repräsentiert werden:

```
Dim TwipsPixelX%, TwipsPixelY%
TwipsPixelX = 1440 / GetDeviceCaps(hDC, LOGPIXELSX)
TwipsPixelY = 1440 / GetDeviceCaps(hDC, LOGPIXELSY)
```

Die Werte in *TwipsPixelX* und *TwipsPixelY* werden nun dazu benutzt, die Twip-Koordinaten eines Steuerelements in Bildschirmkoordinaten umzurechnen:

```
Dim c As Control
...
r.left = c.left / TwipsPixelX
r.top = c.top / TwipsPixelY
r.right = r.left + c.width / TwipsPixelX
r.bottom = r.top + c.height / TwipsPixelY
```

Zu beachten ist, dass Windows die Definition eines Rechtecks durch Angabe des linken oberen und rechten unteren Punkts verlangt, während Access die Position und Dimension (Höhe und Breite) eines Steuerelements zur Verfügung stellt. Wenn man die Dimension zur Position addiert, erhält man jedoch die rechte untere Ecke des Steuerelements.

Twips und Pixel

Unterschiedliche
Dimensionsangaben
zw. Windows und
Access





Kindfenster

Mit den oben aufgeführten Berechnungen lässt sich das Rechteck eines Steuerelements relativ zu seinem Aufenthaltsort (Kopf-, Detail- oder Fußbereich) ermitteln. Will man in ein Access-Fenster zeichnen, muss man dies beachten.

Jedem Bereich sein eigenes Fenster

Access legt innerhalb eines jeden Formulars drei weitere Kindfenster an, jeweils eines für jeden Bereich. Selbst wenn Kopf- und Fußbereich ausgeblendet sind, werden diese Fenster angelegt – allerdings besitzen sie die Höhe 0 und sind damit praktisch unsichtbar. Dieses Verhalten vereinfacht die Ermittlung dieser Kindfenster. Windows verwaltet Kindfenster in einer linearen Liste. Hat man Zugriff auf diese Fensterliste, muss man nur die drei aufeinander folgenden Fenster der Klasse *OFormSub* ermitteln, und schon hat man Zugriff auf die Fenster, die Kopf-, Detail- und Fußbereich repräsentieren, und das sogar in dieser Reihenfolge.

Kindfenster ermitteln

Um die Kindfensterliste eines Fensters zu untersuchen, steht die Funktion *GetWindow()* zur Verfügung. Ausgehend vom Formularfenster kann man diese Funktion einsetzen, um das erste Kindfenster zu ermitteln:

```
Dim hWndChild&
Dim f As Form
```

```
hWndChild = GetWindow( f.hWnd, GW_CHILD )
```

Ausgehend vom ersten Kindfenster eines Formulars kann man sich dann in einer Schleife durch alle weiteren Kindfenster "hangeln", um zu untersuchen, ob diese zur Klasse *OFormSub* gehören und die entsprechenden Fenster bzw. deren Bildschirmkoordinaten z.B. in einer Struktur speichern:

```
"Struktur zur Aufnahme der Fenster-Handles eines
Access-Formulars
Type FormWindows
    hWndHeader As Long        "Fenster-Handle des
                                "Formularkopfes
    hWndDetail As Long        "Fenster-Handle des
                                "Detailbereichs
    hWndFooter As Long        "Fenster-Handle des
                                "Formularfußes
End Type
"GetFormWindows - Ermittelt Fenster-Handles des
Formularkopfes,
"    Detailbereichs und Formularfußes eines Formulars
"hWnd          - Fenster-Handle des Formulars
"fw            - Struktur zur Aufnahme der
Fenster-Handles
Sub GetFormWindows(ByVal hWnd As Long, fw As
FormWindows)
    Dim i As Integer "Zähler
    Dim cn As String "ClassName
```



```
I = 0
hWnd = GetWindow(hWnd, GW_CHILD)      "Erstes
                                       "Kindfenster des
"   Formularfensters ermitteln
"Weitersuchen, solange noch Kindfenster vorhanden
und die
"   Kopf-, Detail- und Fußfenster nicht bestimmt
wurden
While hWnd And (I <> 3)
    cn$ = String(255, 0)
"Stringspeicher für Klassenname allokieren
    cn = Left(cn, GetClassName(hWnd, cn, Len(cn)))
"Klassennamen ermitteln und gleichzeitig überflüssige
Zeichen
"   abschneiden
    If cn = "OFormSub" Then

"Kindfenster eines Formulars (Klasse OFormSub)
    Select Case I
        "Formularkopf-Handle
        Case 0: fw.hwndHeader = hWnd: I = 1

        "Detailbereichs-Handle
        Case 1: fw.hwndDetail = hWnd: I = 2
        "Formularfuß-Handle
        Case 2: fw.hwndFooter = hWnd: I = 3
    End Select
End If
hWnd = GetWindow(hWnd, GW_HWNDNEXT)
"Nächstes
"   Kindfenster ermitteln
Wend
End Sub
```

SubClassen eines Fensters

Ein Thema, das Access sehr stiefmütterlich abhandelt, ist die Grafikprogrammierung. Während ein Visual-Basic-Formular wenigstens über einige wenige Grafikmethoden (*Line*, *Circle* etc.) verfügt, sucht man solche bei Access vergebens. Sogar die Ereignisprozedur *Paint* wurde bei Access vollkommen unterschlagen.

Solch eine Ereignisprozedur wird benötigt, um festzustellen, wann der Inhalt des Fensters neu gezeichnet werden muss – beispielsweise weil sich die Fenstergröße verändert hat oder weil der Benutzer durch Verschieben des Fensters einen neuen Fensterbereich zum Vorschein gebracht hat.

Wie lässt sich nun eine solche Ereignisprozedur nachträglich zu einem Access-Formular hinzufügen? Mit ein wenig "schwarzer Magie" – versteckt in einem so genannten *SubClass-Control* – lassen sich beliebige Fensternachrichten in eine VBA-

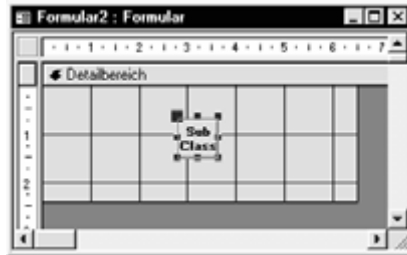




Ereignisprozedur umleiten. Solch ein Steuerelement fängt alle an ein Fenster gesendeten Nachrichten ab, um diese – inklusive der Nachrichtenparameter – über eine Ereignisprozedur bereitzustellen.

Abb. 14.3

Das *SubClass*-Zusatzsteuerelement



Das hier vorgestellte *SubClass*-Control erlaubt das Abfangen aller Standard-Fenster-nachrichten. Allerdings müssen Sie dem Control vorher mitteilen, an welchen Nachrichten Sie interessiert sind. Nach dem Einfügen des Controls in ein Formular können Sie dies über den folgenden Dialog bewerkstelligen, den Sie über den Menüpunkt *SubClass Control Objekt > Eigenschaften* öffnen. Sie finden diesen Menüpunkt im Kontextmenü des eingefügten *SubClass*-Zusatzsteuerelements.

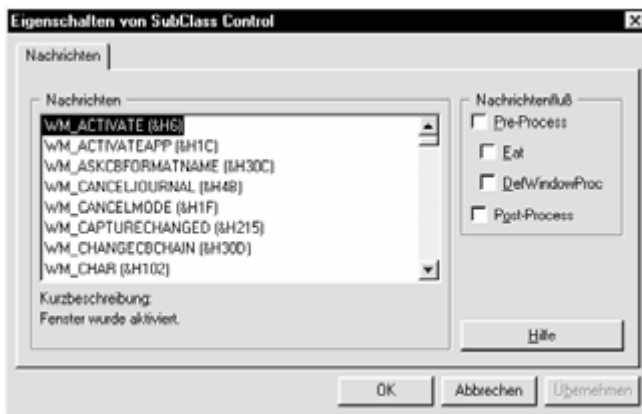


Abb. 14.4

Das Eigenschaften-fenster des *SubClass*-Zusatzsteuerelements

Nachrichtенbearbeitung

Hier können Sie für jede einzelne Nachricht festlegen, ob sie vor Ausführung der Original-Fensterfunktion (*PreProcess*) oder danach ausgeführt werden soll (*PostProcess*). Sie können die Ausführung der Original-Funktion allerdings auch verhindern, indem Sie die *Eat*-Option ("aufessen") einer Nachricht aktivieren. Wollen Sie die Ausführung der Originalfunktion aussetzen, um die Standard-Fensterfunktion auszuführen, müssen Sie die *DefWindowProc*-Option aktivieren.



Bitte haben Sie Verständnis dafür, dass wir an dieser Stelle nicht auf alle Fenster-nachrichten eingehen können – schließlich ist dies kein Windows-Programmier-handbuch.

SubClassen eines Fensters

Dem Control wird über die *SetWindow*-Methode das Fenster-Handle des Fensters übergeben, dessen Nachrichten abgefangen werden und in ein VBA-Ereignis umgeleitet werden sollen. Beim Aufruf der *SetWindow*-Methode können Sie jedes gültige Fenster-Handle angeben.

Erkennt das Control eine abzufangende Nachricht, wird diese in das Ereignis *WndProc* umgeleitet:

```
Sub WndProc( PostProcess As Boolean, hWnd As Long, _  
    uMsg As Long, wParam As Long, lParam As Long)
```

Der Parameter *PostProcess* zeigt an, ob das Ereignis nach Ausführung der Originalfunktion (*PostProcess = True*) oder vor der Ausführung dieser Funktion (*PostProcess = False*) aufgerufen wurde. Die restlichen Parameter entsprechen den Parametern einer Fensterfunktion:

- *hWnd* – das Fenster-Handle
- *uMsg* – die Nummer der abgefangenen Nachricht
- *wParam* und *lParam* – die nachrichtenspezifischen Parameter

Das folgende Codefragment zeigt auf, wie das *SubClass*-Control initialisiert und wie auf eingehende Nachrichten reagiert wird:

```
Private Sub Form_Open(Cancel As Integer)  
    SubClass1.SetWindow Me.hWnd \6Formularfenster  
SubClassen  
End Sub  
Private Sub SubClass1_WndProc(ByVal PostProcess As  
Boolean, _  
    ByVal hWnd As Long, ByVal uMsg As Long, _  
    ByVal wParam As Long, ByVal lParam As Long)  
    Select Case uMsg      "Nachrichten auswerten  
        Case msgWM_PAINT  
            cal_DrawCalendarDay CDim, CDay, -1  
        Case msgWM_ERASEBKGD  
            SubClass1.RetVal = -1  
    End Select  
End Sub
```





Wie Sie sehen, wird auf die einzelnen Nachrichten innerhalb einer *SELECT*-Anweisung reagiert. Die hier zum Einsatz gelangenden Nachrichtenkonstanten (*msgWM_PAINT*, *msgWM_ERASEBKGD*) stehen direkt nach dem Einbinden des Steuerelements in ein Formular zur Verfügung. Sie müssen keine weiteren Moduldateien in Ihre Datenbank einfügen, um diese Nachrichtenkonstanten zu nutzen.

Rückgabewerte

Oftmals erwartet das System einen Rückgabewert nach der Bearbeitung einer Nachricht. Da Ereignisprozeduren die Übergabe eines Rückgabewertes nicht erlauben, muss hierzu ein anderer Mechanismus eingesetzt werden. Über die Eigenschaft *RetVal* können Sie den zurückzugebenden Wert angeben (siehe *msgWM_ERASEBKGD*).

Wird eine Nachricht vor dem Aufruf der Original-Fensterfunktion in das *WndProc*-Ereignis umgeleitet (*PreProcess*), enthält diese Eigenschaft immer den Wert 0. Bei der Nachbearbeitung (*PostProcess*) finden Sie in *RetVal* den Rückgabewert der Original-Fensterfunktion, den Sie dann nach belieben überschreiben können.

Hilfsfunktionen

Bei der Übergabe eines Nachrichtenparameters werden oftmals mehrere Informationen innerhalb eines Langworts übergeben. So wird z. B. die Mausposition beim Senden der Fensternachricht *WM_MOUSEMOVE* im Parameter *lParam* bereitgestellt. Das niederwertige Wort enthält die X-, das höherwertige Wort die Y-Koordinate der Mausposition. Um diese Informationen aus einem Langwort herauszufischen, wird auf die Funktionen der *Acc7Hlp.dll* zurückgegriffen. Die Funktionen *LoWord()* und *HiWord()* dienen zur Extraktion des niederwertigen bzw. höherwertigen Wortes aus dem übergebenen Langwort.

Die Deklaration dieser Funktionen finden Sie innerhalb des Moduls *AccessHelper* in der Datenbank *Kapitel14.mdb*. Außerdem finden Sie den Quellcode dieser einfachen DLL auf der CD. Diese DLL stellt nichts anderes dar als einen so genannten *Wrapper*, der die in der Windows-Programmierung häufig eingesetzten Makros *HIWORD*, *LOWORD*, *HIBYTE* und *LOBYTE* darstellt.

Achtung! Achtung!

Wie schon erwähnt, stellt dieses Buch keinen Windows-Programmierkurs dar. Sollten Sie eigene Experimente in Bezug auf das SubClassen eines Access-Formulars unternehmen, sollten Sie beachten, dass Sie ganz nah am System operieren. Leider ist das Verhalten eines Access-Formulars bezüglich der Fensternachrichten nicht so gut dokumentiert wie dies z. B. unter Visual Basic der Fall ist. Das bedeutet, dass größte Vorsicht bei der Programmierung eigener Nachrichtenfunktionen geboten ist. Sollten Sie sich über die Art der einer Nachricht übergebenen Parameter oder der Bedeutung eines Rückgabewertes nicht im klaren sein, sollten Sie auf jeden Fall die Win32-SDK-Hilfe (*Win32API.hlp*) oder ein entsprechendes Windows-Programmierhandbuch (z. B. Charles Petzold: *Windows-Programming*, Microsoft Press) zu Rate ziehen.





Bevor das *SubClass*-Control eine Fensternachricht in die Ereignisprozedur *WndProc* umleiten kann, muss gewährleistet sein, dass im Eigenschaftendialog die entsprechende Nachricht aktiviert wurde. Würden standardmäßig alle Nachrichten in diese Ereignisprozedur umgeleitet, würde dies die Ablaufgeschwindigkeit Ihrer Anwendung extrem herabsetzen.



Hinweis

Grafikprogrammierung

Nachdem Sie nun das Werkzeug zum SubClassen eines Fensters in Händen halten, um das zum Aktualisieren des Fensterinhalts benötigte *OnPaint*-Ereignis nachzubilden, wollen wir Ihnen nun die Grafikfunktionen näher bringen. Wie Sie schon aus vorherigen Kapiteln erfahren haben, enthält ein Access-Formular drei Kindfenster, die jeweils den Kopf-, Detail- oder Fußbereich eines Formulars darstellen. Wenn Sie Ihre Grafikausgaben sehen wollen, müssen Sie also in eines dieser Fenster zeichnen – das Zeichnen in das über *Me.hWnd* spezifizierte Fenster führt meist nicht zum gewünschten Ergebnis, da der Fensterinhalt des Formularfensters von den drei soeben erwähnten Kindfenstern verdeckt wird.

Handle eines Unterformulars

Das Zeichnen in eines der Formular-Kindfenster ist jedoch auch nicht ganz unproblematisch: Befinden sich innerhalb des Bereichs Steuerelemente, werden diese von den eigenen Grafiken übermalt. Um dies zu verhindern, werden alle Grafikausgaben innerhalb eines eigenen Fensters getätigt. Hierzu wird ein ungebundenes Steuerelement zur Aufnahme eines Unterformulars in ein Access-Formular eingefügt. Die Aufgabe besteht nun zunächst darin, das Handle dieses Unterformular-Containers zu ermitteln. Hier hilft der gleiche Trick weiter, der Ihnen schon die Handles von Kopf-, Detail- und Fußbereich geliefert hat: Beim Durchlaufen der Kindfenster eines Bereichs wird das Fenster-Handle zurückgegeben, das einem Fenster der Klasse *OFormChild* entspricht:

```
"GetChildForm - Ermittelt das Fenster-Handle eines
Unterformulars
"      innerhalb des angegebenen Bereichs
"hWndParent  - Fenster-Handle des Kopf-, Detail oder
Fußbereichs,
"      in den ein Unterformular platziert wurde
"hwnd        - vorheriges Unterformular-Handle oder 0,
um erstes
"      Unterformular-Handle zu ermitteln
Function GetChildForm(hWndParent As Long, ByVal hWnd
As Long) _
    As Long
    Dim cn As String

    If hWnd = 0 Then "Suche starten
        hWnd = GetWindow(hWndParent, GW_CHILD)
    Else "Suche fortführen
        hWnd = GetWindow(hWnd, GW_HWNDNEXT)
```





```

End If
While hWnd
    cn = String(255, 0)    "Stringspeicher für
                           "Klassenname allokieren
    cn = Left(cn, GetClassName(hWnd, cn, Len(cn)))
    "Klassennamen ermitteln und gleichzeitig
    "überflüssige Zeichen abschneiden
    If cn = "OFormChild" Then "Unterformular?
        GetChildForm = hWnd    "Ja, Handle
zurückgeben
    Exit Function
End If
hWnd = GetWindow(hWnd, GW_HWNDNEXT)

    "Nächstes Kindfenster
Wend
End Function

```

Um z.B. das Handle des ersten Unterformulars innerhalb des Detailbereichs eines Formulars zu erhalten, wird folgende Codesequenz eingesetzt:

```

GetFormWindows Me.hWnd, fw

"Das erste Unterformular innerhalb des Detail-Bereichs
ermitteln
hwndUnterformular = GetChildForm(fw.hwndDetail, 0)

```

Das so erhaltene Unterformular kann dann an das SubClass-Control übergeben werden (*SetWindow*), das daraufhin z.B. die *WM_PAINT*-Nachrichten zum Neuzeichnen des Fensterinhalts abfängt.

Nach dem SubClassen sollten Sie durch einen Aufruf der Funktion *InvalidateRect* jedoch dafür sorgen, dass der Fensterinhalt für ungültig erklärt wird, woraufhin automatisch eine Aufforderung zum Neuzeichnen via *WM_PAINT* an das Fenster gesendet wird.

Zeichnen in Fenstern

Um Grafikausgaben in einem Fenster zu tätigen, benötigen Sie einen Gerätekontext. Dieser wird über die Funktion *GetDC()* ausgehend von einem Fenster-Handle ermittelt. Beim anschließenden Aufruf von Grafikfunktionen wird dieser DC übergeben. Nach der Ausgabe der Grafik wird der DC dann durch *ReleaseDC()* wieder freigegeben. Das folgende Codefragment zeigt, wie diese Schritte beim Erhalt der *WM_PAINT*-Nachricht durchgeführt werden:

```

case msgWM_PAINT
    hDC = GetDC( hWnd )
    ...
    ReleaseDC hWnd, hDC

```



Wenn Sie einen DC in Händen halten, steht Ihnen das gesamte GDI mit seinen zahlreichen Funktionen zur Verfügung. Bitte haben Sie auch hier wieder Verständnis dafür, dass wir nicht näher auf Belange der Grafikprogrammierung eingehen können, weil dieses Thema ganze Bände zu füllen vermag. Ohne eine kurze Übersicht möchten wir Sie allerdings nicht ziehen lassen. Zudem finden Sie in der Datenbank *Kapitel14.mdb* das Formular *GDI-Subclass*, das den Einsatz des *SubClass*-Controls und die Nutzung der GDI-Funktionen recht eindrucksvoll demonstriert: In ein Unterformularfenster wird – ähnlich wie bei *Schedule+* – eine Übersicht über die Termine eines Tages gezeichnet.

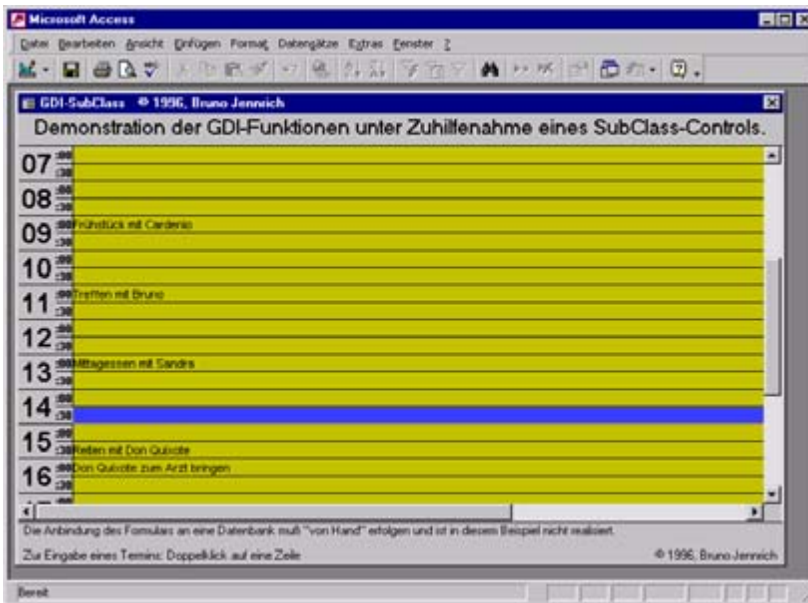


Abb. 14.5

... aus dem Termin-
kalender eines
gestressten
Programmierers

Es folgt die Zusammenfassung der zum Einsatz gelangenden API-Funktionen.

Textausgabe und Zeichensätze

Mit der Funktion *ExtTextOut()* können Sie innerhalb eines DC an einer beliebigen Stelle einen Text ausgeben. Die Textausgabe erfolgt im aktuell ausgewählten Zeichensatz. Allerdings kann *ExtTextOut()* noch mehr: Durch Übergabe des Flags *ETO_CLIPPED* können Sie die Funktion dazu veranlassen, den Text nur innerhalb des angegebenen Rechtecks zu schreiben. Sollte der Text aus diesem Rechteck herausragen, wird er einfach sang- und klanglos abgeschnitten. Das Flag *ETO_OPAQUE* sorgt dafür, dass der Hintergrund des Textes bzw. der Hintergrund des übergebenen Rechtecks mit der aktuellen Texthintergrundfarbe gefüllt wird. Diese Hintergrundfarbe wird durch den Aufruf der Funktion *SetBkColor()* festgelegt, der einfach der RGB-Wert der zu verwendenden Farbe übergeben wird. In welcher Farbe





der Text selbst erscheinen soll, wird durch einen Aufruf der Funktion *SetTextColor()* angegeben. Standardmäßig erfolgt die Textausgabe in Schwarz auf weißem Grund.

Die Textausrichtung wird durch die Funktion *SetTextAlign()* festgelegt. Ob linksbündiger (Standard), rechtsbündiger oder zentrierter Text ausgegeben werden soll, entscheiden die Flags *TA_LEFT*, *TA_RIGHT* bzw. *TA_CENTER*.

Wollen Sie einen anderen Zeichensatz verwenden, müssen Sie diesen über einen der *CreateFont...()*-Befehle erzeugen. Im Modul *Calendar* der Datenbank *Kapitel14.mdb* finden Sie eine Funktion namens *CreateSimpleFont()*, die nur den Namen der zu erzeugenden Schriftart, die Höhe sowie den Schriftschnitt (*FW_BOLD* (Fett), *FW_NORMAL*) entgegennimmt. Das von dieser Funktion zurückgegebene Font-Handle kann dann via *SelectObject()* in den DC selektiert werden und bestimmt so den zur Textausgabe zu verwendenden Zeichensatz.

Alle Funktionen, die die Attribute oder Objekte eines DC ändern, liefern als Rückgabewert den alten Attributwert bzw. das alte Objekt-Handle. Diesen Wert sollten Sie zwischenspeichern, um den DC nach der Benutzung – jedoch vor der Freigabe – wieder zu restaurieren.

Dies ist insbesondere bei der Funktion *SelectObject()* zu beachten. Über sie wird ein neuer Zeichensatz, ein neuer Zeichenstift, ein neuer Pinsel etc. zur Benutzung in nachfolgenden Grafikfunktionen ausgewählt. Da diese Objekte u.U. sehr viele Ressourcen beanspruchen, müssen bei der Selektion eines neuen Objekts die alten Objekte zwischengespeichert werden, um sie später zu reselectieren. Tun Sie dies nicht, schwebt ein Objekt haltlos in Ihrem Adressraum herum und allokiert Speicherplätze – geschieht dies allzu oft, wird Windows unerträglich langsam und kann sogar abstürzen!

Linien und Rechtecke

Zur Ausgabe von Linien und Rechtecken stehen die Funktionen *MoveToEx()*/*LineTo()* und *Rectangle()* bereit. Mit *MoveToEx()* legen Sie den Startpunkt einer Linie fest, mit *LineTo()* veranlassen Sie die Zeichnung der Linie bis zur in *LineTo()* angegebenen Koordinate. *Rectangle()* erwartet als Parameter die Koordinaten der linken, oberen und rechten unteren Ecke des zu zeichnenden Rechtecks, um dann ein im aktuellen Zeichenstift (*Pen*) gerahmtes und mit dem aktuellen Pinsel (*Brush*) gefülltes Rechteck zu zeichnen.

Wenn Sie nur einen Rahmen zeichnen wollen, können Sie sich auf die Funktion *FrameRect()* berufen, die einen Rahmen um das in einer *RECT*-Struktur übergebene Rechteck mit dem aktuell ausgewählten Pinsel zeichnet. Der Inhalt des Rechtecks wird nicht verändert. Gleiches gilt für die Funktion *DrawFocusRect()*, die einen gestrichelten Rahmen zeichnet, der unter Windows zur Hervorhebung der aktuellen Selektion (z. B. in einem Listenfeld) verwendet wird.





Objekte freigeben

Es kann nicht eindringlich genug darauf hingewiesen werden, wie wichtig es ist, allokierte Objekte wieder ordnungsgemäß freizugeben. Ein häufiger Programmierfehler besteht darin, allokierte Objekte einfach beziehungslos im Adressraum herumschweben zu lassen. Dabei muss ein via *CreateFont...()*, *CreatePen()*, *CreateBrush()* etc. erzeugtes Zeichnungsobjekt durch *DeleteObject()* wieder freigegeben werden. Allerdings erst, nachdem es aus dem DC herausgenommen wurde. Dies geschieht durch Resektion des von *SelectObject()* zurückgelieferten Handles. Leider ist das GDI ein wenig inkonsistent, weil nur eine Funktion zur Selektion von Pinseln, Zeichenstiften und Zeichensätzen eingesetzt wird – allerdings ist diese Funktion so intelligent, dass sie aufgrund des übergebenen Handles den Typ des übergebenen Objekts eindeutig bestimmen kann.

Hintergrund löschen

Beim Neuzeichnen eines Fensters wird immer eine *WM_ERASEBKGD*-Nachricht an das Fenster gesendet, um zu gewährleisten, dass der Fensterhintergrund vor dem eigentlichen Zeichnen gelöscht wird. Üblicherweise wird hier einfach der neu zu zeichnende Bereich mit der Hintergrundfarbe des Fensters gefüllt. Wenn Sie jedoch gewährleisten können, dass die freiwerdenden Bereiche innerhalb der Bearbeitung von *WM_PAINT* neu gezeichnet werden, können Sie Windows das Löschen des Hintergrundes ersparen: Dazu reagieren Sie auf den Erhalt der Nachricht *WM_ERASEBKGD* mit der Rückgabe von *-1*. Damit signalisieren Sie Windows, dass alles Nötige zum Löschen des Hintergrundes unternommen wurde. Allerdings darf die *WM_ERASEBKGD*-Nachricht nicht an die Original-Fensterfunktion weitergeleitet werden (*Eat*).

Bildlaufleisten

Leider stellt Access XP kein Steuerelement bereit, mit dessen Hilfe Bildlaufleisten dargestellt werden können. Mithilfe des *SubClass*-Controls und einiger API-Aufrufe lassen sich jedoch die in ein Fenster eingebauten Bildlaufleisten nutzen. Ausgehend von dem Unterformularfenster werden mit *ShowScrollBar()* die in ein Fenster eingebauten Bildlaufleisten angezeigt:

```
ShowScrollBar hWnd, SB_BOTH, True
```

Durch Angabe von *SB_VERT* bzw. *SB_HORZ* können Sie dafür sorgen, dass nur der vertikale (senkrechte) oder horizontale (waagerechte) Schieberegler dargestellt wird.

Nach der Anzeige der Bildlaufleisten müssen die internen Strukturen eines Schiebereglers initialisiert werden. Sie müssen angeben, welchen Bereich der Schieberegler abdeckt (minimaler und maximaler Wert) und festlegen, wie groß der Bereich ist, den der Schieberegler selbst einnimmt. Über diese Größe kann die Höhe oder Breite des Schiebereglers proportional zum Gesamtbereich eingestellt werden. Die Werte werden über die Funktion *SetScrollInfo()* eingestellt:





```
"Scrollbereich (RANGE)
und Größe des Schiebereglers (PAGE) setzen
si.cbSize = Len(si)
si.fMask = SIF_PAGE Or SIF_RANGE
si.nPage = 50
si.nMin = 0
si.nMax = 1000
SetScrollInfo hWnd, SB_HORZ, si, True
```

Sie muss sowohl für die vertikale als auch für die horizontale Bildlaufleiste aufgerufen werden. Nachdem die Bildlaufleisten sichtbar sind und die internen Strukturen gesetzt wurden, müssen die von diesen Steuerelementen ausgehenden Nachrichten bearbeitet werden.

Scrollbar-Ereignisse

Dazu sendet Windows die Nachrichten *WM_HSCROLL* oder *VM_VSCROLL* an das Fenster, je nachdem, welche Bildlaufleiste durch Mauseaktionen verändert wurde. Durch Aufschlüsseln der mit diesen Nachrichten übergebenen Parameter können Sie genau feststellen, ob der Schieberegler selbst verschoben wurde, ob innerhalb des Bildlaufleistenbereichs geklickt wurde oder ob der Benutzer eine der Pfeilschaltflächen gedrückt hat:

```
Private Sub SubClass1_WndProc(ByVal PostProcess As
Boolean, ByVal hWnd As Long, ByVal uMsg As Long, ByVal
wParam As Long, ByVal lParam As Long)
    Select Case uMsg

        "auf waagerechten Schieberegler reagieren
        Case msgWM_HSCROLL
            Select Case LoWord(wParam)
                "Schiebereglercode
                Case SB_BOTTOM
                    PosX = 1000
                Case SB_LINELEFT
                    PosX = PosX - 10
                Case SB_LINERIGHT
                    PosX = PosX + 10
                Case SB_PAGELEFT
                    PosX = PosX - 50
                Case SB_PAGERIGHT
                    PosX = PosX - 50
                Case SB_THUMBPOSITION
                    PosX = HiWord(wParam)
                Case SB_TOP
                    PosX = 0
                Case SB_ENDSCROLL
                Case SB_THUMBTRACK
                    "Bei langsamer Bildschirmdarstellung
                    "evtl. abschalten
```





```
        PosX = HiWord(wParam)
    End Select

    "Neue Schiebereglerposition setzen
    SetScrollPos hWnd, SB_HORZ, PosX, True

    "Neuzeichnen des Fensterbereichs
    InvalidateRect hWnd, ByVal 0&, True

    "auf senkrechten Schieberegler reagieren
    Case msgWM_VSCROLL
        Select Case LoWord(wParam)
            Case SB_BOTTOM
                PosY = 1000
            Case SB_LINEDOWN
                PosY = PosY + 10
            Case SB_LINEUP
                PosY = PosY - 10
            Case SB_PAGEDOWN
                PosY = PosY + 50
            Case SB_PAGEUP
                PosY = PosY - 50
            Case SB_THUMBPOSITION
                PosY = HiWord(wParam)
            Case SB_TOP
                PosY = 0
            Case SB_ENDSCROLL
            Case SB_THUMBTRACK
                "Bei langsamer Bildschirmdarstellung
                "evtl. abschalten
                PosY = HiWord(wParam)
        End Select

        SetScrollPos hWnd, SB_VERT, CDim.Top, True
        InvalidateRect hWnd, ByVal 0&, True
    End Select
End Sub
```

In dem oben abgedruckten Fragment einer Bearbeitungsfunktion für die *WM_HSCROLL*- und *WM_VSCROLL*-Ereignisse sehen Sie, wie aus dem *wParam*-Parameter der Nachricht der Aktionscode extrahiert wird. Dieser Code gibt an, ob der Schieberegler mit der Maus gefasst und verschoben wurde (*THUMBTRACK*, *THUMBPOSITION*), ob eine der Pfeilschaltflächen gedrückt wurde (*LINELEFT*, *LINERIGHT*, *LINEUP*, *LINEDOWN*) oder ob innerhalb des Bildlaufleistencontainers geklickt wurde (*PAGELEFT*, *PAGERIGHT*, *PAGEUP*, *PAGEDOWN*). Sie müssen auf diese Aktionscodes reagieren, indem Sie Ihre internen Variablen entsprechend inkrementieren bzw. dekrementieren.





Damit die neue Position des Schiebereglers angezeigt wird, ist ein Aufruf der Funktion *SetScrollPos()* nötig. Erst dann wird der Schieberegler an die neue Position gesetzt. Damit der Bildschirminhalt an die neue Position angepasst wird, sollte ein Aufruf der Funktion *InvalidateRect()* erfolgen. Sie sorgt dafür, dass der Fensterinhalt via *WM_PAINT* neu gezeichnet wird. Dort kann der Zustand Ihrer internen Variablen ausgelesen werden, um z. B. einen anderen Teil einer Seite darzustellen.

Über den Aktionscode *THUMBTRACK* kann eine laufende Aktualisierung des Bildschirminhalts erfolgen. Jedes Mal, wenn sich die Position des Schiebereglers ändert, wird eine *WM_xSCROLL*-Nachricht an das Fenster gesendet, woraufhin die aktuelle Schiebereglerposition dem Nachrichtenparameter *lParam* entnommen werden kann (*HiWord(lParam)*). Allerdings erweist sich Access meist nicht als schnell genug, um eine direkte Aktualisierung des Fensterinhalts vorzunehmen – Sie könnten diese Nachricht jedoch dazu verwenden, ein Hilfsfenster anzuzeigen, in dem die mit dem Schieberegler angefahrne Position angezeigt wird. (Ein ähnliches Verhalten weist Access beim "Herumscrollen" innerhalb eines Endlosformulars auf: Dort wird nicht sofort der der Schiebereglerposition entsprechende Datensatz angezeigt, sondern zunächst eine Art Vorschau über ein kleines Fenster bereitgestellt, das die Nummer des ausgewählten Datensatzes beinhaltet.)

Lässt der Benutzer den Schieberegler wieder los (durch Loslassen der linken Maustaste), wird an das Fenster ein *THUMBPOSITION*-Aktionscode versendet – spätestens hier sollten Sie dann den übergebenen Wert in Ihren internen Variablen speichern und eine Neuzeichnung des Fensterinhalts veranlassen.

Den Einsatz der eingebauten Schieberegler eines Formulars können Sie aus dem Formular *GDI-Subclass* der mitgelieferten Datenbank *Kapitel4.mdb* ersehen.

Mausereignisse und Tastatur

Wenn Sie wie wir ein Unterformularfenster einsetzen, um darin zu zeichnen, werden Sie feststellen, dass nicht nur ein *Paint*-Ereignis fehlt, sondern auch Ereignisse zur Maus- und Tastaturbearbeitung. Aber auch hier kann durch Umleiten der entsprechenden Fensternachrichten Abhilfe geschaffen werden.

Durch Umleiten der Nachrichten *WM_LBUTTONDOWN*, *WM_LBUTTONUP*, *WM_LBUTTONDOWNBLCLK* und *WM_MOUSEMOVE* können Sie das Niederdrücken, Loslassen und Doppelklicken der linken Maustaste registrieren. Mit *WM_MOUSEMOVE* werden Sie über die Bewegungen der Maus informiert – sobald sich die Maus über dem Fenster befindet, dessen Nachrichten umgeleitet werden.

All diesen Nachrichten wird im *lParam*-Parameter die aktuelle Mauskoordinate bezüglich der Client-Area (dem eigentlichen Zeichenbereich eines Fensters) übergeben. Das niederwertige Wort (*LoWord*) enthält die X-Koordinate, das höherwertige Wort (*HiWord*) die Y-Koordinate.



Diese Koordinaten werden im GDI-SubClass-Formular zur Anzeige eines kleinen Hilfefensters herangezogen. Allerdings werden sie vorher noch in Bildschirmkoordinaten umgewandelt – denn zur Positionierung eines Popup-Fensters müssen die Koordinaten als Bildschirmkoordinaten vorliegen.

Diese Umwandlung geschieht durch einen Aufruf der Funktion *ClientToScreen()*:

```
Dim p As POINTAPI
p.X = LoWord(ByVal lParam)
p.Y = HiWord(ByVal lParam)
"Client-Koordinaten in Screen-Koordinaten
ClientToScreen hWnd, p
```

Dazu werden die Koordinaten zunächst in einer *POINTAPI*-Struktur abgelegt, die dann der Funktion *ClientToScreen()* zur Umwandlung übergeben wird. Nach dem Aufruf der Funktion finden Sie in der *POINTAPI*-Struktur die gewandelten Koordinaten.

Einfangen der Maus

Wie schon angedeutet, werden die Maus-Nachrichten nur dann an ein Fenster gesendet, wenn sich die Maus direkt über dem Fenster befindet. Das ist aber oft nicht ausreichend – vielmehr will man in bestimmten Fällen über alle Mausbewegungen und Maustastendrucke informiert werden. Zu diesem Zweck muss man die Maus einfangen. Nach dem Aufruf von *SetCapture()* werden alle nachfolgenden Mausereignisse an das im Aufruf angegebene Fenster gesendet – egal, wo sich die Maus befindet.

Das führt unter Umständen dazu, dass Mauskoordinaten gemeldet werden, die außerhalb des Fensters liegen – und evtl. sogar negativ sind. Bevor Sie mit diesen Koordinaten weiterarbeiten, sollten Sie die gemeldeten Koordinaten also einer Prüfung unterziehen.

Bei "eingefangener" Maus, wird auch dann das Loslassen einer Maustaste gemeldet, wenn sich die Maus nicht mehr über dem Fenster befindet. So lässt sich beim Empfang der *WM_LBUTTONDOWN*-Nachricht das im GDI-SubClass-Formular verwendete Hinweisfenster immer ausblenden.

Um die Maus wieder freizugeben, muss die Funktion *ReleaseCapture* aufgerufen werden.

Tastatur

Wenn Sie innerhalb eines Fensters auf Tastatureingaben reagieren wollen, müssen Sie die Nachrichten *WM_KEYUP*, *WM_KEYDOWN* oder *WM_CHAR* abfangen. *WM_KEYUP* und *WM_KEYDOWN* liefern den so genannten *virtuellen Tastencode* einer niedergedrückten bzw. losgelassenen Taste. Dieser wird über den *wParam*-Parameter geliefert. Die Definition der unterschiedlichen virtuellen Tastencodes finden Sie in der Datei *Win32API.txt*.



Mit `WM_CHAR` erhalten Sie im `wParam`-Parameter den ANSI-Code einer niedergedrückten Taste. Im Gegensatz zu den `WM_VKEY...`-Nachrichten werden hier keine Sondertasten (F1 etc.) gemeldet, sondern nur tatsächlich druckbare Zeichen (A-Z, a-z, 0-9, etc.).

Windows Registry

Unter Windows ab Version 95 werden `.ini`-Dateien zum Speichern applikationsspezifischer Daten nur noch sehr spärlich eingesetzt. Dafür wird die so genannte *Registatur-Datenbank*, auch als *Registry* bezeichnet, intensiv genutzt. Hier werden Informationen über OLE-Server genauso gespeichert wie die Daten zur Wiederherstellung Ihres Applikationsfensters, der zuletzt geöffneten Dateien etc.

Wenn Sie sich die Registry mithilfe des im Lieferumfang von Windows enthaltenen Registry-Editors (`RegEdit.exe`) betrachten, werden Sie feststellen, dass eine Unmenge an Informationen dort abgelegt wurde. Aber keine Angst – Sie kommen nur mit einem ganz kleinen Teil davon in Berührung.



Zum Start des Registrierungseditors wählen Sie den Menüpunkt *Ausführen* des *Start*-Menüs von Windows aus. Im Dialogfenster *Ausführen* geben Sie unter *Öffnen* den Text `RegEdit` ein und bestätigen mit *OK*.

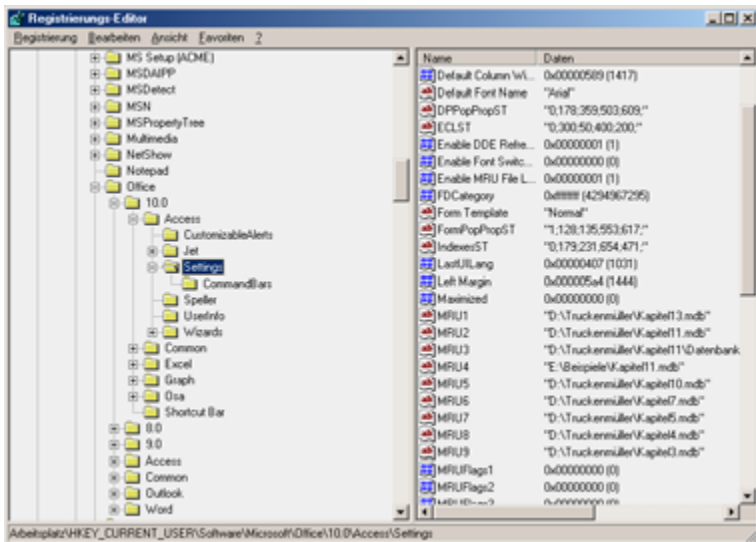


Abb. 14.6

Die öffentlichen Registry-Einträge von Microsoft Access 10.0

In der Terminologie der Registry wird zwischen Schlüsseln (*Keys*) und Werten (*Values*) unterschieden. Ein Wert kann nur unterhalb eines Schlüssels abgelegt werden. Erst wenn Sie sich Zugang zu einem Schlüssel verschafft haben, können Sie die unter dem Schlüssel gespeicherten Werte bearbeiten.





Schlüsselwerte speichern

Der Ort, an dem Sie Ihre applikationsspezifischen Daten speichern dürfen, wird durch einen Schlüssel der Form *HKEY_CURRENT_USER\Software\<Hersteller>\<Produkt>\<Version>* bestimmt, wobei die Namen von *<Hersteller>*, *<Produkt>* und *<Version>* von Ihnen bereitgestellt werden müssen.

Folgendes Codesegment erzeugt einen neuen Schlüssel und legt den String *DatenWert* unter dem benannten Wert *Daten* ab:

```
Dim NewKey As Long
If RegCreateKey(HKEY_CURRENT_USER, _
    "Software\Hersteller\Produkt\Version", NewKey) _
    = ERROR_SUCCESS Then
    If RegSetValue(NewKey, "Daten", REG_SZ, ByVal _
        "DatenWert", 10) _
        = ERROR_SUCCESS Then

        "Schlüsselwert erfolgreich gesetzt"
    End If

    RegCloseKey NewKey
End If
```

Dazu wird mit *RegCreateKey()* zunächst der Schlüssel erzeugt. Als ersten Parameter wird dieser Funktion zunächst die Konstante eines der vordefinierten Registraturverzeichnisse übergeben. Darauf folgt die Angabe des in diesem Verzeichnis zu erzeugenden Unterverzeichnisses. In dem als Referenzparameter übergebenen Langwort wird das Handle des so erzeugten Schlüssels abgelegt.

Dieser Schlüssel kann dann zum Bearbeiten eines Wertes via *RegSetValue()* eingesetzt werden. Diese Funktion kann sowohl bestehende Werte überschreiben als auch neue erzeugen. Dazu werden der Name des Wertes, der Typ der zu speichernden Daten, die Daten selbst sowie die Größe der abzuspeichernden Daten übergeben.

Neben Strings können auch Langworte (Long) und Binärdaten (Strukturen) in der Registry abgelegt werden. Dazu wird statt *REG_SZ* der Typ *REG_DWORD* bzw. *REG_BINARY* angegeben. Beachten Sie, dass bei der Übergabe einer Referenz auf diese Variablen – anders als bei der Übergabe einer Stringreferenz – das Schlüsselwort *ByVal* nicht angegeben werden darf (siehe *Any*-Parameter).

Schlüsselwerte einlesen

Bevor Sie einen so angelegten Schlüsselwert wieder einlesen können, müssen Sie sich Zugang zu dem Schlüssel verschaffen, unter dem der Wert gespeichert wurde. Dies geschieht über die Funktion *RegOpenKeyEx()*. Durch anschließende Aufrufe von *RegQueryValueEx()* können die unter dem Schlüssel gespeicherten Werte eingelesen werden:





```

Dim Value As String
Dim cbValue As Long
Dim dtValue As Long
If RegOpenKeyEx(HKEY_CURRENT_USER, _
    "Software\Hersteller\Produkt\Version", ByVal 0&, _
    KEY_ALL_ACCESS, NewKey) = ERROR_SUCCESS Then
    Value = String(255, 0)
    cbValue = Len(Value)
    dtValue = REG_SZ
    if RegQueryValueEx(NewKey, "Daten", ByVal 0&, _
        dtValue, ByVal Value, cbValue) _
        = ERROR_SUCCESS Then
        Value = Left(Value, cbValue)
    End If
    RegCloseKey NewKey
End If

```

Während *RegOpenKeyEx()* noch recht unkompliziert ist, muss bei *RegQueryValueEx()* beachtet werden, dass nahezu alle Parameter als Referenz übergeben werden. So wird der Typ des einzulesenden Wertes per Referenz übergeben (*dtValue*) – wie der Speicherplatz selbst – und auch die Größe des von Ihnen bereitgestellten Speicherplatzes wird als Referenz auf eine *Long*-Variable bereitgestellt. In dieser *Long*-Variablen legt die Funktion *RegQueryValueEx()* die Größe der tatsächlich eingelesenen Daten ab. So lässt sich – wie im aufgezeigten Beispiel – ein eingelesener String auf die tatsächliche Anzahl eingelesener Zeichen beschränken.

Schlüssel freigeben

Achten Sie darauf, dass ein durch *RegCreateKey()* bzw. *RegOpenKeyEx()* erzeugter bzw. geöffneter Schlüssel immer durch *RegCloseKey()* geschlossen bzw. freigegeben wird – ansonsten könnte anderen Anwendungen der Zugriff auf diesen Schlüssel verweigert werden.

Registry-Klassenmodul

Zum guten Schluss lernen Sie noch ein Access-Klassenmodul kennen, das die Manipulation von Registry-Werten zum Kinderspiel macht. Die Definition der Klasse finden Sie im Modul *clsRegistry* in der Datenbank *Kapitel14.mdb* der beiliegenden CD. Ein entsprechendes Testmodul für die Registry-Klasse finden Sie ebenfalls dort unter dem Namen *TestRegistry*. Die Testfunktion schreibt und liest zwei Registry-Werte: einen numerischen Wert und einen String. Die eingelesenen Werte werden schließlich im Testfenster angezeigt.

Im Klassenmodul befinden sich eine Reihe von Modulen und Eigenschaften, mit denen die Registry-Werte abgefragt bzw. geändert werden können. Die folgende Tabelle gibt Ihnen eine Übersicht:



Übersicht über die Klasse *clsRegistry*

Name	Typ	Beschreibung
<i>TopKey</i>	Eigenschaft	Der Hauptschlüssel der Registry, z. B. <i>HKEY_CURRENT_USER</i> . Die relativen Konstanten finden Sie im Modul <i>Registry-Konstante und Tools</i> .
<i>Key</i>	Eigenschaft	Schlüssel, dessen Wert verarbeitet werden soll, z. B. <i>Software\TestProgram\TestKey</i> .
<i>ReadNum</i>	Methode	Liest einen Zahlwert aus der Registry.
<i>ReadString</i>	Methode	Liest einen Stringwert aus der Registry.
<i>WriteNum</i>	Methode	Schreibt einen Zahlwert in die Registry.
<i>WriteString</i>	Methode	Schreibt einen Stringwert in die Registry.

Tabelle 14.1

Ein Objekt der Klasse deklarieren Sie mit dem Schlüsselwort *New*:

```
Dim reg As New clsRegistry
```

Danach stehen Ihnen sämtliche Eigenschaften und Methoden und damit alle Registry-Werte über die Variable *reg* zur Verfügung. Bevor Sie einen Wert lesen bzw. schreiben können, müssen Sie den Schlüssel bestimmen, dessen Wert für Sie interessant ist. Als Hauptschlüssel (*TopKey*-Eigenschaft) können Sie eine der in der Windows-API definierten Konstanten, wie z. B. *HKEY_CURRENT_USER* oder *HKEY_LOCAL_MACHINE*, verwenden. Die Eigenschaft *Key* bestimmt den eigentlichen Schlüssel:

```
reg.TopKey = HKEY_CURRENT_USER
reg.Key = "Software\TestCompany\TestProgram\1.0\TestKey"
```

Nach diesen Einstellungen können Sie einen bzw. beliebig viele Einträge unter dem Schlüssel *TestKey* speichern. Um beispielsweise die Fensterposition des aktuellen Formulars zu speichern, setzen Sie das *reg*-Objekt in der Ereignisprozedur *Bei Entladen* folgendermaßen ein:

```
Private Sub Form_Unload(Cancel As Integer)

    Dim rctMe As RECT
    Dim rctAccess As RECT
    Dim hwndAccess As Long
    Dim reg As New clsRegistry

    hwndAccess = GetParent(Me.hwnd)
    GetWindowRect Me.hwnd, rctMe
    GetWindowRect hwndAccess, rctAccess
```





```

With reg
    .TopKey = HKEY_CURRENT_USER
    .Key = "Software\MeineFirma\MeineDatenbank\_
        1.0\Fensterposition"

    .WriteNum "Oben", rctMe.Top - rctAccess.Top
    .WriteNum "Links", rctMe.Left - rctAccess.Left
    .WriteNum "Unten", rctMe.Bottom - rctAccess.Top
    .WriteNum "Rechts", rctMe.Right - rctAccess.Left
End With

Set reg = Nothing

```

End Sub

Die gespeicherte Fensterposition können Sie dann beim nächsten Öffnen des Formulars wiederherstellen, indem Sie die Werte mit einem neuen *Registry*-Objekt einlesen. Hierfür eignet sich das *Laden*-Ereignis des Formulars:

```

Private Sub Form_Load()

    Dim lOben As Long, lLinks As Long
    Dim lUnten As Long, lRechts As Long
    Dim reg As New clsRegistry

    With reg
        .TopKey = HKEY_CURRENT_USER
        .Key = "Software\MeineFirma\MeineDatenbank _
            \1.0\Fensterposition"

        If Not .ReadNum("Oben", lOben) Then
            " Fensterkoordinaten waren nicht da
        Else
            .ReadNum "Links", lLinks
            .ReadNum "Unten", lUnten
            .ReadNum "Rechts", lRechts
            MoveWindow Me.hwnd, lLinks, lOben, _
                lRechts - lLinks, _
                lUnten - lOben, True
        End If
    End With

    Set reg = Nothing

End Sub

```

Um diese Idee zu erweitern, könnten Sie auch mehrere Formularpositionen in der Registry speichern, indem Sie weitere Schlüssel mit beispielsweise den Namen des jeweiligen Formulars anlegen. Jeder Schlüssel könnte dann die Fensterkoordinaten enthalten.



1

1\	
1-Beziehung	547
n-Beziehung	545
n-Beziehung auf sich selbst	558

A

Abfangen von Formularfehlern	368
Abfangen von Laufzeitfehlern	161
Abfragen	
ausführen	508
dynamisch zusammen-	
setzen	296
erzeugen	493
Abs-Funktion	83
Absolutwert einer Zahl	83
Access als DDE-Client	674
Access als DDE-Server	681
Access per OLE-Automation steuern	665
Access via DDE steuern	681
Action-Eigenschaft	640
ActiveControl-Eigenschaft	206
ActiveDataAccessPage-Eigenschaft	206
ActiveDatasheet-Eigenschaft	206
ActiveForm-Eigenschaft	206
ActiveReport-Eigenschaft	206
ActiveX Data Objects	264
ActiveX-Steuerelemente	667
Add-Prädikat	502
AddNew-Methode	259
AddToFavorites-Methode	411
Admin-Benutzer	607
Administrator	607
ADO	54, 263
AfterDelConfirm-Ereignis	342
AfterUpdate-Ereignis	373
Aktion, Meldung	3
Aktion ÖffnenFormular	8
Aktionsabfragen ausführen	235
Aktionsargumente	6
Aktivieren des Anmelde-	
verfahrens	611
Aktivieren von OLE-Objekten	641
Aktualisierungsabfragen definieren	303

aktuelle Datenbank	231
All-Prädikat	287
Alter Table-Anweisung	502
And-Operator	86
Anfügeabfragen definieren	302
Anlegen einer Prozedur	35
Anlegen einer Replikatgruppe	563
Anmeldeverfahren	
aktivieren	611
deaktivieren	611
Any	706
Anzahl Datensätze	
ermitteln	228, 239
API	691
API-Funktion	697
Append-Methode	474, 609
Application Programming Interface	690
Application-Eigenschaft	206
Application-Objekt	195, 665
ApplyFilter-Ereignis	349
arithmetische Operatoren	81
Arrays	130, 709, 710
Arrays, dynamisch	133
Asc-Funktion	97
Assistent zur Datenbank-	
aufteilung	562
Attributes-Eigenschaft	
von Beziehungen	490
von Feldern	481
von Indexfeldern	486
von Tabellen	474
Auflistungen	197-198, 203, 217
Aufrufe-Dialogfenster	168
Ausdrücke	78
Datum	100
Zeit	100
Ausführen einer Prozedur	72
Ausführen eines Makros	5
Ausführen von Makroaktionen	
in Access Basic	198
AusführenBefehl-Methode	335
AusführenSql-Anweisung	508
Auswahl einer Prozedur	56
Auswahl von Listeneinträgen	390
Auswahlabfragen definieren	286
Autoexec-Makro	190
automatische Daten-Tips	170
automatische QuickInfo	61
automatische	
Syntaxüberprüfung	60

B

Bedingte Ausführung von Aktionen	12
Bedingungen in Makros	12
Befehlsleisten	572
Befehlsschaltflächen	388, 413
Befehlssyntax	35
Befehlszeilenoptionen	598
BeforeDelConfirm-Ereignis	341
BeforeUpdate-Ereignis	373
Bei Zeitgeber-	
Eigniseigenschaft	367
BeiAktivierung-	
Eigniseigenschaft	365
BeiDeaktivierung-	
Eigniseigenschaft	365
BeiFokuserhalt	380
BeimDoppelklicken-	
Eigniseigenschaft	26, 391
BeimDrucken-	
Eigniseigenschaft	425
BeimFormatieren-	
Eigniseigenschaft	421
BeimKlicken-	
Eigniseigenschaft	24, 383
BeiNichtInListe-	
Eigniseigenschaft	404
benannte Parameter	39
Benutzer	605
einer Gruppe zuordnen	613
löschen	612
neu anlegen	609
Benutzerdatenbank	231
benutzerdefinierte	
Datentypen	129, 709
benutzerdefinierte Hilfe	599
Benutzergruppen	605, 613
Benutzer zuordnen	613
löschen	612
Benutzernamen,	
Namenskonventionen	608
Bereiche	201
Berichte	
öffnen	418
Bezugnahme	198, 203
Circle-Methode	432
CurrentX-Eigenschaft	435
CurrentY-Eigenschaft	435
Datensätze hervorheben	422
Datensatzvorschub	
unterdrücken	425





Datensatzwechsel	424
DrawStyle-Eigenschaft	430
Drucken einzelner	
Datensätze	420
entwerfen	570
erzeugen	521
Grafik-Cursor positionieren	435
Koordinatensystem	
anpassen	439
Kreise zeichnen	432
Layout beeinflussen	421
Line-Methode	428
Linienart auswählen	430
mehrere Datensatzkopien	
drucken	424
NextRecord-Eigenschaft	424
Print-Ereignis	425
Print-Methode	436
PrintCount-Parameter	425
PSet-Methode	434
Punkte zeichnen	432
Rechnen in	425
ScaleHeight-Eigenschaft	440
ScaleMode-Eigenschaft	439
ScaleWidth-Eigenschaft	440
Schließen	419
Schriftart-Eigenschaft	437
Schriftattribute auswählen	437
Schriftgröße-Eigenschaft	437
Seitenfuß-Eigenschaft	427
selbstdefinierte	
Koordinatensysteme	440
Textausgabe	436
Zeichnen auf	428
Berichtssteuerelement	
erzeugen	526
Bezeichnungsfelder	413
Beziehungen	
1\	
1	547
n	545
n:auf sich selbst	
erzeugen	558
definieren	489
löschen	492
n\	
m	547
m:auf sich selbst	
erzeugen	558
m:erzeugen	557
Beziehungstypen	546
Bezugnahme auf Formulare und	
Berichte	198, 203

Bezugnahme auf	
Steuerelemente	204
Bezugnahme auf	
Unterformulare	205
Bibliotheksdatenbanken	179
Bild-Steuerelemente	413
Bildlaufleisten	727
BOF-Eigenschaft	238, 271
Bookmark-Eigenschaft	252
Bookmarkable-Eigenschaft	252

C

Call by reference	145
Call by value	145
Call-Anweisung	72
Caption-Eigenschaft	510
Change-Ereignis	374
Chr-Funktion	97
Circle-Methode	432
Client	630
Clone-Eigenschaft	254
Close-Ereignis	317
Close-Methode	236, 315
Objekttypen	315
Clustered-Eigenschaft	488
Code-Editor	32
Collection-Objekt	458
Column-Eigenschaft	392
Command-Objekt	266
CompactDatabase-Funktion	593
ConflictTable-Eigenschaft	570
Connect-Eigenschaft	475
Connection-Objekt	265
Constraint-Klausel	500
Container	616
Container-Objekte	616
Controls-Auflistung	204
Create Index-Anweisung	504
Create Table-Anweisung	497
Create-Methode	455
CreateControl-Funktion	512
CreateDatabase-Methode	471
CreateForm-Funktion	510
CreateGroupLevel-Funktion	522
CreateObject-Funktion	655
CreateQueryDef-Methode	493
CreateReport-Funktion	521
CreateReportControl-Funktion	526
Currency-Datentyp	113
Current-Ereignisprozedur	330

CurrentData-Objekt	208
CurrentDb-Funktion	230
CurrentProject	207
CurrentUser-Funktion	607
CurrentX-Eigenschaft	435
CurrentY-Eigenschaft	435
CVDate-Funktion	123

D

DAO	54
Database-Objekte	230, 231, 473
DateAdd-Funktion	103
DateDiff-Funktion	104
Dateiliste über eine Funktion	
erstellen	399
Daten-Tipps	170
Datenbankdatei ermitteln	231
Datenbanken	
aufteilen	562
Eigentümer ändern	622
einbinden	179
erzeugen	471
komprimieren	592
öffnen	231
reparieren	592
replizieren	562
schließen	232
Schließen verhindern	317
schützen	616
sichern	622
verwalten	592
Datenbankmodell	
Objektdatenbank -	
Tabellendatenbank	560
Standarddatenbank	559
Datenbankverweise	179
Dateneingabe, Kontrolle der	343
Dateneingabe überwachen	373
Datenfelder	130
Datenfelder, Zugriff auf	241
Datenpflege	592
während der Laufzeit	593
Datensatzanzahl	239
Datensatzanzahl ermitteln	228
Datensatzgruppe, filtern	251
Datensatzvorschub	
unterdrücken	425
Datensatzwechsel	424
Datensatzzeiger verschieben	237



- Datensätze**
 aktualisieren 339
 drucken 420
 filtern per SQL 293
 gruppieren per SQL 293
 hervorheben 422
 hinzufügen 259
 löschen 259, 334
 löschen, per SQL 337
 Neueingabe 332
 Positionieren auf 320
 sortieren per SQL 295
 suchen 244, 318
- Datentypen** 129
 umwandeln 120
 benutzerdefiniert 129
- Datenzugriff mit Access Basic** 225
- DatePart-Funktion** 106
- Datumsformat in Access-SQL** 305
- Datumsausdrücke** 100, 116
- DAvg-Funktion** 228
- DBEngine-Objekt** 231
- DB_OPEN_DYNASET** 235
- DB_OPEN_SNAPSHOT** 235
- DB_OPEN_TABLE** 235
- DCount-Funktion** 228
- DDE** 673
- DDE-Server** 681
- DDEExecute-Anweisung** 676
- DDEInitiate-Funktion** 674
- DDEPoke-Anweisung** 676
- DDERequest-Funktion** 676
- DDETerminate-Anweisung** 676
- DDETerminateAll-Anweisung** 676
- Deactivate-Ereignis** 366
- Debug-Objekt** 171
- Debug.Print-Anweisung** 171
- Deklaration** 109, 697
- Deklarationsbereich** 68
- Delete-Anweisung** 301
- Delete-Ereignis** 340
- Delete-Methode** 259, 338, 484, 492, 499, 612
- DeleteControl-Funktion** 521
- DeleteReportControl-Funktion** 526, 537
- Design-Master-Replik** 563
- DesignMasterID-Eigenschaft** 563
- Dialog zum Löschen von Kunden** 361
- Dialogfenster** 353
 öffnen 354
 Übergabe mehrerer Informationen 359
 Informationen übergeben 358
- Dir-Funktion** 395
- Direktfenster** 169
 Öffnen 73, 169
- Disallow Null-Prädikat** 504
- Distinct-Prädikat** 288
- Distinctrow-Prädikat** 288
- DLookup-Funktion** 226
- DMax-Funktion** 228
- DMin-Funktion** 228
- Do...Loop** 156
- DoCmd-Objekt** 198
- Document-Objekte** 616, 617
- Dokumente** 616
- Domänenfunktionen** 226
- Double-Datentyp** 113
- DrawStyle-Eigenschaft** 430
- DrawWidth-Eigenschaft** 430
- Dreifacher-Status-Eigenschaft** 382
- Drop Table-Anweisung** 505
- Drop-Anweisung** 505
- Drop-Prädikat** 502
- Drucken bestimmter Datensätze** 402
- Drucken einzelner Datensätze** 420
- DStDev-Funktion** 228
- DStDevP-Funktion** 228
- DSum-Funktion** 228
- DVar-Funktion** 228
- dynamische Arrays** 133
- Dynaset** 233
- Eingabehilfen** 60
- Eingeben einer Prozedur** 35
- Einrichten der Benutzeroberfläche** 572
- Einzelschrittausführung eines Programms** 168
- Elemente automatisch auflisten** 61
- Else-Anweisung** 51
- Endlosschleifen** 157
- Enter-Ereignis** 380
- Entitätstypen** 546
- Entity Relationship-Diagramme** 546
- Entscheidungsstrukturen, Select...Case** 152
- EOF-Eigenschaft** 238, 271
- ER-Diagramme** 546
- Ereigniseigenschaften, Übersicht** 26
- ereignisorientierte Programmierung** 185
- Ereignisprozedur** 37
- Ereignisse** 23, 185
- Ereignisse von Kontrollkästchen** 383
- Err-Objekt** 163
- Ersetzen von Programmcode** 65
- Erstellen einer Prozedur** 35, 70
- Erzeugen einer 1:n-Beziehung auf sich selbst** 558
- Erzeugen einer n:m-Beziehung** 557
- m-Beziehung auf sich selbst** 558
- Excel per DDE steuern** 675
- Excel per OLE-Automation steuern** 656
- Execute-Anweisung** 235, 337
- Execute-Methode** 509
- Exit-Ereignis** 380
- Exp-Funktion** 83
- explizite Deklaration** 112
- Exponentialzahlen** 80
- Export von Programmcode** 65

E

- Edit-Methode** 257
- Editieren von Makros** 20
- Eigenschaften** 208, 444
- DesignMasterID** 563
- Hyperlink** 411
- HyperlinkAddress** 414
- HyperlinkSubAddress** 415
- PageIndex** 408
- ReplicaID** 563
- von Tabellenfeldern** 478
- Einbinden einer Tabelle** 475
- Eingabe von Programmcode** 58

F

- F1-Taste abfragen** 378
- False** 86
- Farbcodierungen** 61
- Fehler:**
 abfangen 161
 Kompilierungsfehler 165
 Laufzeitfehler 165
 logische Fehler 165



G

H

Haltepunkte	167
Handle	700
Hauptfenster	712
HAVING-Klausel	294
Help-Compiler	600
Hexadezimalzahlen	80, 81
Hilfdatei	697
Hilfdatei-Eigenschaft	601
Hilfekontext-Eigenschaft	601
Hinzufügen eines Datensatzes	257
Hyperlink-Eigenschaft	411
Hyperlink-Felddatentyp	411
HyperlinkAddress-Eigenschaft	414
HyperlinkPart-Funktion	412
Hyperlinks	411
HyperlinkSubAddress-Eigenschaft	415

I

If-Anweisung	50, 149
If...Then...Else	149
Ignore Null-Prädikat	504
implizite Deklaration	112
Import von Programmcode	65
In-Klausel	291, 304
In-Place-Activation	635
Index-Objekt	485
Indizes	
auswählen	247
definieren	500
erzeugen	485
hinzufügen	485
löschen	488
per Abfrage erzeugen	504
per Abfrage löschen	502, 506
zuweisen	485
Informationen an Dialogfenster	
übergeben	358
Initialisierung	454, 455
Initialisierung von Variablen	109
Initialize-Ereignis	454
Inner Join-Prädikat	290
Insert-Anweisung	302
Instanzen eines Formulars	461
Int-Funktion	83
Integer-Datentyp	113
Into-Prädikat	300

ISAM-Formate	292
IsDate-Funktion	123
IsIconic()	711
IsZoomed()	711
ItemData-Methode	401
ItemsSelected-Auflistung	401

J

JET-Engine	231

K

Kapselung der Daten	445
KeepLocal-Eigenschaft	563
KeepTogether-Eigenschaft	525
Kennzeichenleiste	167
Kernel32.exe	691
KeyCode-Parameter	378
KeyDown-Ereignis	378
KeyPress-Ereignis	375
KeyUp-Ereignis	378
Kindfenster	718
Klassen	444, 459
Klassenmodule	68, 445
Klicken-Ereignis	309
Klonen einer Datensatzgruppe	254
Kombinationsfelder	389, 403
NewData-Parameter	405
Response-Parameter	405
Kompilerdirektiven	176
Kompilieren	175
Kompilierungsfehler	165
Komprimieren von	
Datenbanken	592
Konflikte bei Replikation	569
Konstante	707
Konstanten	48, 136
Kontextmenüs	587
kontextsensitive Menüzeilen	585
kontextsensitive Symbolleisten	585
Kontrolle der Dateneingabe	343
Kontrollkästchen	381
Click-Ereignis	383
Ereignisse	383
in Optionsgruppen	387
Kontrollstrukturen	149
konzeptuelles Datenschema	546
Koordinatensystem anpassen	439

Koordinatensysteme selbst	
definieren	440
Kreuztabellenabfragen	
definieren	303

L

Laufzeitfehler	165
Laufzeitfehler abfangen	161
LBound-Funktion	134
Lebensdauer von Variablen	125
Leer	119
Left Join-Prädikat	290
Lesezeichen	175, 252
Like-Operator	92
Line-Methode	428
Linien	
Füllfarbe beeinflussen	431
FillStyle-Eigenschaft	432
Linienart auswählen	430
Listenbreite anpassen	364
Listeneinträge auswählen	390
Listenfelder	389
Herkunftstyp	394
mit Mehrfachauswahl	401
mit Wertelisten füllen	394
mittels einer Funktion	
füllen	397
Listing drucken	66
Load-Ereignis	315
LockEdits-Eigenschaft	258
Log-Funktion	83
logische Ausdrücke	86
logische Fehler	165
Lokal-Fenster	171
Lokalität	128
Long-Datentyp	113
LostFocus-Ereignis	380
Löschabfrage	337
Löschabfragen definieren	301
Löschen	
von Beziehungen	492
von Datensätzen	334
von Feldern	484
von Tabellen	484



M

Maßeinheiten	716
Makereplica-Methode	567
Makro-Editor	25
Makrofenster	3
Makrogruppe	18
Makroprogrammierung	1, 443
Makros	1, 443
Aktionen verschieben	20
Aktionsargumente	6
anlegen	3
ausführen	5
bearbeiten	3
Bedingung-Spalte	15
Bedingungen	12, 14
Beim	
Klicken-Ereigniseigenschaft	24
Editieren	20
Einzelschrittmodus	22
Ereignisse	23
Fehlermeldungen	23
Fehlersuche	14, 22
Formular öffnen	8
Fortsetzungszeichen	15, 16
Gruppenmakros	17
If...Then...Else	16
Leerzeilen	16
Makro-Editor	25
Meldung anzeigen	4
ÖffnenFormular-Aktion	8
speichern	5
starten	5
Untermakro ausführen	18
Vergleichsoperatoren	13
Zeichenketten	14
Zwischenablage	21
mathematische Funktionen	82
Me-Eigenschaft	200
Mehrbenutzerumgebung	258
mehrdimensionale Arrays	132
Mehrfachauswahl bei	
Listenfeldern	401
Mehrfachauswahl-Eigenschaft	401
Mehrfelderindex	
definieren	488, 500
mehrmaliges Öffnen eines	
Formulars	461
mehrzeilige Texte	97
Meldung-Aktion	3
Member-Variable	446, 448, 449, 451
Menübefehle ausführen	335

Menüleisten	572, 583
Methoden	208, 211, 452
Module	67, 531
Alles markieren	60
Drag & Drop	59
Einfügemarke	58
ersetzen	65
erzeugen	535
Fenster teilen	57
Import und Export	65
Listing drucken	66
Programmzeilen einrücken	60
Prozedur ausführen	72
Prozedur erstellen	70
suchen	63
Syntaxkontrolle	60
Text markieren	60
Zwischenablage	59
Modulfenster	35, 55, 58
Modulnamen	68
MousePointer-Eigenschaft	206
Move-Methoden	237, 241, 270
MoveSize-Methode	361
MsgBox-Anweisung	38

N

n\	
m-Beziehung	547
erzeugen	557
m-Beziehung auf sich selbst	558
NachAktualisierung-Ereignis	343
NachEingabe-Ereignis	334
Name-Eigenschaft	606
Namenskonflikte bei Prozeduren	76
Namenskonflikte bei Variablen	128
Namenskonflikte zwischen	
referenzierten Datenbanken	183
Namenskonventionen	548
Namenskonventionen für	
Datenbankobjekte	548
Steuerelemente	551
Variablen in Access-Basic	551
Namenskonventionen von	
Prozeduren	69
Navigation kontrollieren	330
Navigation in Datensatzgruppen	
	237
Navigationsschaltflächen	327
Navigator	363
Navigator testen	392
Navigieren in Formularen	327
Netzwerk	258
neue Beziehung erstellen	489
neue Tabelle anlegen	474
Neueingabe von Datensätzen	332
neuen Index anlegen	485
neues Feld anlegen	478
New	
Field	478
Index	485
Relation	489
Tabledef	474
New-Schlüsselwort	461, 661
NewData-Parameter	405
NextRecord-Eigenschaft	424
NoMatch-Eigenschaft	246
Not-Operator	86
NULL	119
numerische Ausdrücke	79
numerische Datentypen	113

O

Object-Eigenschaft	664
Objektdatentypen	215, 217, 655
Objekte	194
Debug	171
Objektfelder	636
Objektgruppen	198, 203
Objekthierarchie	229
Objekthierarchie von Access	195
Objektinstanz	194
Objektkatalog	220
Objektklassen	194, 444, 459
objektorientierte	
Programmierung	443
Objektvariablen	213
Objektverben ermitteln	643
Objektverweise in Abfragen	
verwenden	299
OCX-Controls	667, 719
ODBC	262
öffentliche Variablen	126
Öffnen des Direktfensters	73, 169
Öffnen einer Datenbank	231
Öffnen mehrerer	
Formularinstanzen	461
Öffnen von Dialogfenstern	354
ÖffnenFormular-Aktion	8, 44, 198
Oktalzahlen	80, 81
OLD-DB	263



743

ReplicationConflictFunction-Eigenschaft	570	Schließen von Formularen	315	Starten eines Makros beim Öffnen einer Datenbank	190
Replikate, Synchronisieren von	564	Schriftart-Eigenschaft	437	Startmakro	190
Replikatgruppe erzeugen	563	Schriftattribute	437	statische Variablen	125
Replikation		Schriftgröße-Eigenschaft	437	Steuerelement eines Berichts erzeugen	526
Anlegen einer Replikatgruppe	563	schrittweise Programmausführung	168	löschen	526
GUID	563	Schützen einer Datenbank	622	Steuerelemente	
Identifikation von Objekten	563	Screen-Objekt	206	aktivierte farbig hinterlegen	381
KeepLocal-Eigenschaft	563	Section-Eigenschaft	201	Anzahl auf einem Formular	218
Konflikte	569	Seitenfuß-Eigenschaft	427	erzeugen	512
Makereplica-Methode mit VBA	567	Seitenumbrüche	406	löschen	521
neue Eigenschaften	563	selbstdefinierte Symbolleisten	576	programmieren	371
neue Felder und Tabellen	564	Select Into-Anweisung	300	Register	408
Objekte in der Design-Master Datenbank	564	Select-Anweisung	286	Seitenumbrüche	406
Objekte in einem Replikat	564	Select...Case	152	Steuerelementinhalt, Zugriff auf	209
Partielle	565	SendKeys-Anweisung	517	Steuerelementinhalt-Eigenschaft	372
Replicable-Eigenschaft	563	serielles Datumsformat	100	Str-Funktion	120
Sicherheitskopien erzeugen	566	Server	630	String-Datentyp	109
Synchronisieren von Replikaten	564	Set-Prädikat	303	Stringparameter	702
Synchronize-Methode von Datenbanken	562	Set-Anweisung	213	Strings fester Länge	705
Reports-Auflistung	203	SetFocus-Methode	312	Strukturen	709
Requery-Methode	211, 339	SetWindowPos	712	Sub-Prozeduren	70
Required-Eigenschaft	485	SetWindowPos()	700	SubClass-Control	719
Resize-Ereignisprozedur	364	Sgn-Funktion	83	Suchen nach Datensätzen	244, 318
Response-Parameter	341, 405	Shift-Parameter	379	Suchen nach Programmcode	63
Resume-Anweisung	162	ShowAllRecords-Methode	312	Suchen nach weiteren Datensätzen	322
Richtlinien zum Berichtdesign	571	Sicherheitssystem	602, 604	Suchtext eingeben	322
Richtlinien zum Formulardesign	570	Sichern einer Datenbank	622	Summe berechnen	228
Right Join-Prädikat	291	Single-Datentyp	113	Symbol	711
Rnd-Funktion	83	Sinus-Funktion	82	Symbolleisten	
RollUp-Formulare	366	Snapshot	233	Anlegen	576
RunCommand-Methode	335, 366	Sort-Eigenschaft	250	Befehl löschen	577
RunSql-Anweisung	508	Sortieren	277	Befehl verschieben	577
		Sortieren in Formularen	344	Befehle hinzufügen	576
		Sortieren von Datensatzgruppen	249	Befehle selbst definieren	581
		Sortierreihenfolge bei Gruppierungen	525	Einbinden	585
		SortOrder-Eigenschaft	525	Gruppieren von Befehlen	578
		SourceTableName-Eigenschaft	475	Kontextmenüs	587
		Spalten, Zugreifen auf	392	Kontextmenüs verwenden	587
		Speichern von Filtern	350	kontextsensitiv	585
		SQL-Abfragen ausführen	235	Makro aufrufen	582
		Sql-Eigenschaft	493	Menüleisten anlegen	583
		Sql-Funktion	83	Symbole hinzufügen	576
		Standardereigniseigenschaften	187	Untermenüs	584
		Standardereignisse	187	VBA-Prozedur aufrufen	582
		Standardmodule	67	verwenden	585
		Standardparameter von Aktionen	48	Synchronisieren von Replikaten	564
		Starten einer Prozedur	72	Synchronize-Methode	568
		Starten eines Makros	5	Syntaxkontrolle	60

S

ScaleHeight-Eigenschaft	440
ScaleMode-Eigenschaft	439
ScaleWidth-Eigenschaft	440
Schleifenstrukturen	149
Do...Loop	156
For...Next	154
While...Wend	159
Schließen der aktuellen Datenbank verhindern	317



T

Tabellen	
einbinden	475
erzeugen	473, 474
per Abfrage erzeugen	497
per Abfrage löschen	505
per Abfrage verändern	502
Tabellenerstellungsabfrage	497
Tabellenerstellungsabfragen	
definieren	300
Tabellenfelder	
anlegen	478
hinzufügen	479
Table-Eigenschaft	490
TableDef-Objekt	474
Tastatureingaben ignorieren	376
Tastatureingaben umwandeln	377
Tastaturereignisse	375
temporärer String	706
Terminate-Ereignis	454
Terminierung	454
Terminkalender	725
Testen einer Prozedur	73
Textausgabe	436
Texte in VBA angeben	89
Textfelder	
Überwachen der	
Dateneingabe	373
Change-Ereignis	374
KeyPress-Ereignis	375
Kontrolle der Eingabe	374
Kontrolle des Fokus	380
programmieren	372
Tastaturereignisse	375
Titelzeile	713
Top-Prädikat	288
Transform-Anweisung	303
True	86
Twips	362
TwipsPixel	717
Typ-Umwandlung	115, 120, 138
typfreie Parameter	706
typsichere API-Funktionen	707

U

UAE	704
UBound-Funktion	134
Übergabe von Variablen an	
Prozeduren	137

Übersicht aller	
Ereigniseigenschaften	26
Übersicht der Abfragetypen	304
Übersichts-Manager	589
Überwachen von Variablen	173
Umschaltflächen	381, 384
in Optionsgruppen	387
ungebundene Textfelder	372
Unique-Prädikat	500, 504
Unique-Eigenschaft	485
Unload-Ereignis	317
Unterbrechungsmodus	166
Unterformulare,	
Bezugnahme auf	205
Untermenüs	584
Update-Anweisung	303
Update-Ereignis	645
User-Objekte	606
User32.exe	691
UserName-Eigenschaft	618

V

Val-Funktion	121
Value-Eigenschaften	209
Variablen	108
öffentlich	126
global	126
privat	126
Variant-Arrays	135
Variant-Datentyp	116, 135
VarTyp-Funktion	117
VBA	54
Vergleichsoperatoren	87
Verknüpfen von Objekten	633
Verknüpfen von Zeichenketten	90
Verknüpfen zweier	
Zeichenketten	90
Verknüpfungsfelder festlegen	491
Verknüpfungsoperatoren	81
Verknüpfungstabellen	
festlegen	490
Verschlüsseln einer Datenbank	597
Versionsnummer einer	
Datenbank	597
Verwalten von Datenbanken	592
Verwalten von Symbolleisten	573
Verweise zu Datenbanken	
herstellen	179
Visual Basic for Applications	54
Vollbild	711

Vor-Ort-Aktivierung	635
VorAktualisierung-Ereignis	343
VorEingabe-Ereignis	333

W

Weitersuchen	322
Werte	78
Wertparameter	701
Where-Klausel	293
While...Wend	159
Währungen in VBA speichern	113
Win16	178
Win32	178
Win32API.txt	697
Windows-API	691
With-Anweisung	212
With-Klausel	504
Word per OLE-Automation	
steuern	659
Workspace-Objekte	231, 606
WS_BORDER	715
Wurzelfunktion	83
WWW-Adressen	411

Z

Zahlen	78
Zeichenketten	89, 109
verknüpfen	90
Zeichenkettenfunktionen	94
Zeichnen auf Berichten	428
Zeichnen in Fenstern	724
Zeilensprungmarken	159
Zeitangaben	101
Zeitausdrücke	100, 116
Zeitgeberintervall-Eigenschaft	367
Zeitgeberprogrammierung	367
Zeitspannen berechnen	103
Zufallszahlen	83
Zugriff auf eine referenzierte	
Datenbank	182
Zugriff auf externe Datenbanken	
per SQL	291
Zugriff auf fremde Datenbanken	483
Zugriff auf Module	531
Zugriff auf Spalten	392
Zugriffsberechtigungen erteilen	618





Zugriffsrechte kombinieren	604 619	Zuordnung von Benutzern und Gruppen	613	Zusatzsteuerelemente Zuweisen von Indizes	719 485

