

Für die
Versionen
2010 bis
2016

walter DOBERENZ
thomas GEWINNUS



ACCESS

PROGRAMMIEREN

GRUNDLAGEN UND PRAXISWISSEN

// VBA, Makros, Formulare und Berichte

// Datenbankprogrammierung mit DAO, ADO und ADOX

// Zusammenarbeit mit SQL Server, Azure SQL und SQLite

// Zahlreiche Praxisbeispiele

HANSER



Im Internet: Alle Beispieldaten aus
dem Buch zum Download

Doberenz/Gewinnus

Access programmieren

Bleiben Sie auf dem Laufenden!



Unser **Computerbuch-Newsletter** informiert Sie monatlich über neue Bücher und Termine. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter



www.hanser-fachbuch.de/newsletter



Hanser Update ist der IT-Blog des Hanser Verlags mit Beiträgen und Praxistipps von unseren Autoren rund um die Themen Online Marketing, Webentwicklung, Programmierung, Softwareentwicklung sowie IT- und Projektmanagement. Lesen Sie mit und abonnieren Sie unsere News unter



www.hanser-fachbuch.de/update



Walter Doberenz
Thomas Gewinnus

Access programmieren

Grundlagen und
Praxiswissen

Für die Versionen 2010, 2013 und 2016

HANSER

Die Autoren:

Professor Dr.-Ing. habil. Walter Doberenz, Wintersdorf

Dipl.-Ing. Thomas Gewinnus, Frankfurt/Oder

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht. Ebenso übernehmen Autoren und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.



Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über [<http://dnb.d-nb.de>](http://dnb.d-nb.de) abrufbar.

Dieses Werk ist urheberrechtlich geschützt.

Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung des Buches, oder Teilen daraus, sind vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2016 Carl Hanser Verlag München, www.hanser-fachbuch.de

Lektorat: Sylvia Hasselbach

Herstellung: Irene Weilhart

Satz: Ingenieurbüro Gewinnus

Sprachlektorat: Walter Doberenz

Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München

Umschlagrealisation: Stephan Rönigk

Druck und Bindung: Kösel, Krugzell

Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702

Printed in Germany

Print-ISBN: 978-3-446-45027-1

E-Book-ISBN: 978-3-446-45059-2

Inhaltsverzeichnis

Vorwort	27
 Teil I: Grundlagen	
1 Einführung	33
1.1 VBA-Programmierung in Access	33
1.1.1 Visual Basic versus VBA	33
1.1.2 Objekt- und ereignisorientierte Programmierung	34
1.1.3 VBA- oder Makro-Programmierung?	35
1.1.4 Die VBA-Entwicklungsumgebung	36
1.1.5 Formularentwurf	36
1.1.6 Code-Fenster und Symbolleiste	37
1.1.7 Das Eigenschaftenfenster	38
1.1.8 Der Projekt-Explorer	39
1.1.9 Das Code-Fenster	39
1.1.10 Arbeiten mit der Hilfe	42
1.2 Sicherheitseinstellungen	43
1.2.1 Zur Geschichte der Access-Sicherheit	43
1.2.2 Eine nicht vertrauenswürdige Datenbank öffnen	44
1.2.3 Das Sicherheitscenter	46
1.2.4 Definition vertrauenswürdiger Speicherorte	49
1.3 Einführungsbeispiele	50
1.3.1 Erstellen der Testdatenbank	51
1.3.2 Konventionelle Programmierung	52
1.3.3 Programmieren mit VBA	56
1.3.4 Automatische Makrokonvertierung	61
1.3.5 Programmieren mit Datenmakros	62
1.4 Highlights und Features von Access 2016	64
1.4.1 Zur Geschichte der Vorgängerversionen	64
1.4.2 Microsoft Access 2016 – viel Lärm um nichts?	67
1.4.3 Der inoffizielle Access-Friedhof (Access 2013/2016)	68

1.5	Übersichten und Ergänzungen	68
1.5.1	Deutsche und englische Bezeichner	68
1.5.2	DoCmd-Objekt	70
2	Programmieren mit VBA	73
2.1	Datentypen, Variablen und Konstanten	73
2.1.1	Übersicht	73
2.1.2	Variablendeklaration	74
2.1.3	Konstantendeklaration	79
2.1.4	Gültigkeitsbereiche	80
2.2	Einzelheiten zu den Datentypen	83
2.2.1	Single- und Double-Datentypen	83
2.2.2	Integer-, Long- und Boolean-Datentypen	83
2.2.3	Date-Datentyp	84
2.2.4	Currency-Datentyp	86
2.2.5	String-Datentyp	87
2.2.6	Variant-Datentyp	89
2.3	Datenfelder (Arrays)	92
2.3.1	Statische Arrays	92
2.3.2	Dynamische Arrays	94
2.4	Benutzerdefinierte Datentypen	96
2.4.1	Type-Anweisung	96
2.4.2	With-Anweisung	97
2.4.3	Strings innerhalb Type	97
2.4.4	Enumerationen	98
2.4.5	Arrays in benutzerdefinierten Typen	99
2.5	Operatoren	100
2.5.1	Arithmetische Operatoren	101
2.5.2	Logische Operatoren	103
2.5.3	Vergleichsoperatoren	104
2.6	Kontrollstrukturen	105
2.6.1	Bedingte Verzweigungen	105
2.6.2	Schleifenanweisungen	107
2.6.3	GoTo und GoSub	109
2.7	Zeichenkettenfunktionen	110
2.7.1	Stringverarbeitung	110
2.7.2	Format-Funktion	112
2.8	Vordefinierte Funktionen	116
2.8.1	Mathematische Funktionen	116
2.8.2	Finanzmathematische Funktionen	119

2.8.3	Datums-/Zeitfunktionen	120
2.9	Benutzerdefinierte Funktionen/Prozeduren	123
2.9.1	Funktion	123
2.9.2	Prozedur	124
2.9.3	Parameterübergabe ByRef oder ByVal	124
2.9.4	Optionale Argumente	125
2.9.5	Benannte Argumente	126
2.9.6	Parameter-Arrays	126
2.9.7	Dynamische Arrays als Argumente	127
2.9.8	Rückgabe von Arrays	127
2.9.9	Private-, Public- und Static-Deklarationen	128
2.10	Fehlersuche	130
2.10.1	Direktfenster	131
2.10.2	Verwendung des Debug-Objekts	131
2.10.3	Arbeiten mit dem Lokal-Fenster	132
2.10.4	Überwachungs-Fenster	134
2.10.5	Noch mehr Debugging	135
2.11	Fehlerbehandlung	139
2.11.1	Anweisungen zum Error-Handling	139
2.11.2	Beispiele zum Error-Handling	140
2.11.3	Fehlerbehandlung per Ereignis	142
2.11.4	Fehlerbehandlung komplett deaktivieren	143
2.12	Standarddialogfelder	143
2.12.1	Einfache MsgBox-Anweisung	144
2.12.2	Ausführliche MsgBox-Anweisung	145
2.12.3	Rückgabewerte der MsgBox-Funktion	145
2.12.4	Abfrage von Werten mit der InputBox-Funktion	147
2.13	Übersichten und Ergänzungen	148
2.13.1	Datumskonstanten	148
2.13.2	Rückgabewerte der VarType-Funktion	148
2.14	Praxisbeispiele	149
2.14.1	In einem Textfeld suchen	149
2.14.2	Zeitangaben runden	150
2.14.3	Das Wochenende feststellen	152
2.14.4	Mit dynamischen Arrays rechnen	153
2.14.5	Arbeiten mit dem Debugger	157
3	Makros – eine Einführung	163
3.1	Klassische Makros	163
3.1.1	Entwurfsoberfläche	163

3.1.2	Eigenständige Makros	164
3.1.3	Eingebettete Makros	167
3.1.4	Das AutoKeys-Makro	171
3.1.5	Das AutoExec-Makro	173
3.1.6	Potenziell gefährliche Makroaktionen	173
3.2	Datenmakros	174
3.2.1	Einsatzmöglichkeiten	175
3.2.2	Funktionsprinzip	175
3.2.3	Erzeugen von Datenmakros	176
3.2.4	Datenmakros umbenennen, löschen und ändern	177
3.2.5	USysApplicationLog	177
3.2.6	Aktionen in Datenmakros	178
3.2.7	Auswahl des richtigen Tabellenereignisses	179
3.3	Praxisbeispiele	180
3.3.1	Eingabe-Formular mit neuem Datensatz öffnen	181
3.3.2	Einen Datensatznavigator selbst bauen	182
3.3.3	Ein ereignisgesteuertes Datenmakro erstellen	184
3.3.4	Arbeiten mit einem benannten Datenmakro	189
3.3.5	Per VBA auf ein benanntes Datenmakro zugreifen	193
3.3.6	Änderungen von Tabelleninhalten protokollieren	194
4	Formulare und Steuerelemente	197
4.1	Allgemeines	197
4.1.1	Gruppen von Eigenschaften	198
4.1.2	Methoden	198
4.1.3	Gruppen von Ereignissen	198
4.2	Das Form-Objekt	199
4.2.1	Format-Eigenschaften	199
4.2.2	Daten-Eigenschaften	206
4.2.3	Weitere Eigenschaften	206
4.2.4	Fenster- und Fokus-Ereignisse	208
4.2.5	Tastatur- und Maus-Ereignisse	210
4.2.6	Daten- und Filter-Ereignisse	212
4.2.7	Weitere Ereignisse	214
4.2.8	Methoden	214
4.2.9	Unterformulare	217
4.3	Steuerelemente (Controls)	218
4.3.1	Allgemeines	218
4.3.2	Allgemeine Eigenschaften auf einen Blick	219
4.3.3	Allgemeine Ereignisse auf einen Blick	229

4.3.4	Methoden von Steuerelementen	230
4.3.5	Das Screen-Objekt	231
4.4	ActiveX-Steuerelemente	233
4.4.1	Vergleich mit den integrierten Steuerelementen	233
4.4.2	StatusBar als Beispiel	235
4.5	Praxisbeispiele	239
4.5.1	Das Textfeld programmieren	239
4.5.2	In ungebundene Textfelder ein- und ausgeben	241
4.5.3	Ein ungebundenes Kombinationsfeld füllen	242
4.5.4	Ein Unterformular programmieren	245
4.5.5	Das Register-Steuerelement kennen lernen	248
4.5.6	Die Statusleiste programmieren	252
4.5.7	Verwenden von Bild-Ressourcen	255
4.5.8	Programmieren des Navigationssteuerelements	257
5	Berichte	261
5.1	Allgemeines	261
5.1.1	Reportansichten	261
5.1.2	Die OpenReport-Methode	262
5.1.3	Parameterübergabe	263
5.2	Wichtige Berichtseigenschaften	263
5.2.1	Formateigenschaften	263
5.2.2	Dateneigenschaften	264
5.2.3	Grafikeigenschaften	264
5.2.4	Linien- und Stifteigenschaften	268
5.2.5	Schrifteigenschaften	269
5.2.6	Farb- und Mustereigenschaften	269
5.2.7	Sonstige Eigenschaften	271
5.3	Berichtsergebnisse	273
5.3.1	Allgemeine Ereignisse	273
5.3.2	Tastatur- und Mausereignisse	275
5.4	Berichtsmethoden	276
5.4.1	Grafikmethoden (Übersicht)	276
5.4.2	Scale	276
5.4.3	Line	277
5.4.4	PSet	278
5.4.5	Circle	279
5.4.6	Print	280
5.4.7	TextWidth und TextHeight	281
5.4.8	Sonstige Methoden	282

5.5	Weitere Features des Report-Objekts	282
5.5.1	Rich-Text-Felder drucken	282
5.5.2	Verlauf eines Memofeldes drucken	283
5.5.3	Eine Liste der Anlagen drucken	283
5.5.4	Berichte nachträglich filtern	286
5.5.5	Berichte als PDF-Datei exportieren	287
5.5.6	Berichte als RTF-Datei exportieren	288
5.6	Das Printer-Objekt	288
5.6.1	Wo finde ich das Printer-Objekt?	289
5.6.2	Die Printers-Collection	289
5.6.3	Auswahl eines Druckers	290
5.6.4	Speichern von Berichts-Optionen	292
5.6.5	Eigenschaften des Printers	293
5.7	Direkte Druckausgabe	294
5.8	Übersichten	294
5.8.1	DrawMode-Eigenschaft	294
5.8.2	Farbkonstanten	295
5.9	Praxisbeispiele	295
5.9.1	Aufruf eines Berichts mit Datenfilter	295
5.9.2	Im Report gruppieren und rechnen	299
5.9.3	Erstellen und Drucken eines Diagramms	303
5.9.4	Berichte in Formularen anzeigen	307
6	Programmieren mit Objekten	309
6.1	Objektvariablen	309
6.1.1	Objekttypen und Set-Anweisung	309
6.1.2	Object-Datentyp	311
6.1.3	Form- und Report-Objekt	312
6.1.4	Control-Objekt	313
6.2	Formular- und Berichtsmodule	317
6.2.1	Instanzen von Formularen und Berichten	317
6.2.2	Benutzerdefinierte Form-/Report-Objekte	319
6.2.3	Eigenständige Klassenmodule	320
6.3	Auflistungen	324
6.3.1	Forms/Reports	324
6.3.2	Controls	325
6.3.3	Collection-Objekt	327
6.3.4	Dictionary-Objekt	329
6.3.5	Property und Properties	329
6.3.6	Module-Objekt und Modules-Auflistung	330

6.3.7	Reference-Objekt und References-Auflistung	332
6.4	Die Access-Objekthierarchie	334
6.4.1	Der Objektkatalog	334
6.4.2	Das Application-Objekt allgemein	335
6.4.3	Eigenschaften und Methoden des Application-Objekts	338
6.4.4	Weitere wichtige Objekte	343
6.4.5	AccessObject	344
6.4.6	CurrentProject	345
6.4.7	CurrentData	347
6.5	Übersichten	347
6.5.1	Konstanten der ControlType-Eigenschaft	347
6.5.2	Rückgabewerte der CurrentObjectType-Funktion	348
6.6	Praxisbeispiele	348
6.6.1	Ein Steuerelemente-Array automatisch erstellen	348
6.6.2	Mit Formular-Instanzen arbeiten	352
6.6.3	Mit einer eigenständigen Klasse experimentieren	354
6.6.4	Auf Objekte in Auflistungen zugreifen	357
6.6.5	Properties-Auflistungen untersuchen	360

Teil II: Datenschnittstellen

7	DAO-Programmierung	365
7.1	Allgemeines	365
7.1.1	DBEngine	365
7.1.2	Workspace-Objekt	366
7.1.3	Database-Objekt	367
7.1.4	Recordset-Objekt	367
7.1.5	Verwendung der Datenbankobjekte	368
7.2	Grundlegende Arbeitstechniken	368
7.2.1	Arbeitsumgebung festlegen	369
7.2.2	Datenbank anlegen und öffnen	369
7.2.3	Tabellen/Indizes anlegen	373
7.2.4	Tabellen einbinden	378
7.2.5	Tabellen verknüpfen (Relationen)	379
7.2.6	Abfragen erstellen/ausführen	381
7.2.7	Öffnen von Tabellen/Abfragen	383
7.3	Arbeiten mit Recordsets	386
7.3.1	Eigenschaften und Methoden von Recordsets	386
7.3.2	Datensätze anzeigen	389
7.3.3	Datensätze hinzufügen/ändern	391

7.3.4	Datensätze löschen	393
7.3.5	Datensätze sortieren	395
7.3.6	Datensätze suchen	396
7.3.7	Datensätze filtern	397
7.3.8	DAO in gebundenen Formularen	398
7.3.9	Auf Anlage-Felder zugreifen	401
7.3.10	Auf mehrwertige Felder zugreifen	404
7.3.11	Verlaufsverfolgung eines Memo-Felds	405
7.4	Weitere Funktionen	406
7.4.1	Eigenschaften (Properties)	406
7.4.2	Transaktionen	408
7.5	Praxisbeispiele	409
7.5.1	Eine Tabelle anlegen	409
7.5.2	Navigieren mit DAO	412
7.5.3	Den Datensatzzeiger bewegen	415
7.5.4	In Recordsets suchen	419
7.5.5	Eine Datenbank analysieren	422
7.6	Komplexbeispiel: Telefonverzeichnis	425
7.6.1	Eingabemaske	425
7.6.2	Anforderungen	425
7.6.3	Programmierung	426
7.6.4	Test und Bemerkungen	435
8	ADO-Programmierung	437
8.1	Ein erster Blick auf ADO	437
8.1.1	Kleines Einführungsbeispiel	438
8.1.2	Zur Geschichte von ADO	439
8.1.3	Hinweise zu den ADO-Bibliotheken	440
8.1.4	ADO und OLE DB	441
8.1.1	ADO-Objektmodell	442
8.2	ADO-Grundoperationen	444
8.2.1	Beziehungen zwischen den Objekten	444
8.2.2	Die Verbindung zur Datenquelle	445
8.2.3	Aktionsabfragen mit dem Command-Objekt	449
8.2.4	Recordsets mit Daten füllen	451
8.3	Weitere Operationen mit Recordsets	456
8.3.1	Welche Recordset-Features werden unterstützt?	456
8.3.2	Editieren von Datensätzen	457
8.3.3	Hinzufügen von Datensätzen	458
8.3.4	Löschen von Datensätzen	458

8.3.5	Recordsets filtern	459
8.3.6	Ungebundene Recordsets	460
8.3.7	Recordsets abspeichern	461
8.3.8	Bewegen in Recordsets	461
8.3.9	Daten direkt einlesen	462
8.3.10	Sortieren	463
8.3.11	Suchen	464
8.3.12	Ereignisse auswerten	464
8.4	Zugriff auf ADO-Auflistungen	466
8.4.1	Allgemeine Features	466
8.4.2	Property und Properties	467
8.4.3	Field und Fields	468
8.4.4	Parameter und Parameters	469
8.4.5	Error und Errors	470
8.5	Übersichten	471
8.5.1	Connection-Objekt	471
8.5.2	Command-Objekt	472
8.5.3	Recordset-Objekt	472
8.6	Praxisbeispiele	474
8.6.1	Mit ADO auf eine Access-Datenbank zugreifen	474
8.6.2	Ein ADO-Datenklassenmodul verwenden	476
8.6.3	Ein intelligentes ADO-Frontend entwickeln	479
9	Datenbankverwaltung	485
9.1	Datenbankverwaltung mit ADOX	485
9.1.1	Datenbanken erstellen	487
9.1.2	Tabellendefinition	489
9.1.3	Indexdefinition	493
9.1.4	Erstellen von Prozeduren und Sichten	495
9.1.5	Tabellen verknüpfen (Relationen)	496
9.2	Erstellen spezieller Feldtypen	497
9.2.1	Automatische Zufallswerte (GUID)	497
9.2.2	Memofeld mit Archiv-Funktion (Nur anfügen)	499
9.2.3	Anlage-Feld	501
9.2.4	Rich-Text-Feld	502
9.2.5	Multivalue-Feld (MVF)	503
9.2.6	Berechnete Spalten	508
9.2.7	Beschreibung von Datenbankfeldern setzen	510
9.3	Zugriffsschutz in Access-Datenbanken	512
9.3.1	Grundlagen	512

9.3.2	Sichern auf Datenbankebene (DAO)	514
9.3.3	Sichern auf Datenbankebene (ADO/ADOX)	515
9.3.4	Erstellen neuer Benutzer und Gruppen (DAO)	515
9.3.5	Vergabe von Rechten (DAO)	517
9.3.6	Komplettbeispiel: Nutzerbasierte Sicherheit	519
9.3.7	Erstellen neuer Benutzer und Gruppen (ADOX)	524
9.3.8	Vergabe von Rechten (ADOX)	525
9.3.9	Verschlüsseln von Datenbanken	527
9.4	Multiusernzugriff	530
9.4.1	Verwenden der DAO	531
9.4.2	Verwenden der ADO	534
9.5	ODBC-Verbindungen	535
9.5.1	Ein Blick auf den ODBC-Datenquellen-Administrator	535
9.5.2	Erstellen einer ODBC-Verbindung (DAO)	537
9.5.3	Öffnen einer ODBC-Verbindung (DAO)	538
9.5.4	Öffnen einer ODBC-Verbindung (ADO)	540
9.5.5	Konfigurieren von ODBC-Verbindungen	541
9.6	Zugriff auf Fremdformate	541
9.6.1	dBASE III/IV- und FoxPro-Datenbanken	542
9.6.2	Textdateien (TXT/ASC/CSV)	546
9.7	Einbinden externer Tabellen	550
9.7.1	Verwenden der DAO	550
9.7.2	Verwenden der ADOX	552
9.8	Exportieren von Daten	554
9.8.1	TransferDatabase-Methode	554
9.8.2	Exportieren mit SQL-Anweisungen	555
9.9	Replizieren von Datenbanken	555
9.10	Optimierung	556
9.10.1	Indizes	556
9.10.2	Abfrage-Optimierung	556
9.10.3	Weitere Möglichkeiten	557
9.10.4	ADO/DAO/ODBC – Was ist schneller?	558
9.11	Tipps & Tricks	560
9.11.1	Wie prüft man die ADO-Versionsnummer?	560
9.11.2	Access-Datenbanken exklusiv öffnen	560
9.11.3	Access-Datenbanken im Netzwerk	561
9.11.4	Alle aktiven Verbindungen zur Datenbank auflisten	561
9.11.5	Das Datenbank-Kennwort ändern	562
9.11.6	Abfragen über mehrere Datenbanken	563
9.11.7	Datenbanken reparieren/komprimieren	563

10	Microsoft SQL Server	565
10.1	Ein erster Schock ...	565
10.2	Allgemeines	566
10.2.1	SQL Server LocalDB	567
10.2.2	SQL Server Express	568
10.2.3	Unterschiede SQL Server-Varianten/Jet-Engine	569
10.2.4	Client- versus Fileserver-Programmierung	571
10.2.5	Installation SQL Server Express	573
10.2.6	Netzwerkzugriff für den SQL Server Express	577
10.2.7	Die wichtigsten Tools von SQL Server	579
10.2.8	Vordefinierte Datenbanken	582
10.2.9	Einschränkungen	583
10.2.10	Weitere SQL Server-Funktionen im Kurzüberblick	584
10.2.11	Datenbanken verwalten	585
10.3	Transact-SQL – die Sprache des SQL Servers	587
10.3.1	Schreibweise	587
10.3.2	Kommentare	588
10.3.3	Zeichenketten	588
10.3.4	Variablen deklarieren/verwenden	589
10.3.5	Bedingungen mit IF/ELSE auswerten	590
10.3.6	Verwenden von CASE	591
10.3.7	Verwenden von WHILE...BREAK/CONTINUE	591
10.3.8	Datum und Uhrzeit in T-SQL	592
10.3.9	Verwenden von GOTO	592
10.4	Praktisches Arbeiten mit dem SQL Server	593
10.4.1	Erstellen neuer SQL Server-Datenbanken	593
10.4.2	Erzeugen und Verwalten von Tabellen	594
10.4.3	Erzeugen und Verwenden von Sichten (Views)	595
10.4.4	Verwenden von Gespeicherten Prozeduren	597
10.4.5	Programmieren von Triggern	601
10.4.6	Erzeugen von Datenbankdiagrammen	605
10.4.7	Volltextabfragen	606
10.4.8	Datenbanken sichern und wiederherstellen	612
10.5	Fehlerbehandlung	615
10.5.1	Das Fehlermodell des SQL Servers	616
10.5.2	Verwenden von @@ERROR	616
10.5.3	Verwenden von RAISEERROR	617
10.5.4	Fehlerbehandlung mit TRY...CATCH	618
10.5.5	Fehlerbehandlung mit den ADO	620
10.6	Datensicherheit auf dem Microsoft SQL Server	622

10.6.1	Überblick Sicherheitsmodell	623
10.6.2	Verwalten mit dem SQL Server Management Studio	625
10.6.3	Verwalten mit T-SQL	629
10.7	Tipps & Tricks	632
10.7.1	Alle registrierten Microsoft SQL Server ermitteln	632
10.7.2	Alle Datenbanken ermitteln	633
10.7.3	Alle Tabellen ermitteln	633
10.7.4	Eine Tabelle löschen	634
10.7.5	Anzahl der Datensätze beschränken	635
10.7.6	Platzhalterzeichen in TSQL	636
10.7.7	Leerzeichen entfernen	636
10.7.8	Teilstrings erzeugen	636
10.7.9	Mit einer Datenbankdatei verbinden	637
10.7.10	Warum wird @@ERROR nicht korrekt verarbeitet?	638
10.7.11	Die Anzahl der Datensätze bestimmen	638
10.7.12	Warum sind Abfragen mit Platzhaltern so langsam?	638
10.7.13	Groß-/Kleinschreibung berücksichtigen	639
10.7.14	Das Ergebnis einer Stored Procedure speichern	639
10.7.15	Eine Datenbank umbenennen	639
10.7.16	Eine Datenbank zwischen Servern verschieben	640
10.7.17	Die Datenbankstruktur kopieren	641
10.7.18	Nach dem Löschen IDENTITY auf 0 setzen	642
10.7.19	Eine Tabellenspalte umbenennen	642
10.7.20	Temporäre Tabellen unterscheiden	642
10.7.21	Daten aus verschiedenen Datenbanken anzeigen	643
10.7.22	Einen SMO-Mapper realisieren	643
10.8	Übersichten	648
10.8.1	Datentypen	648
10.8.2	Unterschiede Access- und SQL Server-Datentypen	649
11	Access und Azure SQL	651
11.1	Einführung in SQL Azure-Datenbanken	652
11.1.1	Das Grundprinzip der "Webdatenbank"	652
11.1.2	Der Azure-Server	654
11.1.3	Die Frage nach den Kosten	655
11.2	Einrichten des Servers	656
11.2.1	Die zentrale Organisationsstruktur	657
11.2.2	Einen Server und eine Datenbank erstellen	658
11.2.3	IP-Filter konfigurieren	662
11.2.4	Bemerkungen zum neu erstellten Account	663

11.2.5	Die drei konzeptionellen Zugriffsmodelle	663
11.3	Administrieren von Azure SQL-Datenbanken	666
11.3.1	Zugriff mit dem SQL Server Management Studio	666
11.3.2	Weitere Accounts erstellen	668
11.3.3	Lokale Datenbanken migrieren	671
11.3.4	Migrieren von Access-Datenbanken	674
11.4	Praktische Umsetzung in Access	676
11.4.1	Tabellen einbinden	677
11.4.2	DAO- oder ADO-Zugriff – keine Frage!	680
11.4.3	Unsere AzureSQL-Library	680
11.4.4	Verbindung mit ADO aufbauen	681
11.4.5	Datenbank erstellen	685
11.4.6	Ist die Datenbank schon vorhanden?	686
11.4.7	Den aktuellen "Füllstand" abrufen	687
11.4.8	Was passiert, wenn die Datenbank zu klein wird?	690
11.4.9	Eine Datenbankkopie erstellen	690
11.4.10	Tabelle(n) erstellen	691
11.4.11	Daten exportieren	692
11.4.12	Daten einbinden	693
11.4.13	Daten lesen	694
11.4.14	Daten schreiben	695
11.5	Abschließende Hinweise	696
11.5.1	Synchronisieren	696
11.5.2	Performance-Tipps	697
11.5.3	Die Firewall per T-SQL konfigurieren	697
11.5.4	Arbeiten mit sqlcmd	699
11.6	Fazit	699
12	Zugriff auf SQLite	701
12.1	Was eigentlich ist SQLite?	702
12.1.1	Vorteile	702
12.1.2	Nachteile	703
12.2	Vorbereitungen	703
12.2.1	Download/Installation des ODBC-Treibers	704
12.2.2	Download/Installation SQLite for Excel	705
12.3	Datenbank-Tools	706
12.3.1	Database .NET	706
12.3.2	SQLite Administrator	708
12.4	Praktische Aufgabenstellungen	709
12.4.1	Einbinden von SQLite-Tabellen per Assistent	709

12.4.2	Einbinden von SQLite-Tabellen per VBA-Code	710
12.4.3	Datenbank per Code erstellen	711
12.4.4	Tabellen erzeugen	712
12.4.5	Datenbankzugriff per ADO realisieren	714
12.4.6	Die Bedeutung von Transaktionen bei SQLite	715
12.4.7	SOUNDEX verwenden	716
12.4.8	Volltextabfragen realisieren	717
12.5	Praxisbeispiele	720
12.5.1	Verwenden der Volltextsuche	720
12.5.2	Implementieren der Klasse SQLiteDatabase	725
12.5.3	Verwenden der Klasse SQLiteDatabase	735
12.6	Tipps & Tricks	737
12.6.1	Für Liebhaber der Kommandozeile – Sqlite3.exe	737
12.6.2	Eine SQLite-Datenbank reparieren	739
12.6.3	Eine Beispieldatenbank herunterladen	740
12.6.4	Testen ob Tabelle vorhanden ist	740
12.6.5	Die Datenbank defragmentieren	740
12.6.6	Mehrere Datenbanken verknüpfen	741
12.6.7	Eine Abfrage/Tabelle kopieren	741
12.6.8	Ein Backup implementieren	742
12.6.9	Tabellen zwischen Datenbanken kopieren	742
12.6.10	Ersatz für TOP	742
12.6.11	Metadaten auswerten	743
12.6.12	Timestamp als Defaultwert verwenden	744
12.6.13	Export in XML-Format	745
12.7	Fazit	745

Teil III: Weitere Technologien

13	Dateien und Verzeichnisse	749
13.1	Allgemeines	749
13.1.1	ANSI/ASCII/Unicode	749
13.1.2	Gemeinsamer Dateizugriff	750
13.1.3	Verwenden der File System Objects	750
13.2	Zugriff auf das Dateisystem	752
13.2.1	Ermitteln aller Laufwerke und deren Eigenschaften	752
13.2.2	Ermitteln aller Verzeichnis-Eigenschaften	753
13.2.3	Auflisten aller Unterverzeichnisse eines Folders	754
13.2.4	Rekursiv alle Unterverzeichnisse auflisten	755
13.2.5	Ein Verzeichnis erzeugen	756

13.2.6	Das Datenbankverzeichnis bestimmen	757
13.2.7	Abfragen des Temp-/System/...-Verzeichnisses	757
13.2.8	Prüfen, ob eine Datei existiert	758
13.2.9	Verzeichnisse/Dateien kopieren/löschen	759
13.2.10	Auflisten aller Dateien eines Verzeichnisses	759
13.2.11	Name, Pfad und Extension einer Datei ermitteln	760
13.2.12	Einen Tempfile-Namen erzeugen	762
13.3	Textdateien	762
13.3.1	Klassischer Zugriff auf Textdateien	763
13.3.2	Zugriff auf Textdateien mit den File System Objects	765
13.4	Typisierte Dateien	768
13.4.1	Öffnen	768
13.4.2	Lesen/Schreiben	768
13.5	Weitere Dateien	770
13.5.1	Binärdateien	770
13.5.2	INI-Dateien	771
13.6	Die Registrierdatenbank	773
13.6.1	Einführung	773
13.6.2	API/VBA-Zugriff auf die Registrierungsdatenbank	774
13.6.3	API-Konstanten/Funktionen für den Registry-Zugriff	775
13.6.4	Prüfen, ob ein Schlüssel existiert	777
13.6.5	Einen vorhandenen Wert auslesen	777
13.6.6	Einen Schlüssel erstellen	778
13.6.7	Einen Wert setzen bzw. ändern	779
13.6.8	Einen Schlüssel löschen	779
13.6.9	Ein Feld löschen	779
13.6.10	Aufruf der Funktionen	780
13.7	Dateidialoge	782
13.7.1	Variante 1 (Office 16 Library)	783
13.7.2	Variante 2 (Windows-API)	785
13.7.3	Verzeichnisdialog (Windows-API)	790
13.8	Übersichten	792
13.8.1	Dateifunktionen in Access	792
13.8.2	FSO-Eigenschaften und -Methoden	793
13.9	Praxisbeispiele	794
13.9.1	Auf eine Textdatei zugreifen	794
13.9.2	Dateien suchen	799
13.9.3	Die Shellfunktionen verwenden	801
13.9.4	Einen Verzeichnisbaum in eine TreeView einlesen	806

14	XML in Theorie und Praxis	807
14.1	XML – etwas Theorie	807
14.1.1	Allgemeines	807
14.1.2	Der XML-Grundaufbau	810
14.1.3	Wohlgeformte Dokumente	811
14.1.4	Processing Instructions (PI)	813
14.1.5	Elemente und Attribute	814
14.1.6	Verwendbare Zeichensätze	815
14.2	XSD-Schemas	817
14.2.1	Das Grundprinzip	817
14.2.2	Ein XSD-Schema mit Microsoft Access erzeugen	819
14.3	XML-Verarbeitung mit dem DOM	823
14.3.1	Was ist das DOM?	823
14.3.2	Erste Schritte	824
14.3.3	Erzeugen von Instanzen	825
14.3.4	Laden von Dokumenten	826
14.3.5	XML-Fehlerprüfung/-Analyse	827
14.3.6	Erzeugen von XML-Dokumenten	829
14.3.7	Auslesen von XML-Dateien	831
14.3.8	Einfügen von Informationen	834
14.3.9	Attribute oder Element	837
14.3.10	Suchen in den Baumzweigen	840
14.3.11	Das Interpretieren von Leerzeichen	843
14.4	XML-Integration in Access	846
14.4.1	Importieren	846
14.4.2	Exportieren	849
14.4.3	XML-Transformation mit XSLT	853
14.4.4	Application-Objekt (ImportXML/ExportXML)	856
14.4.5	ADO-Recordset	859
14.5	Vor- und Nachteile von XML	861
14.5.1	Grundsätzlicher Vergleich	862
14.5.2	Zeitvergleich ADO/XML	862
14.6	Praxisbeispiele	864
14.6.1	Speichern im UTF-8-/UTF-16-Format	864
14.6.2	UTF-8-/UTF-16 aus einem ADO-Stream laden	866
14.6.3	XML-Daten asynchron verarbeiten	868
14.6.4	XML-Daten in einer TreeView darstellen	870
14.6.5	Navigieren zwischen einzelnen XML-Baumknoten	873
14.6.6	ADO-XML-Streams nachbearbeiten	876
14.6.7	Textdaten in XML-Dokumente umwandeln	879

15	SQL im Einsatz	881
15.1	Einführung	881
15.1.1	SQL-Dialekte	882
15.1.2	Kategorien von SQL-Anweisungen	882
15.2	Etwas (Datenbank-)Theorie	884
15.2.1	Allgemeines/Begriffe	884
15.2.2	Normalisieren von Tabellen	889
15.2.3	Beziehungen zwischen den Tabellen	894
15.2.4	Verknüpfen von Tabellen	896
15.3	Testprogramm und Beispieldatenbank	900
15.3.1	Hinweise zur Bedienung	900
15.3.2	Die Beispieldatenbank im Überblick	900
15.3.3	ADO Query	901
15.3.4	Bemerkungen	902
15.4	Daten abfragen	903
15.4.1	Abfragen mit SELECT	904
15.4.2	Alle Spalten auswählen	904
15.4.3	Auswahl der Spalten	905
15.4.4	Filtern	906
15.4.5	Beschränken der Ergebnismenge	912
15.4.6	Eindeutige Records/doppelte Datensätze	913
15.4.7	Tabellen verknüpfen	914
15.4.8	Tabellen vereinigen	917
15.4.9	Datensätze sortieren	918
15.4.10	Datensätze gruppieren	918
15.4.11	Unterabfragen	920
15.4.12	Anlage-Felder mit SQL verwalten	924
15.4.13	History-Felder mit SQL abfragen	926
15.4.14	Mehrwertige Felder mit SQL abfragen	928
15.5	Daten manipulieren	929
15.5.1	Einfügen einzelner Datensätze	930
15.5.2	Einfügen von Abfragedaten	931
15.5.3	Exportieren/Importieren von Abfragedaten	933
15.5.4	Aktualisieren/Ändern	935
15.5.5	Löschen	936
15.6	Erweiterte SQL-Funktionen	937
15.6.1	Berechnete/Formatierte Spalten	938
15.6.2	Berechnungsfunktionen	944
15.6.3	NULL-Werte	945
15.6.4	Datum und Zeit in SQL-Abfragen	947

15.6.5	Datentypumwandlungen	952
15.6.6	Kreuztabellenabfragen	953
15.6.7	Steuerelemente in Abfragen verwenden	956
15.6.8	Globale Variablen in Abfragen verwenden	957
15.7	Datenbankverwaltung mit SQL (DDL)	957
15.7.1	Datenbanken	958
15.7.2	Tabellen	958
15.7.3	Indizes	959
15.7.4	Tabellen/Indizes löschen oder verändern	962
15.7.5	Sichten (Views)	962
15.7.6	Nutzer- und Rechteverwaltung	964
15.7.7	Transaktionen	964
15.8	SQL in der Access-Praxis	965
15.8.1	SQL in Abfragen	965
15.8.2	SQL im Basic-Code	966
15.8.3	SQL beim Oberflächenentwurf	968
15.8.4	VBA-Funktionen in SQL-Anweisungen	968
15.9	Tipps & Tricks	970
15.9.1	Wie kann ich die Anzahl der Datensätze ermitteln?	970
15.9.2	Wie nutze ich Datumsteile in SQL zur Suche?	971
15.9.3	Die Groß-/Kleinschreibung berücksichtigen	971
15.9.4	Warum erhalte ich zu viele Datensätze ?	972
15.9.5	Doppelte Datensätze aus einer Tabelle löschen	973
15.10	Praxisbeispiele	974
15.10.1	Ein komfortables SQL-Abfrageprogramm erstellen	974
15.10.2	Datum und Zeit in SQL einbauen	976
16	Anwendungsdesign	979
16.1	Access-Oberflächengestaltung	979
16.1.1	Beschriften der Kopfzeile	980
16.1.2	Informationen in der Statuszeile anzeigen	982
16.1.3	Fortschrittsanzeige mit dem Progressbar realisieren	983
16.1.4	Navigationsbereich konfigurieren	984
16.1.5	Access-Hauptfenster komplett ausblenden	990
16.2	Steuern der Anwendung	993
16.2.1	Autostart mit AutoExec-Makro	993
16.2.2	Formulare automatisch starten	995
16.2.3	Warten auf das Ende	995
16.2.4	Access per VBA beenden	996
16.2.5	Den Runtime-Modus aktivieren	996

16.2.6	Befehlszeilen-Optionen verwenden	997
16.3	Entwickeln von Assistenten und Add-Ins	999
16.3.1	Assistenten-Typen	999
16.3.2	Einbinden der Assistenten in die Access-IDE	999
16.3.3	Menü-Assistent (Beispiel)	1001
16.3.4	Objekt-Assistent (Beispiel)	1003
16.3.5	Steuerelemente-Assistent (Beispiel)	1007
16.3.6	Eigenschaften-Assistent	1010
16.4	Entwickeln/Einbinden von Managed Add-Ins	1011
16.4.1	Interaktion Anwendung/Add-In	1011
16.4.2	Entwurf des Add-Ins	1012
16.4.3	Oberfläche	1014
16.4.4	Der Quellcode des Add-Ins	1015
16.4.5	Formularentwurf für das Add-In	1017
16.4.6	Kompilieren und Einbinden	1017
16.4.7	Testen	1018
16.5	Libraries unter Access	1019
16.5.1	Erstellen und Einbinden	1019
16.5.2	Debugging	1020
16.5.3	Einfaches Beispiel	1021
16.5.4	Hinweise	1023
16.6	Praxisbeispiele	1024
16.6.1	Mehr über die aktuelle Access-Version erfahren	1024
16.6.2	Access neu starten/Datenbank neu laden	1025
16.6.3	Mit Reference-Objekten arbeiten	1026
16.6.4	Benutzerdefinierte Eigenschaften einsetzen	1030
16.6.5	Den aktuellen Datenbanknutzer ermitteln	1034
16.6.6	Überlappende Fenster einstellen	1034
16.6.7	Access-Optionen abrufen/setzen	1035
17	Menüband und Backstage	1037
17.1	Allgemeine Grundlagen	1037
17.1.1	Manuelle Konfigurationsmöglichkeiten	1038
17.1.2	Grundprinzip der Programmierung	1039
17.1.3	Verwenden der Tabelle USysRibbons	1041
17.1.4	Application.LoadCustomUI als Alternative	1042
17.2	Ein kleines Testprogramm	1042
17.3	Praktische Aufgabenstellungen	1044
17.3.1	Infos über Steuerelemente und Symbole erhalten	1044
17.3.2	Hauptregisterkarten ausblenden	1046

17.3.3	Einzelne Registerkarten ausblenden	1047
17.3.4	Einzelne Gruppen ausblenden	1048
17.3.5	Registerkarten, Gruppen und Schaltflächen einfügen	1049
17.3.6	Ereignisbehandlung mit VBA-Code/Makros	1051
17.3.7	Verändern von Eigenschaften mit VBA-Callbacks	1052
17.3.8	Aktualisieren des Menübands per VBA-Code	1052
17.3.9	Kontextabhängige Registerkarten	1053
17.3.10	Registerkarten per VBA aktivieren	1054
17.3.11	Fehlermeldungen des Menübands anzeigen	1055
17.3.12	Vorhandene Funktionen des Menübands ändern	1055
17.3.13	Die Schnellzugriffsleiste erweitern/programmieren	1056
17.4	Übersicht der Steuerelemente	1057
17.4.1	labelControl-Steuerelement	1057
17.4.2	button-Steuerelement	1057
17.4.3	separator-Steuerelement	1061
17.4.4	toggleButton-Steuerelement	1062
17.4.5	buttonGroup-Steuerelement	1063
17.4.6	checkBox-Steuerelement	1064
17.4.7	editBox-Steuerelement	1065
17.4.8	comboBox-Steuerelement	1066
17.4.9	dropDownElement-Steuerelement	1070
17.4.10	gallery-Steuerelement	1073
17.4.11	menu-Steuerelement	1075
17.4.12	splitButton-Steuerelement	1078
17.4.13	dialogBoxLauncher	1079
17.5	Die Backstage-Ansicht anpassen	1081
17.5.1	Die Standardansicht verändern	1082
17.5.2	Die drei möglichen Layouts für Registerkarten	1084
17.5.3	Die neuen Möglichkeiten von Gruppen	1091
17.5.4	Verwenden von LayoutContainern	1093
17.5.5	Das Verhalten der Schaltflächen beeinflussen	1094
17.5.6	Mit VBA/Makros auf Ereignisse reagieren	1095
17.6	Tipps & Tricks	1096
17.6.1	Die guten alten Access 2003-Menüs anzeigen	1096
17.6.2	Das Office-Menü für Access 2007 anpassen	1098
17.6.3	XML-Daten komfortabel editieren	1099
17.6.4	Arbeiten mit dem RibbonCreator	1100
17.7	Übersichten	1101

18	Programmschnittstellen	1103
18.1	Zwischenablage	1103
18.1.1	Kopieren/Einfügen mittels DoCmd-Objekt	1103
18.1.2	Ein Clipboard-Objekt programmieren	1104
18.2	API- und DLL-Einbindung	1107
18.2.1	Allgemeines	1107
18.2.2	Und was ist mit der 64-Bit Access Version?	1108
18.2.3	Woher bekomme ich Infos über die Win32-API?	1110
18.2.4	Einbinden der Deklaration	1111
18.2.5	Wert oder Zeiger?	1113
18.2.6	Übergabe von Strings	1114
18.2.7	Verwenden von As Any	1115
18.2.8	Übergabe von Arrays	1115
18.2.9	Besonderheiten mit Records	1116
18.2.10	Zuordnen der Datentypen	1118
18.3	OLE/ActiveX	1119
18.3.1	Überblick	1119
18.3.2	OLE	1120
18.3.3	Übersicht zum Objektfeld-Steuerelement	1121
18.3.4	Programmieren mit dem Objektfeld	1124
18.3.5	ActiveX-Code-Komponenten (OLE-Automation)	1127
18.3.6	Programmieren mit ActiveX-Code-Komponenten	1128
18.4	DDE	1130
18.4.1	Funktionsprinzip	1130
18.4.2	Verwenden von Steuerelementen	1131
18.4.3	VBA-Programmierung	1132
18.5	Scanner-Unterstützung per WIA	1134
18.5.1	Was kann WIA?	1134
18.5.2	Installation/Vorbereitung	1134
18.5.3	Einstieg mit kleiner Beispielanwendung	1135
18.5.4	Reagieren auf das Verbinden/Trennen von Geräten	1136
18.5.5	Ermitteln der verfügbaren Geräte	1137
18.5.6	Anzeige der Geräteeigenschaften	1137
18.5.7	Ein Bild einlesen	1139
18.5.8	Bild(er) drucken (Assistent)	1140
18.5.9	Den Scanner-Assistent aufrufen	1141
18.5.10	Grafikbearbeitung	1142
18.6	Zugriff auf Excel-Arbeitsmappen	1144
18.6.1	Zugriffsmöglichkeiten	1144
18.6.2	TransferSpreadsheet	1145

18.6.3	Import/Export per DAO	1149
18.6.4	Daten auslesen	1153
18.6.5	Zugriff auf Tabellenbereiche	1155
18.6.6	OLE-Automation	1156
18.7	Praxisbeispiele	1159
18.7.1	Eine externe Anwendung starten	1159
18.7.2	Informationen über das aktuelle System ermitteln	1161
18.7.3	Das aktuelle Betriebssystem ermitteln	1164
18.7.4	Den Windows-Lizenznehmer ermitteln	1167
18.7.5	Zeitmessungen in Access durchführen	1169
18.7.6	Microsoft Word über ActiveX steuern	1173
18.7.7	Microsoft Excel über ActiveX steuern	1177
	Index	1179



Vorwort

Microsoft Access ist weit mehr als nur eine komfortable Datenbankverwaltung, denn dem fortgeschrittenen Anwender steht darüber hinaus eine vollständige und vor allem preiswerte Entwicklungsumgebung für individuell geprägte Datenbankanwendungen zur Verfügung. Zusammen mit der objekt- und ereignisorientierten Programmiertechnik bilden VBA, SQL, DAO/ADO, Makros und XML ein mächtiges Werkzeug, um leistungsfähige Datenbank-Applikationen mit besserer Performance und auf beachtlich höherem Niveau zu erstellen.

Zum Buchinhalt

HINWEIS: Dieses Buch bietet Ihnen nicht nur eine fundierte Einführung in die Programmierung von Datenbanken mit Access 2016, sondern ist gleichermaßen auch für die Versionen 2010 und 2013 geeignet.

Die Autoren haben bei der Zusammenstellung des Inhalts weniger aus den zu Access 2016 mitgelieferten Dokumentationen, sondern vor allem aus eigenen Quellen und langjährigen praktischen Erfahrungen geschöpft:

- Lehrgänge für Datenbankprogrammierer
- Vorlesungen über Datenbankprogrammierung an Fachhochschulen
- und (last, but not least) das zahlreiche Feedback unserer Leser zu unseren Vorgängertiteln

Mit VBA (*Visual Basic for Applications*) stoßen Sie das Tor zur Windows-Programmierung weit auf und greifen direkt auf die mächtige Access-Bibliothek, die DAO-/ADO-Datenzugriffsobjekte oder auf das Windows-API (*Application Programming Interface*) zu. Verständlicherweise ist es im Rahmen dieses Buches unmöglich, auf alle Funktionen sowie auf alle Objekte und Sprachelemente der einzelnen Bibliotheken einzugehen. Dies ist Sache der Befehlsreferenz, auf die Sie am bequemsten über die integrierte Online-Hilfedatei zugreifen können. Ziel des Buches soll es sein, einen Gesamtüberblick zu geben und praktische Konzepte zu vermitteln.

In den insgesamt 18 Kapiteln finden Sie:

- Ausführliches Know-how über die objekt- und ereignisorientierte Gestaltung der Benutzerschnittstelle von Microsoft Access
- Eine gestraffte Einführung in die Programmierung von Makros
- Eine umfassende Einführung in VBA, SQL und XML mit einer übersichtlichen und auf das Wesentliche reduzierten Sprachbeschreibung
- Eine gestraffte Einführung in den Datenbankzugriff (Jet, Microsoft SQL Server, Azure SQL) unter Verwendung von DAO/ADO
- Viele Kapitel verfügen über einen Übersichtsteil, in dem wichtige Informationen (z.B. relevante Eigenschaften, Methoden und Ereignisse von zentralen Objekten) griffbereit zur Verfügung stehen
- Zahlreiche Praxisbeispiele am Ende der Kapitel dienen der Vertiefung der vermittelten theoretischen Grundlagen.

Begleitdateien

Den Quellcode dieses Buchs können Sie sich unter folgender Adresse herunterladen:

LINK: www.doko-buch.de

Für Einsteiger einige Hinweise, die wir aufgrund von Erfahrungen mit unseren Vorgängertiteln diesmal nicht vergessen wollen:

- Sie sollten natürlich vorher Microsoft Access 2016/2013/2010 auf Ihrem PC installiert haben.
- Sollte doch einmal eine Datenbank/Datei beim Testen der Beispiele nicht gefunden werden, müssen Sie im betreffenden Modul die Pfadangaben anpassen.
- Falls einmal ein Beispiel nicht funktionieren sollte, lesen Sie die beigelegte Readme-Datei.
- Zumindest für das Kapitel 10 ist die Installation des Microsoft SQL Server ab Version 2008 bzw. der entsprechenden Express Version erforderlich. Beachten Sie die erforderlichen Sicherheitseinstellungen für den Zugriff.

Sicherheitseinstellungen

Im Zeitalter wachsender Internetkriminalität müssen Sie sich daran gewöhnen, dass zum Schutz Ihres Computers reichlich Sicherheitshürden aufgebaut wurden: Access-Sicherheit (VBA), System-sicherheit (Windows 7/8/10), SQL Server-Sicherheit, .NET-Sicherheit.

Das führt für Sie als VBA-Programmierer zunächst zu folgender Konsequenz:

HINWEIS: Aufgrund der restriktiven Sicherheitsanforderungen von Access 2016 wird es Ihnen in der Regel nicht gelingen, ohne entsprechende Anpassungen der Entwicklungsumgebung, ihren eigenen Code bzw. die zu diesem Buch mitgelieferten Beispiele zum Laufen zu bringen!

Auch unter Access 2016 wird VBA-Code standardmäßig als "unsicherer Code" eingestuft. Falls das Formular den Code nicht ausführt und stattdessen unterhalb des Menübands eine Sicherheitswarnung zeigt, klicken Sie auf die *Inhalt aktivieren*-Schaltfläche.

Um diese lästigen Sicherheitswarnungen generell zu verhindern, sollten Sie (zumindest für die Dauer der Programmentwicklung) im *Sicherheitscenter* Ihr Datenbankverzeichnis als *Vertrauenswürdigen Speicherort* hinzufügen.

HINWEIS: Eine detaillierte Anleitung, wie Sie mit diesem "heißen Eisen" am besten umgehen, finden Sie im Abschnitt "Sicherheitseinstellungen" des Kapitels 1 (Seite 43).

Ein gutgemeinter Ratschlag an Einsteiger

Programmieren lernt man nur durch Beispiele! Dies ist eine knallharte Wahrheit, um die sich eine zu stark akademisch geprägte Leserschaft gern herumogeln möchte. Wie alle unsere anderen Programmier-Bücher ist deshalb auch dieses Buch kein Lehrbuch, das Sie nach dem Motto "Jetzt lernen wir Microsoft Access" linear von vorn nach hinten durcharbeiten sollen. Sehr schnell wird Ihnen dabei unterwegs die Luft ausgehen und Sie verlieren die Lust. Falls Sie es dennoch bis zum Ende schaffen, sind Sie dennoch nicht in der Lage, eine praxistaugliche Datenbankapplikation zu erstellen.

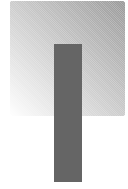
Wir empfehlen hingegen dem Einsteiger, sich zunächst nur auf einige ausgewählte Kapitel zu konzentrieren und parallel zum Studium des Buchs und tatkräftiger Einbeziehung der Online-Hilfe ein privates Datenbankprojekt in Angriff zu nehmen, wie zum Beispiel die Verwaltung der eigenen Bücher- oder CD-Sammlung. Der Lerneffekt, der aus selbst begangenen und selbst behobenen Fehlern resultiert, ist ein ungeheurer!

Motiviert durch kleine Erfolgserlebnisse und Aha-Effekte werden Sie Ihr Projekt und damit Ihr Wissen schrittweise vergrößern und verfeinern und sich nach dem Prinzip "soviel wie nötig" nur auf die Kapitel konzentrieren, die Sie zur Lösung Ihrer aktuellen Probleme wirklich benötigen.

Nobody is perfect

Sie werden – trotz der 1200 Seiten – in diesem Buch nicht alles finden, was Access zu bieten hat. Manches ist sicher in einem anderen Spezialtitel noch besser oder ausführlicher beschrieben. Aber Sie halten mit unserem Buch einen optimalen und überschaubaren Breitband-Mix in den Händen, der sowohl vertikal vom Einsteiger bis zum Profi als auch horizontal von den einfachen Sprachelementen bis hin zu komplexen Anwendungen jedem etwas bietet, ohne dabei den Blick auf das Wesentliche zu verlieren.

Wir hoffen, dass wir Ihnen mit diesem Buch einen nützlichen Begleiter bei der Access-Programmierung zur Seite gestellt haben, der es verdient, seinen Platz nicht im Regal, sondern griffbereit neben dem Computer einzunehmen.



Teil I: Grundlagen

- Einführung
- Programmierung mit VBA
- Makros – eine Einführung
- Formulare und Steuerelemente
- Berichte
- Programmieren mit Objekten

Einführung

Dieses Kapitel soll Ihnen einen ersten Überblick über die VBA-Programmierung unter Microsoft Access (ab Version 2010) vermitteln. Da das Buch nicht für den absoluten Einsteiger geschrieben wurde, gehen die Autoren davon aus, dass Sie bereits über gewisse Erfahrungen in der konventionellen Access-Datenbankprogrammierung verfügen, die Entwicklungsumgebung bedienen können und zumindest auch wissen, wie ein Makro funktioniert.

1.1 VBA-Programmierung in Access

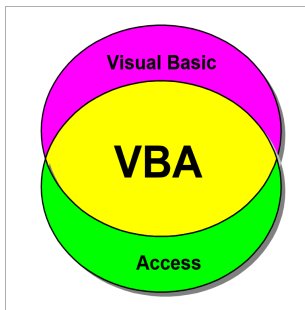
VBA (Visual Basic for Applications) ist weit mehr als nur eine Alternative zur gängigen Makro-Programmierung unter Microsoft Access. Es handelt sich hier um eine komfortable und leistungsfähige Programmiersprache für alle Microsoft Office-Produkte.

1.1.1 Visual Basic versus VBA

Im Unterschied zu *Visual Basic* ist allerdings *VBA* keine eigenständige Sprache, mit der allein man lauffähige Programme entwickeln könnte, sondern es lassen sich mit VBA bestimmte Anwendungen, wie z.B. Microsoft Access, in ihrer Funktionalität erweitern bzw. nutzerspezifischen Anforderungen anpassen. Oder fachmännischer ausgedrückt: Von VBA aus lassen sich die Objektmodelle von Access, Word, Excel etc. steuern. Wenn wir im Folgenden mal von "Visual Basic" und mal von "VBA" sprechen, so sollten Sie diese "Laxheit" nicht auf die Goldwaage legen, denn gemeint ist ein und dasselbe. Fakt ist, dass VBA als Untermenge vollständig in Visual Basic enthalten ist¹, bei weitem aber nicht alle Features von Visual Basic unterstützt. Man könnte mit anderen Worten VBA auch als "kleinsten gemeinsamen Nenner" von Visual Basic² und Access bezeichnen (siehe folgende).

¹ Zumindest seit Office 2000 wird dieser Anspruch erhoben.

² Das galt zumindest bis zur Version VB 6.0.



HINWEIS: Es gibt zahlreiche Anwendungen, in die VBA integriert ist, und dabei handelt es sich bei weitem nicht nur um Microsoft Office-Produkte.

1.1.2 Objekt- und ereignisorientierte Programmierung

Im Unterschied zur klassischen prozeduralen Anwendungsentwicklung hat sich der VBA-Programmierer verstärkt mit folgenden Begriffen auseinander zu setzen:

Objekte (Objects)

Die verschiedenen Office-Anwendungen bieten ihre Funktionalität in Form von Objekten an, auf die mit VBA-Code zugegriffen werden kann. Obwohl wir mittlerweile im .NET-Zeitalter angekommen sind, liefert das COM (*Component Object Model*) immer noch die Grundlage für die Zusammenarbeit der verschiedenen Office-Objekte (auch über Anwendungsgrenzen hinaus).

Aus Anwendersicht kennen Sie bereits Datenbankobjekte wie Tabelle, Abfrage, Formular, Bericht, Makro, Modul. Für den VBA-Programmierer sind neben den datenbankspezifischen Objekten *Tabelle* und *Abfrage* vor allem *Forms*, *Reports* und *Controls* von besonderem Interesse, da sie die wesentlichen Anwenderschnittstellen zur Verfügung stellen. Hinzu kommen die *Module*, die keine Objekte im engeren Sinne sind, sondern lediglich Quelltextbibliotheken darstellen.

Eigenschaften (Properties)

Hinter diesem Begriff verbergen sich die Attribute von Objekten, wie z.B. Höhe (*Height*) und Breite (*Width*) eines Textfeldes. Jedes Objekt verfügt über seine eigene Menge von Eigenschaften, die teilweise nur zur Entwurfszeit (im Eigenschaftenfenster) oder zur Laufzeit (per Quellcode) zugewiesen werden können. Allgemein unterscheiden wir zwischen Format-, Daten- und anderen Properties (siehe dazu Kapitel 4).

Methoden (Methods)

Diese auf dem Objekt definierten Funktionen und Prozeduren hauchen "Leben" in die bislang nur mit statischen Attributen behafteten Objekte. So erzeugt z.B. die *Line*-Methode eine Linie vor dem Hintergrund eines Reports, mit *Print* kann Text im *Debug*-Objekt (Testfenster) ausgegeben werden usw.

Ereignisse (Events)

Diese werden von Nachrichten ausgelöst, die vom Objekt empfangen werden. Dieser Nachrichtenverkehr stellt die eigentliche Schnittstelle zu Windows dar. Der Mausklick auf ein Formular löst z.B. ein *MouseDown*-Ereignis für dieses Objekt aus. Wir unterscheiden zwischen Fenster- und Fokus-, Tastatur- und Maus- sowie Daten- und Filter-Events (siehe Kapitel 4 und 5). Eine Hauptaufgabe des Programmierers ist das Schreiben von so genannten Ereignisbehandlungsroutinen (Event-Handler), in denen er festlegt, wie das Objekt bei Eintreffen eines bestimmten Ereignisses zu reagieren hat.

Einen ersten Eindruck der objekt- und ereignisorientierten VBA-Programmierung gewinnen Sie am besten anhand der Beispiele im Praxisteil dieses Kapitels. Weitere grundlegende Ausführungen zur objektorientierten Programmierung (OOP) folgen in Kapitel 6.

1.1.3 VBA- oder Makro-Programmierung?

Die klassischen Programmiermöglichkeiten unter Microsoft Access (Makro- und Ausdruckseditor, QbE-Fenster etc.) sind bereits so komfortabel, dass Sie mit Recht die Frage stellen werden: "Wozu brauche ich denn dann überhaupt noch VBA?"

Hier eine Aufzählung wichtiger Argumente, die für den Einsatz von VBA sprechen:

- Erhöhung der Performance (Ausführungsgeschwindigkeit)
- Definition eigener Funktionen (Makros können keine Werte zurückliefern!)
- Spezielle Fehlerbehandlungsroutinen sind möglich
- Verwendung von Ereignissen mit Übergabeparametern
- Definition neuer Objekte (Tabellen, Abfragen, Formulare, Berichte) per Code
- Zugriff auf andere Windows-Programme per OLE oder DDE
- Nutzung spezieller Funktionen des Windows-API
- Auslagerung von Teilen des Anwendungscodes in eine Bibliothek
- Arbeiten mit den Datenbank-Objekten
- Zugriff auf integrierte Funktionen des Datenbanksystems (z.B. Routinen zur Datendefinition oder SQL-Server-Prozeduren)

Auch der genialste Makro-Programmierer wird wohl früher oder später feststellen, dass er an Grenzen stößt, die sich nur mit dem mächtigen Instrumentarium von VBA durchbrechen lassen.

Der Umstieg zu VBA wird für den erfahrenen Makro-Programmierer vor allem durch zwei in Microsoft Access eingebaute Features erleichtert:

- Automatische Makro-Konvertierung
- *DoCmd*-Objekt

Konkrete Anleitungen dazu finden Sie im entsprechenden Einführungsbeispiel (ab Seite 50) bzw. im Übersichtsteil dieses Kapitels (ab Seite 68).

HINWEIS: Allerdings soll – bei aller Liebe zu VBA – auch nicht verschwiegen werden, dass die Programmierung von Datenmakros und Web-Datenbanken mit VBA nicht möglich ist.

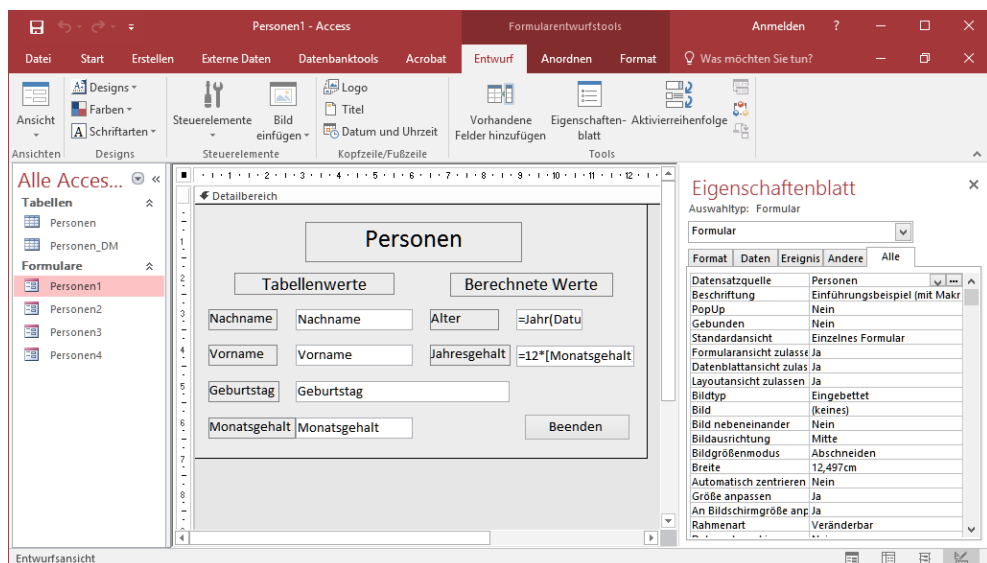
1.1.4 Die VBA-Entwicklungsumgebung

Da in diesem Buch die VBA-Programmierung eine zentrale Rolle spielt, werden wir uns im Folgenden schwerpunktmäßig auf die Elemente der VBA-Entwicklungsumgebung (IDE¹) konzentrieren und die übrigen Access-Bedienfunktionen weitgehend als bekannt voraussetzen.

1.1.5 Formularentwurf

Beim visuellen Entwurf der Benutzeroberflächen von Formularen bzw. Berichten haben wir es zunächst mit den standardmäßigen Fenstern der Access-IDE zu tun:

- Navigationsbereich
- Formular-/Berichtsfenster (Entwurfsansicht)
- Eigenschaftenblatt
- Menüband



¹ Integrated Development Environment

Da vorausgesetzt wird, dass Sie bereits über Erfahrungen beim visuellen Entwurf von Formularen/Berichten verfügen, kann wohl auf weitere Erklärungen verzichtet werden (siehe Einführungsbeispiel ab Seite 50).

1.1.6 Code-Fenster und Symbolleiste

Das Tor zur VBA-Programmierung öffnet sich Ihnen weit, nachdem Sie die Registerkarte *Datenbanktools* gewählt haben und auf die Schaltfläche *Visual Basic* (Befehlsgruppe *Makro*) klicken. Hier wählen Sie im Projekt-Explorer das gewünschte Formular aus.

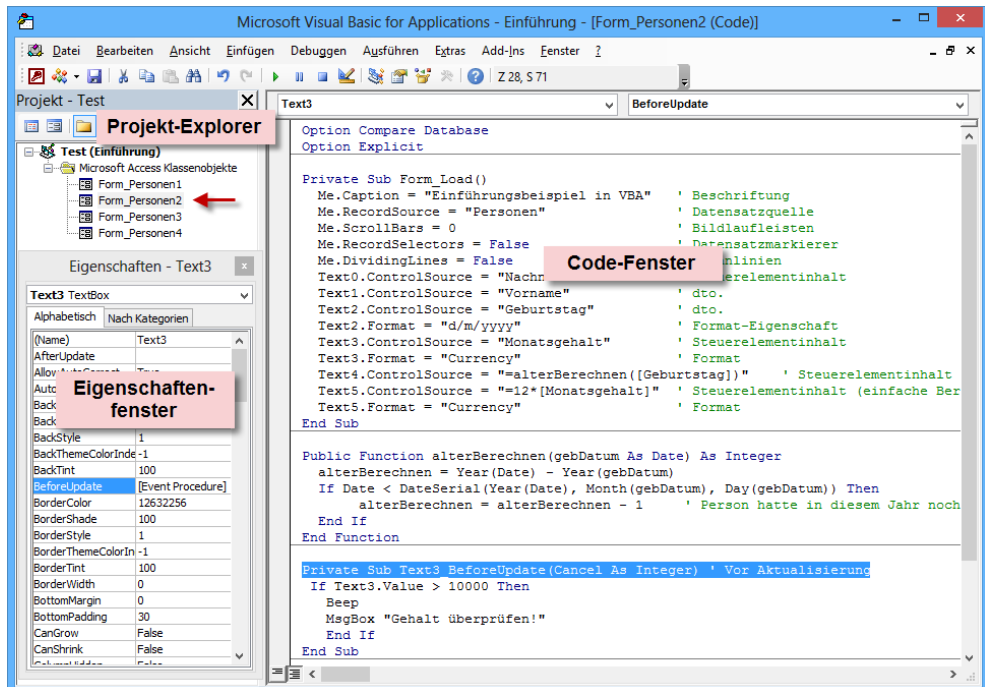
Die wichtigsten Fenster

Das VBA-Fenster sieht unter Access genauso aus wie bei den übrigen Office-Anwendungen.

Die wichtigsten Fenster innerhalb der VBA-IDE sind

- Code-Fenster (öffnen über **[F7]** oder *Ansicht/Code*),
- Eigenschaftsfenster (öffnen über **[F4]** oder *Ansicht/Eigenschaftsfenster*) und
- Projektfenster (öffnen über **[Strg]+[R]** oder *Ansicht/Projekt-Explorer*).

Auf einige Besonderheiten dieser Fenster werden wir zu einem späteren Zeitpunkt eingehen.



Die wichtigsten Menübefehle

Wenn Sie in der VBA-IDE arbeiten, wechselt das Menüband das Aussehen und stellt Ihnen eine Reihe von Bedienfunktionen zur Verfügung, die Sie speziell für die Arbeit mit dem Code-Editor benötigen. Sie werden feststellen, dass viele Funktionen starke Ähnlichkeiten zur Bedienung einer Textverarbeitung unter Windows aufweisen.

Die am häufigsten benötigten Menüpunkte sind über Schaltflächen. In Abhängigkeit vom momentanen Zustand des Editors können einige Schaltflächen gesperrt sein und/oder das Aussehen bzw. die Funktion wechseln. Außerdem haben Sie die Möglichkeit, über das Menü *Ansicht/Symbolleisten* Änderungen vorzunehmen.

Die folgende Tabelle zeigt eine Zusammenstellung der für den Einstieg in die Codeprogrammierung zunächst wichtigsten Schaltflächen samt zugehöriger Menübefehle.

Symbol	Menübefehl	Tasten	Erläuterung
	Ansicht/Microsoft Access	[Alt] + [F1]	Zeigt das Access-Hauptfenster an
	Ansicht/Code	[F7]	Zeigt das Code-Fenster an
	Ansicht/Projekt-Explorer	[Strg] + [R]	Zeigt den Projekt-Explorer an
	Ansicht/Objektkatalog	[F2]	Zeigt den Objektkatalog an
	Einfügen/Modul		Fügt ein neues (leeres) Modul in die Datenbank ein
	Datei/Speichern		Speichert den Inhalt des Moduls
	Bearbeiten/Ausschneiden	[Strg] + [X]	Schneidet markierten Code aus und kopiert ihn in die Zwischenablage
	Bearbeiten/Kopieren	[Strg] + [C]	Kopiert markierten Code in die Zwischenablage
	Bearbeiten/Einfügen	[Strg] + [V]	Fügt Text aus Zwischenablage in Code ein
	Bearbeiten/Suchen	[Strg] + [F]	Sucht markierten Text in diesem oder in anderen Codemodulen
	Bearbeiten/Rückgängig	[Strg] + [Z]	Macht die zuletzt durchgeführte Bedienaktion rückgängig (falls möglich)
	Bearbeiten/Wiederholen		Stellt den Ausgangszustand vor Bearbeiten/Rückgängig wieder her
	Ausführen/Fortsetzen	[F5]	Setzt die Ausführung des Codes nach einer Unterbrechung fort (wenn möglich)
	Ausführen/Unterbrechen		Unterbricht die Ausführung des Codes
	Ausführen/Zurücksetzen		Beendet die Ausführung des Codes und löscht Variablen
	Ausführen/Entwurfsmodus		Kehrt in den Entwurfsmodus zurück

1.1.7 Das Eigenschaftenfenster

Hier handelt es sich nicht mehr um das dem Makro-Programmierer bestens vertraute Eigenschaftenblatt mit den eingedeutschten Bezeichnungen, sondern hier haben Sie es mit einem völlig neuen Outfit und den originalen VBA-Bezeichnungen zu tun (z.B. *Datensatzquelle* → *RecordSource*). Als

weiteren Unterschied werden Sie feststellen, dass die *Ereignis*-Seite nicht mehr da ist (die Auswahl der Ereignisse wird nunmehr im Ereignisselector rechts oben im Code-Fenster vorgenommen).

Mit dem Objektselector (oberes Dropdownfeld) können Sie das gewünschte Objekt (Form, Textfeld etc.) auswählen, die Eigenschaften werden wie gewohnt zugewiesen. Eine Gegenüberstellung der eingedeutschten und der exakten VBA-Bezeichner finden Sie im Übersichtsteil des Kapitels (Seite 68).

1.1.8 Der Projekt-Explorer

Dies ist gewissermaßen die "Schaltzentrale" Ihrer Access-Anwendung. In einer baumartigen Struktur finden Sie alle zu Ihrer Anwendung gehörenden Objekte wie Formulare, Reports, Code- und Klassenmodule übersichtlich aufgelistet. Durch Doppelklick werden das entsprechende Codefenster und das Eigenschaftsfenster geöffnet. Über das Kontextmenü der rechten Maustaste können Sie den Eigenschaftendialog aufrufen, den Quellcode ausdrucken lassen, das Objekt wieder entfernen etc.

1.1.9 Das Code-Fenster

In diesem Fenster, dem VBA-Editor, wird sich Ihr zukünftiges Leben als Programmierer im Wesentlichen abspielen. Der Objektselector (links oben) bzw. der Ereignisselector (rechts oben) leisten im Zusammenhang mit dem Schreiben von Ereignisbehandlungsroutinen (Event-Handlern) wertvolle Hilfsdienste. Konkretes dazu erfahren Sie im VBA-Einführungsbeispiel dieses Kapitels (Seite 50).

Im Folgenden sollen nur einige allgemeine Hinweise zum Schreiben von Quellcode gegeben werden, was ähnlich wie in einem Textverarbeitungsprogramm vonstatten geht (siehe z.B. das Menü *Bearbeiten*). Einige Besonderheiten sind jedoch zu beachten:

Groß- und Kleinschreibung

Die Groß-/Kleinschreibung spielt zwar keine Rolle, trotzdem korrigiert der Code-Editor bereits bei der Eingabe die Schreibweise, indem er Schlüsselwörter mit Großbuchstaben beginnen lässt. Bei selbst definierten Bezeichnern sollten Sie die Groß- und Kleinschreibung sinnvoll dafür einsetzen, um die Lesbarkeit des Listings zu verbessern.

BEISPIEL: Die Bedeutung von

HaushaltKassenSaldo

ist sicherlich eher zu begreifen, als die gleichwertige Schreibweise

haushaltkassensaldo

Mehrfachanweisungen

Normalerweise steht in jeder Zeile eine einzige Anweisung. Um Platz zu sparen, ist es aber möglich, auch mehrere Anweisungen pro Zeile unterzubringen. Als Separator dient ein Doppelpunkt (:).

BEISPIEL: Die beiden Anweisungen:

```
i = i+1
Call addiere(i)
```

können so in einer Zeile zusammengefasst werden:

```
i = i+1 : Call addiere(i)
```

Zeilenumbruch

Die maximale Zeilenlänge des Editors liegt erheblich über der Lesbarkeitsgrenze, sodass man sich bei langen Anweisungen (z.B. Aufruf von API-Funktionen) eine übersichtlichere Darstellung wünscht. Für diesen Zweck existiert ein Zeilenumbruchzeichen, der Unterstrich (_) in Kombination mit einem vorangestellten Leerzeichen.

BEISPIEL: Die Deklaration:

```
Sub verschiebMich(ctrl As Control, x0 As Integer, y0 As Integer, Optional breite)
```

kann wie folgt auf zwei Zeilen aufgeteilt werden:

```
Sub verschiebMich(ctrl As Control, x0 As Integer, y0 As Integer, _
Optional breite)
```

HINWEIS: Achten Sie unbedingt auf das Leerzeichen vor dem Unterstrich!

Innerhalb von Strings (Zeichenketten) ist ein Zeilenumbruch auf die oben beschriebene Weise allerdings nicht möglich. Hier müssen Sie den String vorher in Teilstrings zerlegen, die dann addiert werden.

BEISPIEL: Die Anweisung:

```
Me.RecordSource = "SELECT Name, Ort, Telefon, KontoNr FROM Kunden"
```

lässt sich wie folgt aufteilen:

```
Me.RecordSource = "SELECT Name, Ort, Telefon, KontoNr " & _
"FROM Kunden"
```

Listing strukturieren

Im Interesse der Übersichtlichkeit sollte man die Blockstruktur des Codes deutlich durch Einrückungen hervorheben. Dabei genügt im Allgemeinen ein Versatz nach rechts von einem Zeichen pro Ebene.

BEISPIEL: Ein gut strukturiertes Listing

```
Private Sub Form_Load()
Dim i As Integer
  For i = 0 To Controls.Count - 1
    If Controls(i).ControlType = acCommandButton Then
      Call verschiebe(Controls(i), links+i * breite, breite, hoehe)
```

```
End If
Next i
End Sub
```

HINWEIS: Wenn Sie das Kontrollkästchen *Automatisch Einzug vergrößern* auf der Registerkarte *Editor* (Menü *Extras/ Optionen*) aktivieren, kann der Einrückvorgang automatisiert werden.

Verwendung von Ausrufezeichen, Punkt und eckigen Klammern

Beim Quelltextstudium in diesem Buch wird Ihnen vielleicht auffallen, dass Objektbezeichner von ihren Eigenschaften, Methoden oder untergeordneten Objekten scheinbar willkürlich manchmal durch ein Ausrufezeichen (!) oder durch einen Punkt (.) abgetrennt werden.

BEISPIEL: Verwendung von Ausrufezeichen/Punkt

```
Forms!Kunden.Enabled = True
```

Hinter dieser Vorgehensweise verbirgt sich die Idee, dass der von Ihnen geschriebene Code auch noch dann von Access-Nachfolgeversionen erkannt wird, wenn mit einer Erweiterung des Sprachumfangs neue Bezeichner eingeführt werden, die zufällig namensgleich mit benutzerdefinierten Namen von Objekten, Eigenschaften oder Methoden sind. Um dem in solchen Fällen vorprogrammierten Crash aus dem Wege zu gehen, ist es eine reine Vorsichtsmaßnahme, wenn Sie sich an folgende Regeln halten:

- Verwenden Sie ein Ausrufezeichen (!) vor Objekten, die Sie selbst benannt haben.
- Benutzen Sie einen Punkt (.) vor Eigenschaften und Methoden, deren Namensgebung von Microsoft Access vorgenommen wurde.
- Verwenden Sie eckige Klammern, um die Namen von Objekten einzuschließen, die Leerzeichen enthalten, z.B. *[Personal Table]*, oder verzichten Sie besser vollständig auf Leerzeichen innerhalb von Objektbezeichnern, verwenden Sie z.B. *PersonalTable*.

Kommentare

Sie sollten Kommentare stets reichlich in den Quelltext einbauen. Nicht nur andere Programmierer, die später Ihren Code verstehen bzw. ergänzen sollen, werden Ihnen dafür dankbar sein. Auch Sie selbst werden nach einer mehrmonatigen Unterbrechung weniger Mühe haben, sich wieder in Ihr eigenes Programm einzuarbeiten. Kommentare erleichtern den Wiedereinstieg.

BEISPIEL: Kommentare

```
Dim anzahl As Integer      ' Anzahl von Datensätzen der Tabelle "Personen"
' es folgen weitere Deklarationen: -----
```

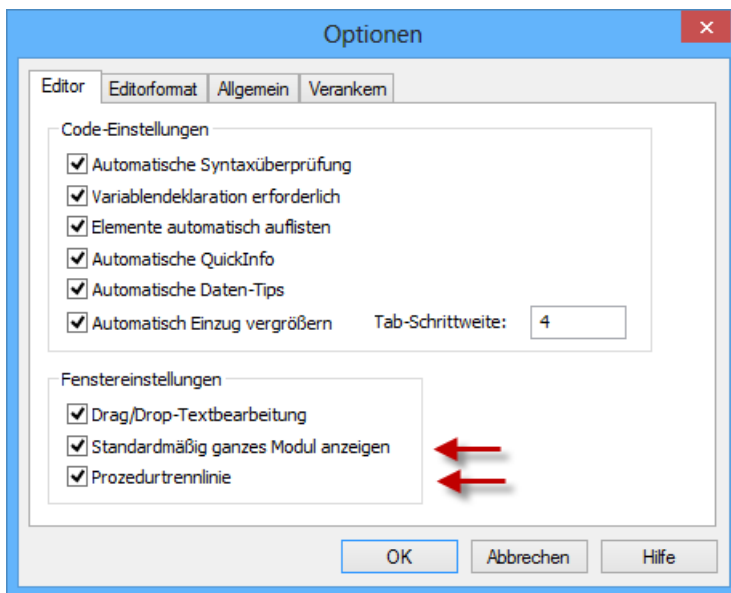
Wie Sie sehen, werden Kommentare durch ein Apostroph (') eingeleitet. Wenn sich der Kommentar, wie oben gezeigt, über mehrere Zeilen erstreckt, so ist trotzdem in jeder Zeile das Apostroph voranzustellen.

Farbige Hervorhebungen

Eine weitere Möglichkeit, die Lesbarkeit Ihres Quelltextes zu verbessern, erschließt sich Ihnen auf den Registerkarten *Editorformat* und *Editor* (Menü *Extras/Optionen*).

Für die verschiedenen Textbereiche (normaler Text, Markierungstext, Syntaxfehlertext, Haltepunkttext, Kommentartext, Schlüsselworttext, Bezeichnertext etc.) können Sie bestimmte Vorder- und Hintergrundfarben wählen.

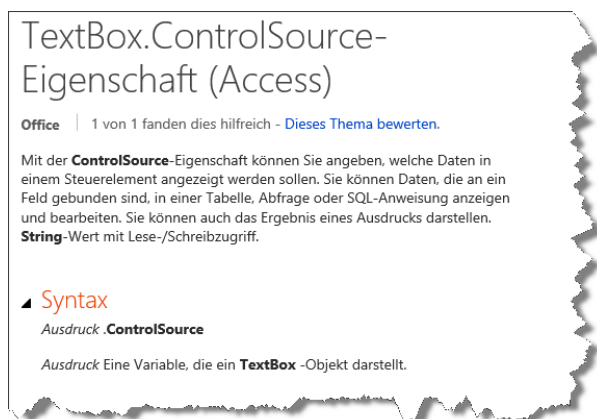
Für die "Fenstereinstellungen" empfiehlt sich insbesondere die Aktivierung der Optionen *Standardmäßig ganzes Modul anzeigen* und *Prozedurtrennlinie*. In diesem Fall erscheint der komplette Quelltext in einem einzigen Fenster, die einzelnen Prozeduren sind durch waagrechte Linien voneinander getrennt.



HINWEIS: Den gleichen Effekt können Sie aber auch erreichen, wenn Sie die Fenstereinstellung mittels der beiden Schaltflächen in der linken unteren Ecke des Codefensters umschalten.

1.1.10 Arbeiten mit der Hilfe

Anstatt ziellos im üppigen Angebot herumzustochern, sollte man als VBA-Programmierer die leistungsfähigen Suchfunktionen nutzen. So kann man z.B. den Textcursor in einen unbekannten VBA-Ausdruck setzen und anschließend die **[F1]**-Taste drücken.



1.2 Sicherheitseinstellungen

Unter Access werden benutzerdefinierte Makros und VBA-Code zunächst als "unsicherer Code" verteuftelt, die Ausführung verweigert. Ganz bewusst haben wir deshalb diesen Abschnitt bereits in dieses erste Kapitel gestellt, da die Gefahr groß ist, dass der Einsteiger sich Hals über Kopf in die Makro- und VBA-Programmierung stürzt, ohne sich vorher das notwendigste Wissen über das doch recht restriktive Sicherheitsmodell von Access angeeignet zu haben.

1.2.1 Zur Geschichte der Access-Sicherheit

Als Antwort auf die wachsende Bedrohung durch Viren und Würmer hat Microsoft bereits 2002 unter dem Namen *Trustworthy Computing* eine Initiative gestartet, um alle Produkte sicherer zu machen.

Bill Gates hat die Ernsthaftigkeit dieser Initiative wie folgt unterstrichen:

"In der Vergangenheit haben wir unsere Software hauptsächlich durch das Hinzufügen neuer Features und Funktionen auf die Bedürfnisse unserer User ausgerichtet, indem wir unsere Plattform reichlich erweiterbar machten. Wir haben da großartige Arbeit geleistet, aber all diese wunderbaren Features sind nichts wert wenn die User unserer Software kein Vertrauen mehr entgegenbringen. Wenn wir aber heute vor der Wahl zwischen dem Hinzufügen neuer Features und der Lösung von Sicherheitsproblemen stehen, so müssen wir uns für die Sicherheit entscheiden. Unsere Produkte müssen nicht nur von Haus aus sicher sein, sondern wir müssen auch unverzüglich auf Sicherheitslücken reagieren, sobald sie sichtbar werden."

In Zeiten vor Access 2003 konnte Ihnen ein "guter Freund" ohne weiteres eine Datenbankdatei schicken, die böartigen Code enthielt. Sobald Sie die Datenbank öffneten konnte dieser Code ausgeführt werden und Ihr System beschädigen. Alternativ konnte der Programmierer gefährlichen Code in eine Abfrage, ein Formular oder einen Report einfügen.

In Version 11 (Access 2003) wurden Sie beim Öffnen einer nicht signierten/zertifizierten Datenbank mit einer Reihe verwirrender Dialoge konfrontiert, wenn Sie die Sicherheitsstufe für Makros auf Medium oder High eingestellt hatten. Nach dem Abarbeiten dieser verschiedenen Dialoge konnte es passieren, dass Sie verzweifelt vor einer Datenbank saßen, die sich nicht mehr öffnen ließ.

Access 2007 hatte das Sicherheitsmodell verbessert, indem zum Access Interface eine Komponente namens "Trust Center" hinzugefügt wurde. Dieses Sicherheits-Interface ist weitaus weniger verwirrend als das "Macro Security"-Feature von Access 2003. Mit der High-Sicherheitsebene in Access 2003 waren Sie nicht in der Lage irgendeine Datenbankdatei zu öffnen, weil potenziell alle Access-Datenbanken irgendwelche Makros, VBA-Code oder Aufrufe an unsichere (unsafe) Funktionen enthalten konnten.

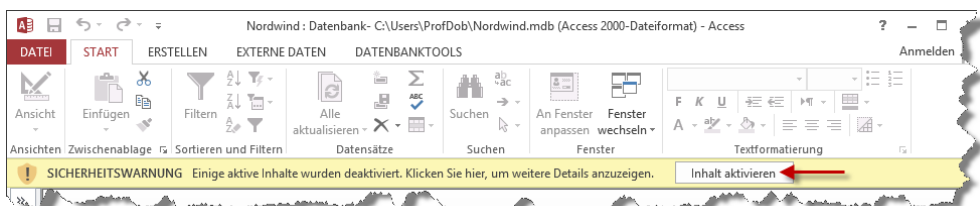
Access 2010 hat das Sicherheitsmodell von Access 2007 durch Hinzufügen von vertrauenswürdigen Dokumenten (*Trusted Documents*) weiter verbessert. Jede Datenbank mit Abfragen wird zunächst als unsicher eingestuft, weil die Abfragen Aufrufe an unsichere Funktionen enthalten können.

Ab Access 2013 wird eine Datenbankdatei geöffnet, ohne dass Sie von nervigen Dialogboxen gequält werden. Sicherheitsmeldungen werden übersichtlich in einer Statusleiste dargestellt. In Abhängigkeit davon, ob sich die Datenbank auf der lokalen Festplatte befindet oder aber für den Netzwerkzugriff freigegeben ist, kann Access auch unbemerkt (also ohne Meldung) alle bösartigen Makros oder VBA Code unterdrücken.

HINWEIS: Alle Beispielanwendungen haben wir so entworfen, dass Sie sie erfolgreich öffnen können, allerdings zeigt jede einen Warndialog an, falls die Datenbank nicht vertrauenswürdig ist. Falls Sie die Datenbank in einem nicht vertrauenswürdigen Verzeichnis gespeichert haben, zeigt Ihnen die Applikation Hinweise an, die Ihnen dabei helfen, die Anwendung vollständig in Gang zu setzen.

1.2.2 Eine nicht vertrauenswürdige Datenbank öffnen

Wenn Sie eine Datenbank oder eine entsprechende Vorlage öffnen, können Sie häufig eine Sicherheitswarnung in der gelben Statusleiste (unterhalb des Menübands) sehen. Diese Meldung besagt, dass Access bestimmte Features deaktiviert hat, weil die Datei nicht digital signiert oder kein vertrauenswürdiges Dokument ist oder aber sich in einem Verzeichnis befindet, welches nicht als vertrauenswürdig gekennzeichnet ist.



Um Makros oder irgendwelchen restriktiven Code in dieser Datenbank zu aktivieren, müssen Sie die Schaltfläche *Inhalt aktivieren* klicken. Access schließt die Datenbank und versucht erneut sie zu öffnen, wobei alle Inhalte aktiviert werden. Danach wird die Datenbank zur Liste der vertrauenswürdigen Dokumente hinzugefügt.

Falls Ihre Datenbank aktuell als nicht vertrauenswürdig eingestuft ist, zeigt Access eine Sicherheitswarnung auf der *Informationen*-Registerkarte der Backstage-Ansicht:



Wenn Sie auf die Schaltfläche *Inhalt aktivieren* klicken, bietet Access zwei Optionen an:

- *Alle Inhalte aktivieren*
Access fügt diese Datenbank zur Liste der vertrauenswürdigen Datenbankdateien hinzu. Jedes Mal, wenn Sie diese Datenbank vom gleichen Speicherort aus öffnen, bleiben alle Inhalte der Datenbank aktiv. Verschieben Sie allerdings die Datenbank in ein anderes Verzeichnis, so werden beim Öffnen der Datenbank deren entsprechende Inhalte deaktiviert.
- *Erweiterte Optionen*
Es öffnet sich der Dialog *Microsoft Office-Sicherheitsoptionen*. Dieser Dialog warnt Sie, wenn der Inhalt der Datei nicht überprüfbar ist, da eine digitale Signatur fehlt.

Sie haben zwei Möglichkeiten:

- *Vor unbekanntem Inhalt schützen*
Wenn Sie bei dieser empfohlenen Standardoption bleiben, müssen Sie sich leider damit abfinden, dass sämtlicher VBA-Code geblockt wird und auch alle Makros mit potenziell gefährlichen Kommandos nicht ausgeführt werden. Damit ist es wahrscheinlich, dass diese Anwendung nicht korrekt funktionieren wird.
- *Inhalt für diese Sitzung aktivieren*
Diese Option bewirkt, dass Access die Datenbank schließt und erneut mit aktivierten Inhalten öffnet. Jetzt läuft sämtlicher VBA-Code und auch alle Makros funktionieren.

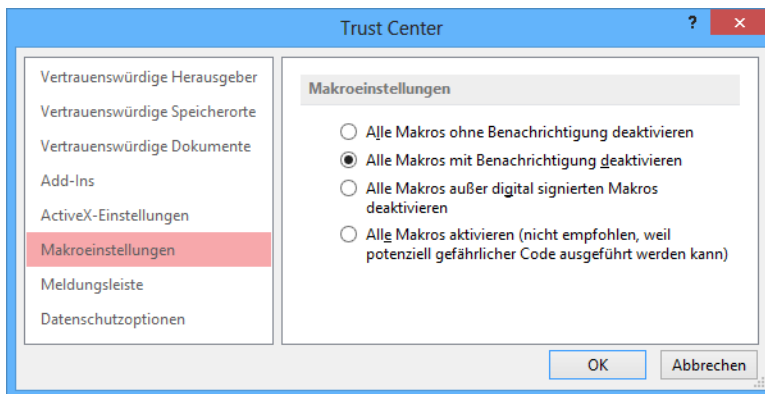
HINWEIS: Wenn Sie den Inhalt nach dem Öffnen einer nicht vertrauenswürdigen Datenbank aktivieren, wird diese nur für die aktuelle Sitzung vertrauenswürdig. Schließen Sie die Datenbank und öffnen Sie sie wieder, so werden Sie erneut mit der Sicherheitswarnung konfrontiert.

1.2.3 Das Sicherheitscenter

In der unteren linken Ecke des Dialogs *Microsoft Office-Sicherheitsoptionen* haben Sie vielleicht schon den Link *Trust Center öffnen* bemerkt. Zum gleichen Ziel kommen Sie aber auch über die *Informationen*-Registerkarte in der Backstage-Ansicht. Hier können Sie über *Optionen* den Dialog *Access-Optionen* öffnen um von dort aus zum Sicherheitscenter zu gelangen. Schneller geht es, wenn Sie innerhalb der Sicherheitswarnung auf den Link *Einstellungen für das Sicherheitscenter* klicken.

Übersicht

Im Dialog *Sicherheitscenter* oder *Trust Center* sehen Sie insgesamt acht Kategorien für diverse Sicherheitseinstellungen.



Vertrauenswürdige Herausgeber

Hier können Sie Herausgeber betrachten oder entfernen, die als vertrauenswürdig eingestuft wurden. Wenn Applikationen von einem dieser Herausgeber digital signiert wurden, wird Access keinerlei Inhalte der Datenbank deaktivieren und die Statusleiste wird keine Warnung melden.

HINWEIS: Standardmäßig sind alle von Microsoft digital signierten Applikationen vertrauenswürdig!

Möglicherweise erscheinen auch noch andere vertrauenswürdige Herausgeber in der Liste, falls Sie signierte Applikationen installiert haben und Windows dabei mitteilen, dass Sie dem Herausgeber vertrauen und dessen Zertifikat speichern wollen.

Vertrauenswürdige Speicherorte

Hier legen Sie spezielle Verzeichnisse und Unterverzeichnisse als vertrauenswürdige Speicherorte fest. Access betrachtet jede Datenbankdatei innerhalb dieser Verzeichnisse als vertrauenswürdige und alle Inhalte sind aktiviert.

Vertrauenswürdige Dokumente

Standardmäßig erlaubt es Access, Datenbankdateien in einer Netzwerkumgebung als vertrauenswürdige einzustufen. Nach Ausschalten dieses Kontrollkästchens deaktiviert Access alle Inhalte, die Sie vorher als vertrauenswürdige gekennzeichnet haben und entfernt alle Datenbankdateien von der internen Liste vertrauenswürdiger Dokumente.

Add-Ins

Ein Add-In ist ein separates Programm (oder Datei) welches die Fähigkeiten von Access erweitert. Sie können ein solches Add-In unter Verwendung von VBA (oder einer anderen Programmiersprache wie z.B. C#) selbst erstellen.

Drei Einstellungen können wahlweise ein-/ausgeschaltet werden:

- *Anwendungs-Add-Ins müssen von einem vertrauenswürdigen Herausgeber signiert sein*
- *Benachrichtigung für nicht signierte Add-Ins deaktivieren*
Der Code bleibt deaktiviert.
- *Alle Anwendungs-Add-Ins deaktivieren*
Dies führt möglicherweise zu Funktionsbeeinträchtigungen.

ActiveX-Einstellungen

Fünf Optionen sind verfügbar, allerdings kann von den ersten vier immer nur eine aktiv sein:

- *Alle Steuerelemente ohne Benachrichtigung deaktivieren*
Access deaktiviert alle verdächtigen ActiveX-Steuerelemente, zeigt allerdings keine Statusleiste an.
- *Eingabeaufforderung anzeigen bevor UFI¹-Steuerelemente mit zusätzlichen Einschränkungen und SFI²-Steuerelemente mit minimalen Einschränkungen aktiviert werden.* Falls ein VBA-Projekt vorliegt, deaktiviert Access alle ActiveX-Steuerelemente und zeigt die Statusleiste an. Ist VBA-Code vorhanden, so aktiviert Access alle SFI- und deaktiviert alle UFI-Steuerelemente, wobei die Statusleiste angezeigt wird. Aktivieren Sie UFI-Inhalte, werden diese zwar initialisiert, allerdings nur eingeschränkt.
- *Eingabeaufforderung anzeigen, bevor alle Steuerelemente mit minimalen Einschränkungen aktiviert werden.* Dies ist die Standardeinstellung nach einer Neuinstallation von Access. Ist

¹ *Unsafe for Initialization*

² *Safe for Initialization*

VBA-Code vorhanden, so deaktiviert Access alle ActiveX-Steuerelemente und zeigt die Statusleiste an. Anderenfalls aktiviert Access die SFI- und deaktiviert die UFI-Steuerelemente und zeigt die Statusleiste an. Aktivieren Sie UFI-Inhalte, werden diese zwar initialisiert, allerdings nur eingeschränkt.

- *Alle Steuerelemente ohne Einschränkungen und ohne Eingabeaufforderung aktivieren*
Diese Einstellung wird nicht empfohlen, weil potenziell gefährliche Steuerelemente eingeführt werden können.
- *Abgesicherter Modus*
Diese Einstellung (Standard) beschränkt den Zugriff des Steuerelements auf Ihren Computer und gestattet das Abspielen von SFI-Steuerelementen im abgesicherten Modus.

Makroeinstellungen

Hier legen Sie fest, wie Access mit Makros umgeht, wenn sich die Datenbank nicht an einem vertrauenswürdigen Speicherplatz befindet. Es stehen vier Alternativen zur Verfügung:

- *Alle Makros ohne Benachrichtigung deaktivieren*
In diesem Fall werden verdächtige Makros zwar deaktiviert, eine Warnung über die Statusleiste erscheint allerdings nicht.
- *Alle Makros mit Benachrichtigung deaktivieren*
Dies ist die Standardeinstellung nach einer Neuinstallation von Access. Die gelbe Statusleiste setzt Sie davon in Kenntnis, dass verdächtige Makros deaktiviert wurden (entspricht mittlerer Sicherheitsebene).
- *Alle Makros außer digital signierten Makros deaktivieren*
Jeder potenziell gefährlicher Code wird geblockt (entspricht höchster Sicherheitsebene).
- *Alle Makros aktivieren*
Diese Einstellung ist nicht zu empfehlen, weil potenziell gefährlicher Code ausgeführt werden kann. Hier können Sie in Teufels Küche kommen, denn Sie erhalten auch keinerlei Warnungen (entspricht niedrigster Sicherheitsebene).

Meldungsleiste

Hier können Sie entscheiden, ob Sie durch die gelbe Statusleiste "belästigt" werden wollen oder nicht. Haben Sie diese abgeschaltet, erhalten Sie keinerlei Warnungen zu gesperrten Inhalten.

Datenschutzoptionen

Hier haben Sie Einfluss auf ein Sammelsurium diverser Access-Aktionen:

- *Office das Herstellen einer Internetverbindung gestatten*
Wenn eine Verbindung mit dem Internet besteht, sucht Access die Hilfeseite von Microsoft auf. Ist das Kontrollkästchen deaktiviert, so wird beim Aufruf der Hilfe nur die lokale Festplatte durchsucht.

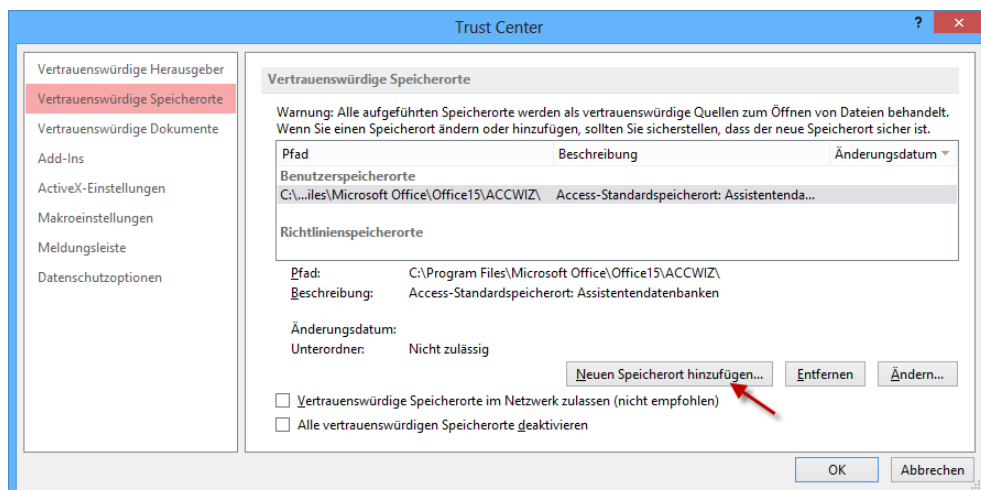
- *Regelmäßig eine Datei herunterladen, mit deren Hilfe Systemprobleme bestimmt werden können*
Access lädt eine spezielle Datei von der Microsoft Website, welche Ihnen bei der Fehlersuche in Access und in anderen Office-Programmen behilflich ist.
- *Beim Programm zur Verbesserung der Benutzerfreundlichkeit anmelden*
Microsoft nutzt dieses Programm für statistische Auswertungen um häufige Fehlerquellen festzustellen und um künftige Access-Versionen zu verbessern.
- *Office Dokumente überprüfen, die von verdächtigen Websites stammen oder dorthin verlinken*
Diese Option ist standardmäßig eingeschaltet um Ihren Computer vor Dokumenten mit böswärtigen Weblinks zu schützen.
- *Dem Aufgabenbereich "Recherchieren" das Prüfen auf neue Dienste und deren Installation erlauben*
Access sucht automatisch nach neuen Updates.

1.2.4 Definition vertrauenswürdiger Speicherorte

Sie können den Inhalt einer nicht vertrauenswürdigen Datenbank dauerhaft aktivieren, wenn Sie auf Ihrer Festplatte (oder im Netzwerk) ein vertrauenswürdiges Verzeichnis anlegen und die Datenbank in dieses Verzeichnis verschieben. Alternativ können Sie auch das aktuelle Datenbankverzeichnis als vertrauenswürdige definieren.

Vorgehensweise

Um einen vertrauenswürdigen Speicherort zu definieren, müssen Sie das Sicherheitscenter öffnen, wofür es verschiedene Möglichkeiten gibt. Wählen Sie die Kategorie *Vertrauenswürdige Speicherorte*:



Klicken Sie dann die Schaltfläche *Neuen Speicherort hinzufügen...* Access zeigt Ihnen nun den Dialog *Vertrauenswürdiger Microsoft Office-Speicherort* an.

Klicken Sie auf die *Durchsuchen...*-Schaltfläche und wählen Sie das Verzeichnis aus welches Sie als vertrauenswürdig einstufen wollen. Sie haben zusätzlich die Möglichkeit, auch alle Unterverzeichnisse mit zu erfassen und einen erklärenden Text einzugeben.

Nach dem *OK* sehen Sie im Sicherheitscenter, dass das neue Verzeichnis zur Liste vertrauenswürdiger Speicherorte hinzugefügt wurde.

HINWEIS: Microsoft empfiehlt, dass Sie keinesfalls das Stammverzeichnis Ihrer Windows-Installation (zum Beispiel *C:* nach einer Standardinstallation) als vertrauenswürdigen Speicherort festlegen sollten!

Markieren Sie nur selbst angelegte Verzeichnisse als vertrauenswürdig. Falls Sie später die Vertrauenswürdigkeit wieder aufheben wollen, so klicken Sie einfach die *Entfernen*-Schaltfläche.

Weitere Optionen

Obige Abbildung zeigt auch zwei Kontrollkästchen am unteren Rand des Dialogs:

- *Vertrauenswürdige Speicherorte im Netzwerk zulassen*
Microsoft empfiehlt, diese Option nicht zu selektieren, weil Sie nicht kontrollieren können, welche Dateien sich in einem Netzwerk befinden
- *Alle vertrauenswürdigen Speicherorte deaktivieren*
Es werden nur Inhalte von vertrauenswürdigen Herausgebern zugelassen

HINWEIS: Um abzusichern, dass alle Beispiele der Begleitdateien korrekt funktionieren, fügen Sie das Verzeichnis in welches Sie die Beispieldatenbanken kopiert haben zu Ihren vertrauenswürdigen Speicherorten hinzu.

1.3 Einführungsbeispiele

Wir wollen ein sehr einfaches Beispiel aus einer Personaldatenbank verwenden. Für Ihren VBA-Einstieg ist es vorteilhaft, wenn wir zunächst die beiden klassischen Programmiermethoden (Makro bzw. Code) einem direkten Vergleich unterziehen. Als dritte Alternative zeigen wir die Realisierung mit Hilfe von Datenmakros.

Bevor Sie aber loslegen und möglicherweise angesichts nervtötender "Sicherheitsmeckereien" gleich frustriert das Handtuch werfen, sollten Sie sich mit der im Vorgängerabschnitt erörterten Sicherheitsproblematik vertraut machen, um Ihre Werkstatt in einen arbeitsfähigen Zustand zu versetzen.

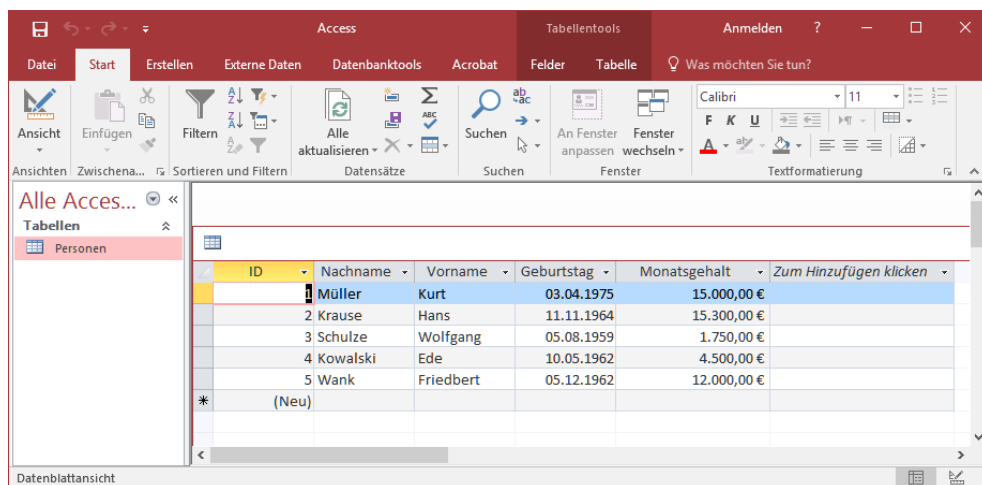
1.3.1 Erstellen der Testdatenbank

Grundlage ist eine Datenbank mit einer einzigen Tabelle *Personen*:

Feldname	Felddatentyp
ID	AutoWert
Nachname	Text
Vorname	Text
Geburtsdatum	Datum/Uhrzeit
Monatsgehalt	Währung

Obwohl wir in diesem Buch voraussetzen, dass Sie bereits über einige Access-Erfahrungen verfügen, soll hier noch einmal – mit Rücksicht auf Quereinsteiger – ganz kurz ein möglicher Weg zum Erstellen obiger Tabelle beschrieben werden:

- Starten Sie Microsoft Access und wählen Sie die Vorlage *Leere Desktopdatenbank*
- Fügen Sie vier Spalten/Felder hinzu
- Ändern Sie Feldnamen und Felddatentypen entsprechend obiger Tabelle
- Geben Sie einige Datensätze ein, also Nachname, Vorname etc., und schließen Sie das Fenster wieder
- Speichern Sie die Tabelle unter dem Namen *Personen* und schließen Sie Access
- Verschieben Sie die Datenbank aus ihrem Standardverzeichnis an einen vorher von Ihnen angelegten vertrauenswürdigen Speicherort (siehe Seite 46) und benennen Sie die Datei eventuell um (z.B. in *Test.accdb*)
- Nach dem erneuten Öffnen der Datenbank sollte sich Ihnen etwa die in der folgenden Abbildung gezeigte Datenblattansicht bieten:



1.3.2 Konventionelle Programmierung

Jeder, der bereits unter Microsoft Access individuelle Formulare gestaltet hat, kennt die vier Etappen des Entwurfsprozesses:

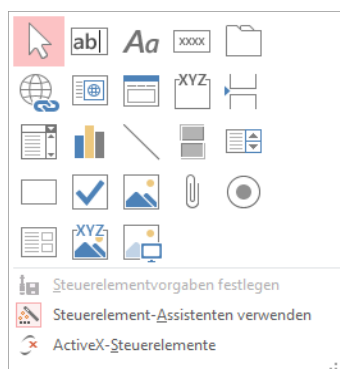
- Visuelle Gestaltung der Bedienoberfläche
- Zuweisen der Objekteigenschaften
- Verknüpfen der Objekte mit Ereignissen
- Programmtest

Erste Etappe: Visuelle Formulargestaltung

Am einfachsten ist es natürlich, wenn man sich eine komplette Eingabemaske für die *Personen*-Tabelle durch Klick auf das *Formular*-Symbol im Menüband (Registerkarte *Erstellen*, Befehlsgruppe *Formulare*) automatisch generieren lässt.

Da wir uns aber mit dieser standardmäßigen Bedienoberfläche nicht zufrieden geben wollen, müssen wir selbst Hand anlegen und zunächst ein neues leeres Formular erzeugen (Symbol *Formularentwurf*), welches wir dann nach eigenem Ermessen mit sechs *Textfeldern*, mehreren *Bezeichnungsfeldern* und einer *Befehlsschaltfläche* bestücken:

Der Access-Kundige wird keinerlei Mühe mit dem "Zusammenschieben" des Formulars haben. Die einzelnen Steuerelemente werden der Befehlsgruppe *Steuerelemente* (Hauptregisterkarte *Entwurf*) entnommen, mit der Maus an der gewünschten Stelle im Detailbereich des Formulars platziert und gegebenenfalls in ihrer Größe verändert.



Zweite Etappe: Objekteigenschaften zuweisen

Im Eigenschaftenblatt stellen Sie nur die von den Standardeinstellungen abweichenden Werte für das Formular und die Steuerelemente ein:

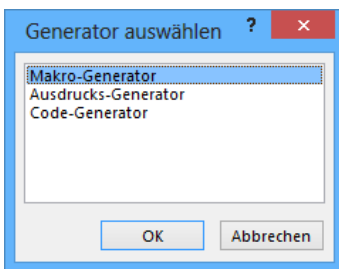
Objekt-Name	Eigenschaft	Wert
<i>Form1</i>	Datensatzquelle	Personen
	Beschriftung	Einführungsbeispiel
	Bildlaufleisten	Nein
	Datensatzmarkierer	Nein
	Trennlinien	Nein
<i>Text0</i>	Steuerelementinhalt	Nachname
<i>Text1</i>	Steuerelementinhalt	Vorname
<i>Text2</i>	Steuerelementinhalt	Geburtsdatum
<i>Text2</i>	Format	Datum, lang
<i>Text3</i>	Steuerelementinhalt	Monatsgehalt
<i>Text3</i>	Format	Euro
<i>Text4</i>	Steuerelementinhalt	=Jahr(Datum())-Jahr([Geburstag])
<i>Text5</i>	Steuerelementinhalt	=12*[Monatsgehalt]
	Format	Euro
<i>Befehl0</i>	Beschriftung	Beenden

HINWEIS: Normalerweise wird die *Name*-Eigenschaft der Steuerelemente automatisch beim Entwurf vergeben und entspricht der Reihenfolge, in der die Objekte in den Detailbereich des Formulars eingefügt wurden. Bei der herkömmlichen Makro-Programmierung braucht man sich deshalb um diese Eigenschaft nicht besonders zu kümmern. Im Hinblick auf eine spätere VBA-Programmierung sollten wir trotzdem die *Name*-Eigenschaft der Textfelder und der Befehlsschaltfläche bereits jetzt (entsprechend der linken Spalte in obiger Tabelle) überprüfen und gegebenenfalls anpassen.

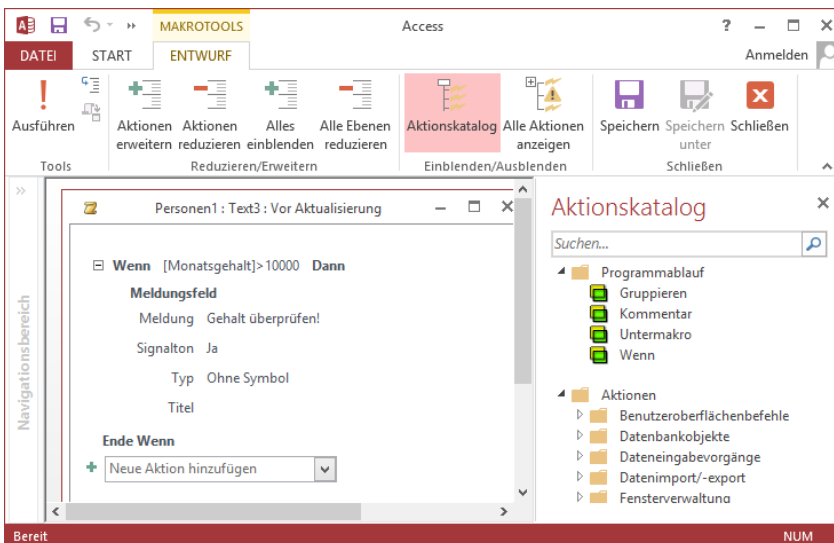
Dritte Etappe: Ereignisse besetzen

Es soll in unserem Beispiel eine Warnung ausgegeben werden, wenn für Personen ein Monatsgehalt von mehr als 10.000 Euro eingegeben wurde. Für den routinierten Makroprogrammierer dürfte die Umsetzung kein Problem darstellen:

- Klicken Sie mit der rechten Maustaste auf das Textfeld *Monatsgehalt*
- Im Kontextmenü wählen Sie den Eintrag *Eigenschaften*
- Im Eigenschaftenblatt wählen Sie die Registerkarte *Ereignis*
- Klicken Sie auf die Zeile des Ereignisses *Vor Aktualisierung* und anschließend auf die Schaltfläche mit den drei Pünktchen.
- Das sich öffnende Dialogfeld ist charakteristisch für die "Dreifach-Weggabelung", an welcher der Access-Programmierer Farbe bekennen muss: "Will ich mit Makros arbeiten, oder möchte ich in die Regionen der VBA-Programmierung aufsteigen?"



- Nun müssen Sie zunächst im Aktionskatalog des sich öffnenden Makro-Entwurfsfensters unter dem Knoten *Programmablauf* die *Wenn*-Bedingung auswählen und spezifizieren.



- Dann öffnen Sie die Kombinationsfeld *Neue Aktion hinzufügen*, wählen die Aktion *Meldungsfeld* aus und tragen den Meldungstext ein.
- Schließen Sie die Makro-Entwurfsansicht und speichern Sie dabei alle vorgenommenen Änderungen ab.
- Eine "Kleinigkeit" fehlt noch: Nach dem Klick auf die *Beenden*-Befehlsschaltfläche soll das Formular geschlossen werden. Wir erstellen dazu ein weiteres (eingebettetes) Makro, welches dem Ereignis *Beim Klicken* der Befehlsschaltfläche die Aktion *FensterSchließen* zuordnet.

Vierte Etappe: Programmtest

Dazu gibt es nicht viel zu sagen, öffnen Sie das Formular und blättern Sie durch die Datensätze.

HINWEIS: Die Warnung *Gehalt überprüfen* erscheint nicht beim normalen Durchblättern, sondern nur unmittelbar nachdem Sie eine neue Person mit zu hohem Gehalt eingefügt bzw. das Gehalt einer vorhandenen Person über das vorgegebene Limit erhöht haben.

The screenshot shows a Microsoft Access form titled 'Einführungsbeispiel (mit Makros)' with the main title 'Personen'. The form is divided into two columns: 'Tabellenwerte' (Table values) and 'Berechnete Werte' (Calculated values). Under 'Tabellenwerte', there are input fields for 'Nachname' (Krause), 'Vorname' (Hans), 'Geburtsdag' (Mittwoch, 11. November 1964), and 'Monatsgehalt' (15.300,00 €). Under 'Berechnete Werte', there are input fields for 'Alter' (49) and 'Jahresgehalt' (183600). A 'Beenden' button is located at the bottom right of the form. A warning dialog box titled 'Microsoft Access' is open in the foreground, displaying the message 'Gehalt überprüfen!' (Check salary!) and an 'OK' button. The status bar at the bottom of the form indicates 'Datensatz: 2 von 5'.

Hinweise für den Einsteiger

- Um das Eigenschaftenblatt für ein bestimmtes Objekt (Formular, Textfeld etc.) zu öffnen, müssen Sie das Objekt erst aktivieren (mit der Maus anklicken). Ansonsten hilft das mit der rechten Maustaste zu öffnende Kontextmenü *Eigenschaften*.

- Die linken vier Textfelder des Formulars wurden direkt an die entsprechenden Felder der *Personen*-Tabelle "angebunden", während die beiden rechten Textfelder die Resultate von Berechnungen beinhalten. Die dazu notwendigen Formeln können direkt oder über den Ausdrucks-Editor eingegeben werden.
- Das Alter einer Person wird in unserem Beispiel nur vereinfacht berechnet (Differenz aus aktuellem Jahr und Geburtsjahr) und ist nur dann exakt, wenn die Person am aktuellen Datum bereits Geburtstag hatte (ansonsten wird ein Jahr zu viel ermittelt).
- Da die klassischen Makros in der Regel an die Benutzerschnittstelle (UI bzw. *User Interface*) gekoppelt sind, werden sie im Folgenden auch als *UI-Makros* bezeichnet.
- Wer die Grundlagen der Makro-Programmierung noch nicht kennt, der sei auf das Kapitel "Einführung in die Makro-Programmierung" verwiesen.

1.3.3 Programmieren mit VBA

Nun wollen wir die UI-Makros "außen vor" lassen und das Ganze mit VBA realisieren.

HINWEIS: Der Umsteiger sollte sich zunächst an die englischen Bezeichner gewöhnen, die nun den Platz der lokalisierten deutschen Schlüsselwörter einnehmen (siehe Seite 68).

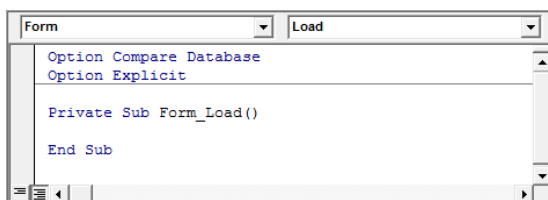
Erste Etappe: Visueller Entwurf der Benutzeroberfläche

Wir empfehlen Ihnen, ein neues Formular anzulegen. Den Formularentwurf nehmen Sie genauso wie beim vorhergehenden Beispiel vor, kümmern sich aber im Folgenden nicht weiter um das Eigenschaftsfenster und um das "Anbinden" der Textfelder an die Tabelle, denn all dies kann nun per VBA-Code erledigt werden. Haben Sie den Detailbereich mit allen notwendigen Steuerelementen bestückt, sollten Sie im Eigenschaftsblatt lediglich die *Name*-Zuweisungen (*Text0*, *Text1* etc.) kontrollieren und gegebenenfalls ändern.

HINWEIS: Um Zeit zu sparen, können Sie das Formular des Vorgängerbeispiels kopieren und unter neuem Namen einfügen. Löschen Sie aber dann alle bereits voreingestellten Eigenschaften.

Zweite Etappe: Zuweisen der Objekteigenschaften

Öffnen Sie den VBA-Editor (Code-Generator) über  oder durch Anklicken des entsprechenden Symbols  der Hauptregistertkarte *Entwurf*:



Die bereits vorhandenen Anweisungen sollen kurz erklärt werden (mehr dazu in Kapitel 2):

<i>Option Compare Database</i>	Der Zeichenkettenvergleich wird durch die Gebietskennung der Datenbank bestimmt.
<i>Option Explicit</i>	Alle Variablen müssen vor ihrer Verwendung deklariert werden.
<i>Private Sub Form_Load()</i> <i>End Sub</i>	Der Rahmencode für das <i>Load</i> -Event (entspricht dem Ereignis <i>Beim Laden</i>) des Formulars.

HINWEIS: Falls *Option Explicit* fehlt, ergänzen Sie diese Anweisung zunächst per Hand. Damit das später automatisch passiert, öffnen Sie (bei geöffnetem Codefenster) über das Menü *Extras/Optionen...* die Registerkarte *Editor* und setzen das Häkchen bei *Variablen Deklaration erforderlich*.

Tippen Sie folgenden Ereignisbehandlungscode für das *Load*-Event ein:

```
Private Sub Form_Load()           ' Beim Laden
    Me.Caption = "Einführungsbeispiel in VBA" ' Beschriftung
    Me.RecordSource = "Personen"   ' Datensatzquelle
    Me.ScrollBars = 0             ' Bildlaufleisten
    Me.RecordSelectors = False     ' Datensatzmarkierer
    Me.DividingLines = False      ' Trennlinien
    Text0.ControlSource = "Nachname" ' Steuerelementinhalt
    Text1.ControlSource = "Vorname"  ' dto.
    Text2.ControlSource = "Geburtsdatum" ' dto.
    Text2.Format = "d/m/yyyy"       ' Format-Eigenschaft
    Text3.ControlSource = "Monatsgehalt" ' Steuerelementinhalt
    Text3.Format = "Currency"       ' Format
    Text4.ControlSource = "=alterBerechnen([Geburtsdatum])" ' Steuerelementinhalt (VBA-Funktion)
    Text5.ControlSource = "=12*[Monatsgehalt]" ' Steuerelementinhalt (einfache Berechnung)
    Text5.Format = "Currency"       ' Format
End Sub
```

Wie Sie erkennen können, haben wir soeben alle Eigenschaften, die bei der herkömmlichen Makro-Programmierungsmethode noch direkt im Eigenschaftenfenster gesetzt wurden, mittels VBA-Code zugewiesen. Dabei dient der Punkt (.) als Separator zwischen Objekt und Eigenschaft. Wie Sie sehen, sind hier nur noch die englischen Bezeichner erlaubt.

HINWEIS: Gewissermaßen als Vorgriff auf Kapitel 4 sei erwähnt, dass der Objektbezeichner *Me* stellvertretend für das aktuelle Formular (*Form*-Objekt) steht.

Im Vorgängerbeispiel war die Altersbestimmung leider nur dann richtig, wenn die betreffende Person im aktuellen Jahr bereits Geburtstag hatte. Wir korrigieren das mit einer "selbst gestrickten" VBA-Funktion, der wir den Namen *alterBerechnen* geben und die wir anschließend eintippen:

```
Public Function alterBerechnen(gebDatum As Date) As Integer
    alterBerechnen = Year(Date) - Year(gebDatum)
```

```

    If Date < DateSerial(Year(Date), Month(gebDatum), Day(gebDatum)) Then
        alterBerechnen = alterBerechnen - 1      ' Person hatte in diesem Jahr noch nicht
Geburtstag!
    End If
End Function

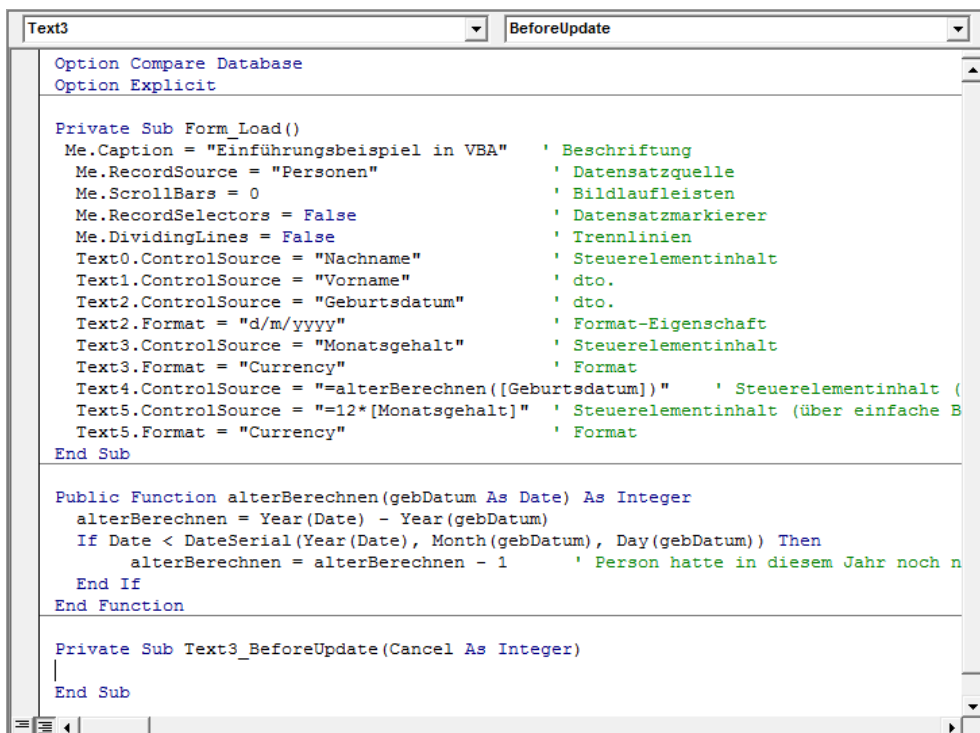
```

Dritte Etappe: Besetzen der Ereignisse

Wenn der konventionelle Access-Programmierer von der "Zuweisung von Ereignisseigenschaften" spricht (bekanntlich sind diese im Eigenschaftenblatt aufgeführt), redet der VBA-Kundige stattdessen vom "Programmieren einer Ereignisbehandlungsroutine" bzw. der "Besetzung eines Event-Handlers".

Bereits in der zweiten Etappe hatten wir die Starteigenschaften im *Load*-Event-Handler zugewiesen. Ähnlich ist die Vorgehensweise bei den folgenden zwei Event-Handlern (Reihenfolge ohne Bedeutung):

Um Monatsgehälter größer *10.000 Euro* vor ihrer Übernahme in die Datenbank zu überprüfen, werten Sie das *BeforeUpdate*-Event aus (entspricht *Vor Aktualisierung*-Ereignis). Dazu wählen Sie zuerst im Objektselektor (oben links) das Steuerelement *Text3* aus und anschließend im rechts daneben befindlichen Ereignisselektor das *BeforeUpdate*-Ereignis.



Auch hier wird ein vorgefertigter Prozedurrumpf bereitgestellt, Sie brauchen also nur noch die beiden mittleren Zeilen einzugeben:

```
Private Sub Text3_BeforeUpdate(Cancel As Integer)      ' Vor Aktualisierung
    If Text3.Value > 10000 Then
        Beep
        MsgBox "Gehalt überprüfen!"
    End If
End Sub
```

Für das Schließen des Formulars soll die *Close*-Methode des *DoCmd*-Objekts aufgerufen werden. Die Vorgehensweise für das automatische Erstellen des Rahmencodes ist analog: Im Objektfenster stellen Sie das Objekt *Befehl0* ein und im Ereignisfenster das *Click*-Ereignis.

```
Private Sub Befehl0_Click()      ' Beim Klicken
    DoCmd.Close
End Sub
```

Es sei vorweggenommen, dass das *DoCmd*-Objekt dem erfahrenen Makro-Programmierer eine "Goldene Brücke" baut, indem es die meisten Makro-Aktionen als Methoden zur Verfügung stellt (siehe Übersicht Seite 70). Analog den Eigenschaften werden auch die Methoden durch einen Punkt (.) vom Objektbezeichner abgetrennt.

Vierte Etappe: Programmtest

Obwohl Sie dieses Formular, genauso wie seinen mit Makros programmierten Vorgänger, sofort aus dem Navigationsbereich heraus starten könnten, empfiehlt sich eine andere Vorgehensweise: Wählen Sie vom Codefenster aus den Menüpunkt *Debuggen/Kompilieren von Test*. So erhalten Sie, falls erforderlich, detaillierte Fehlermeldungen des Compilers. Erst wenn alle Fehler im Quelltext beseitigt sind, sollten Sie das Formular wie gewohnt öffnen.

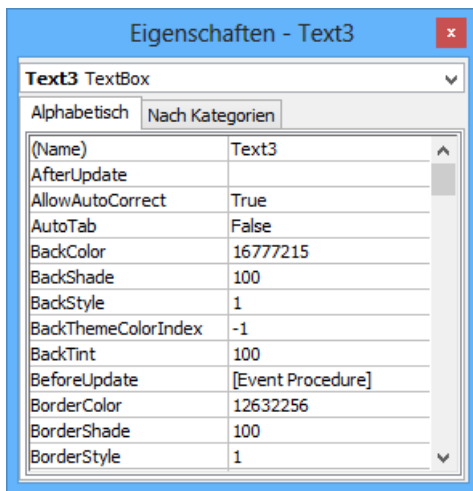
Das Formular sollte das gleiche Verhalten wie im Vorgängerbeispiel zeigen, allerdings wird diesmal (VBA sei Dank!) das Alter der Personen richtig berechnet.

Bemerkungen zum Eigenschaftfenster

Das Zuweisen der Starteigenschaften¹ haben wir innerhalb des *Load*-Event-Handlers, d.h. "per Code", vorgenommen (siehe *Erste Etappe*). Dies bedeutet für uns zwar zusätzliche Tipparbeit, erhöht aber die Übersichtlichkeit des Programms, da man sofort sieht, welche Eigenschaften wie geändert wurden.

Alternativ haben wir natürlich nach wie vor auch die Möglichkeit, die Eigenschaften direkt im Eigenschaftsblatt zuzuweisen. Wenn wir aber (bei geöffnetem Codefenster!) den Menüpunkt *Ansicht/Eigenschaftfenster* (F4) wählen, erscheint dieses Fenster in einer für den Makro-Programmierer bislang ungewohnten Aufmachung: Anstatt der deutschen sind nur noch die originalen, d.h. englischen, Bezeichner vorhanden (siehe folgende Abbildung)!

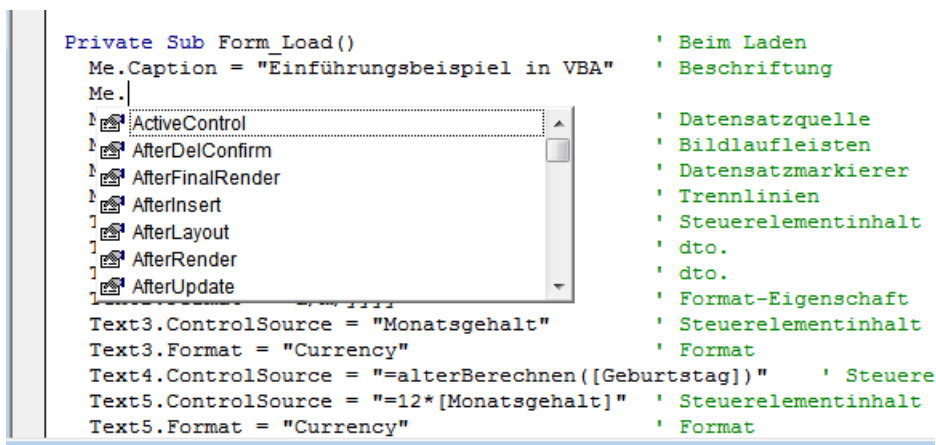
¹ Der Begriff "Starteigenschaften" wurde bewusst gewählt, um zu verdeutlichen, dass diese Eigenschaften später, d.h. zur Programmausführung, per Code wieder verändert werden können.



HINWEIS: Im oberen Kombinationsfeld des Eigenschaftensfensters wählen Sie das Objekt aus, dessen Eigenschaften Sie anzeigen bzw. verändern wollen.

Bemerkungen zur IntelliSense

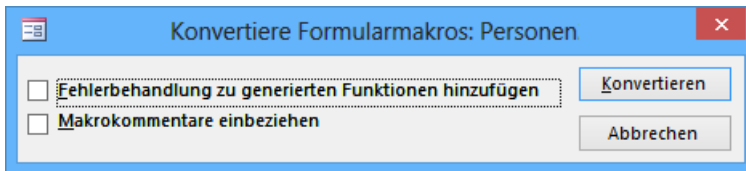
Beim Eintippen des VBA-Codes werden Sie mit einem erstaunlichen Feature des Code-Editors konfrontiert: Kaum haben Sie einen bestimmten Objektbezeichner (z.B. *Me*) eingegeben, so erscheint "wie von Geisterhand" ein Popup-Menü, welches alle in Frage kommenden Eigenschaften und Methoden in alphabetischer Reihenfolge auflistet. Doppelklicken Sie auf einen Eintrag, so wird Ihnen nicht nur die Schreibarbeit abgenommen, sondern Sie ersparen sich auch das umständliche Nachschlagen in der Online-Hilfe.



1.3.4 Automatische Makrokonvertierung

Dieses Feature ermöglicht die schnelle Umstellung von Makro- auf VBA-Programmierung. Wie es funktioniert, lässt sich am besten anhand des Vorgängerbeispiels erklären, welches bereits mit zwei bescheidenen Makros ausgestattet ist.

1. Öffnen Sie dieses Formular in der Entwurfsansicht, klicken Sie die Hauptregisterkarte *Entwurf* und wählen Sie auf dem Menüband ganz rechts das Symbol *Makros des Formulars zu Visual Basic konvertieren*. Es erscheint folgendes Dialogfeld:



2. Um sich auf das Wesentliche zu konzentrieren, deaktivieren Sie die Optionen *Fehlerbehandlung ...* und *Makrokommentare ...* und klicken auf die Schaltfläche *Konvertieren*. Nach kurzer Wartezeit erscheint die Erfolgsmeldung *Makrokonvertierung abgeschlossen*.
3. Das Ergebnis können Sie nun im (vorher leeren) Codefenster des Formulars bewundern:

```
Option Explicit

Private Sub Text3_BeforeUpdate(Cancel As Integer)
    If (Monatsgehalt > 10000) Then
        Beep
        MsgBox "Gehalt überprüfen!", vbOKOnly, ""
    End If
End Sub

Private Sub Befehl0_Click()
    DoCmd.Close , ""
End Sub
```

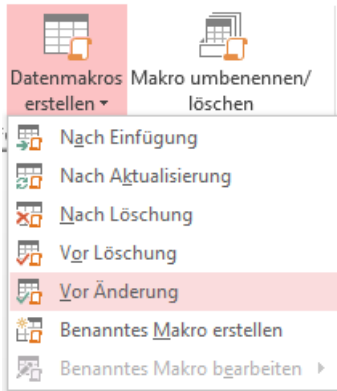
Beim Vergleich mit dem Original-Quellcode stellen Sie bezüglich der beiden Ereignisbehandlungsroutinen eine (fast) 100%-ige Übereinstimmung fest.

HINWEIS: Wie nicht anders zu erwarten, können mittels Makrokonverter nur die vorhandenen Makros in Code umgewandelt werden. Beispielsweise bleiben die Eigenschaftenzuweisungen von der Konvertierung unberührt.

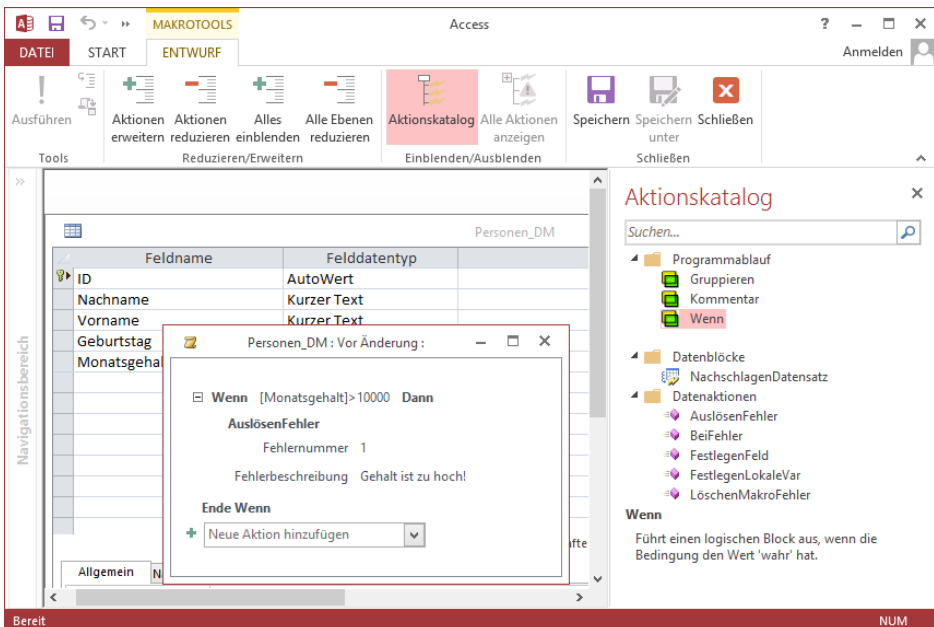
1.3.5 Programmieren mit Datenmakros

Als letzte Alternative wollen wir unser Problem mit einem Datenmakro lösen. Datenmakros wirken direkt auf Tabellenebene.

1. Öffnen Sie die Tabelle *Personen* in der Entwurfsansicht und klicken Sie auf der Registerkarte *Entwurf* das Symbol *Datenmakros erstellen*. Wählen Sie das Ereignis *Vor Änderung*:



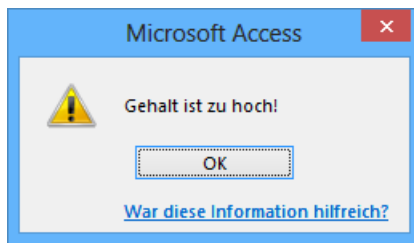
2. Es öffnet sich der Ihnen bereits bekannte Makro-Editor (siehe Seite 52), seine Bedienung entspricht der bei herkömmlichen UI-Makros, das Angebot an Aktionen ist allerdings deutlich geringer.



3. Klicken Sie im Aktionskatalog zunächst die *Wenn*-Bedingung und tragen Sie dort den Ausdruck *[Monatsgehalt]>10000* ein.
4. Wählen Sie dann die Aktion *AuslösenFehler* und tragen Sie beispielsweise die *Fehlernummer* 1 und die *Fehlerbeschreibung* "Gehalt ist zu hoch!" ein.

Da ein Datenmakro nur mit einer Tabelle (nicht mit einem Formular!) verknüpft ist, können Sie es innerhalb der Tabelle testen oder aber ein beliebiges Formular nehmen, das mit dieser Tabelle verbunden ist, beispielsweise das Formular des ersten Einführungsbeispiels (löschen Sie dort lediglich in der *Ereignis*-Registerkarte des Eigenschaftenblatts von *Text3* das bereits vorhandene und zum Ereignis *Vor Aktualisierung* gehörende eingebettete UI-Makro).

Bei Überschreitung des Monatsgehalts erscheint die Meldung des Datenmakros:



Man erkennt in diesem Beispiel zumindest einen wesentlichen Vorteil der Datenmakros gegenüber den klassischen (UI-)Makros:

HINWEIS: Ohne Mehraufwand kann der Programmierer mit einem einzigen Datenmakro beliebig viele Formulare an ein Tabellenereignis binden. Die herkömmliche Makroprogrammierung verlangt hingegen pro Formular ein eigenes Makro.

Aber es gibt noch einen weiteren nicht zu unterschätzenden Vorteil von Datenmakros, den Sie dann erkennen, wenn Sie die Datenblattansicht der *Personal*-Tabelle öffnen: Auch hier erscheint obiges Meldungsfenster, falls Sie versuchen, ein unzulässiges Monatsgehalt direkt in die Tabelle einzugeben. Auf diese Weise wird verhindert, dass von trickreichen Anwendern unter Umgehung des Eingabeformulars direkt an bestimmten Tabelleninhalten "herumgebastelt" wird, um Eingabebeschränkungen außer Kraft zu setzen.

HINWEIS: Mehr zur Programmierung von Datenmakros erfahren Sie im Kapitel 3 "Makros – Eine Einführung".

1.4 Highlights und Features von Access 2016

Weniger für den Einsteiger, sondern mehr für den erfahrenen Programmierer sind die folgenden Ausführungen zu den relativ bescheidenen Neuerungen von Access 2016 gedacht. Um diese richtig einordnen zu können, sei ein kurzer Ausflug in die Vergangenheit vorangestellt.

1.4.1 Zur Geschichte der Vorgängerversionen

Hier die wichtigsten Merkmale der einzelnen Entwicklungsetappen:

Microsoft Access 95

Der Umstieg von der 16- zur 32-Bit-Programmierung, wie er mit der Einführung von Microsoft Access 95 realisiert wurde, brachte folgende Veränderungen:

- Neue Objekte (z.B. *DoCmd*)
- Neue Sprachelemente (z.B. *With ... End With*; *For Each ... Next*)
- Erweiterte vordefinierte mathematische Funktionen
- Access 7.0 konnte als OLE-Server verwendet werden; Unterstützung der OLE-Automation
- Eine Prozedur oder eine Variable kann öffentlich oder privat sein
- Formularmodule und Berichtsmodule sind Klassenmodule

Microsoft Access 97

Neben der 3.5-Version der Jet-Datenbank-Engine und einer Verbesserung der Kompilierungsleistung waren die wichtigsten Neuerungen:

- Erstellen benutzerdefinierter Objekte mit eigenständigen Klassenmodulen
- Office 97-Objektmodell zum Programmieren von Menü- und Symbolleisten
- Neue ActiveX-Komponenten
- Festlegen von Verweisen (Referenzen) auf Programmbibliotheken
- Zugriff auf das Internet/Intranet

Microsoft Access 2000

Die Flut der Neuerungen in den Bereichen Datenbankzugriff, Internet- und Intranetfähigkeit, Installation und Wartung sowie die Internationalität machten aus Access 2000 ein zukunftsweisendes Produkt.

- Als neues Objekt wurde die *Datenzugriffsseite* eingeführt, mit der sich eine Webseite quasi wie ein Formular gestalten lässt.
- VBA 6.0 entsprach dem in Visual Basic integrierten VBA.

- Mit VBA konnten Sie nun selbst Add-Ins schreiben, die zu allen Office-Anwendungen kompatibel sind.
- Die *Jet Engine 4.0* unterstützte nun endlich auch Unicode (2 Byte pro Zeichen). Damit wurde es möglich, beliebige Schriftzeichen in einer Access-Datenbank zu speichern.
- Unter Verwendung eines *Access-Projektes (.adp)* ließen sich Anwendungen entwickeln, die vollständig kompatibel zum *SQL-Server 7.0* waren.
- Mit der *MSDE (Microsoft Data Engine)* wurde eine frei verfügbare, aber etwas eingeschränkte Runtime-Version des Microsoft SQL-Servers 7.0 zur Verfügung gestellt.
- Das wesentliche neue Feature bei den Datenzugriffsobjekten hieß *ADO (ActiveX Data Objects)*, es erlaubt einen Direktzugriff auf ODBC-Datenquellen und Serverdatenbanken nach einem einheitlichen, auf OLE DB basierenden Modell.

Microsoft Access 2002

Ein Highlight dieser Version war zweifelsohne die umfassende XML¹-Unterstützung. Sie konnten nun die vertraute Benutzeroberfläche von Access verwenden, um problemlos XML-Dokumente aus Jet- oder SQL Server-Datenbanken zu erstellen. Außerdem konnten Sie XML-Daten aus anderen Anwendungen in Ihren Formularen, Berichten und Datenzugriffsseiten verwenden.

Weitere Neuerungen:

- die Weiterentwicklung der Datenzugriffsseiten,
- das verbesserte Ereignismodell (z.B. Batchaktualisierungen für Formulare),
- das lang ersehnte *Printer*-Objekt,
- Ersatz der MSDE durch die *SQL Server 2000 Desktop Engine*, damit wird Access von Microsoft zum strategischen Frontend für den SQL Server erklärt!

Microsoft Access 2003

Zu den erwähnenswerten Neuigkeiten zählten Informationen über Abhängigkeiten zwischen den Datenbankobjekten, eine automatische Fehlerüberprüfung bei Formularen und Berichten, die Weitergabe von Feldeigenschaften an gebundene Steuerelemente, das automatische Erstellen einer Sicherheitskopie der Datenbank, sowie eine verbesserte XML-Unterstützung durch Einführung von Transformationsdateien.

Weitere Neuerungen:

- Möglich wurde das Importieren, Exportieren und Verknüpfen von Daten aus Access mit einer Microsoft Windows SharePointServices-Liste sowie das Erstellen lokaler Tabellen aus verknüpften Tabellen.

¹ Extensible Markup Language

- Sie konnten – auch unter Auswertung digitaler Signaturen – verschiedene Sicherheitsstufen für Makros bzw. VBA-Code festlegen. Das Blockieren potenziell unsicherer Informationen wurde möglich.
- Access 2003 brachte kein neues Datenbankformat, sondern verwendete das von Access 2002 (Standardformat blieb das von Access 2000).

Microsoft Access 2007

Diese Version enthielt mehr Neuerungen und Änderungen als die beiden Vorgängerversionen 2002 und 2003 zusammengekommen. Hier nur die wichtigsten:

- Access 2007 verwendete ein neues Dateiformat (*.accdb) und ermöglichte damit neue Features wie mehrwertige Nachschlagefelder (MVF = Multi Value Field), Anlagen-Felder (Attachments), klassische Rich Text-Formatierungen (eigentlich HTML) und interaktive Zusammenarbeit mit dem Microsoft SharePoint-Server.
- Verschiedene alteingesessene Konzepte und Tools waren im neuen ACCDB-Format nicht mehr zu finden, so hatte auch das klassische Benutzer-Sicherheitssystem (.mdw-Datei, Benutzer, Gruppen, Passwörter, ...) ausgedient.
- Das *Attachment*-Objekt dient der Anzeige des neuen *Anlage*-Datentyps. Auch der Zugriff auf mehrwertige Felder wurde möglich.
- Die Datenbank-Replikation und auch die Data Access Pages (DAP) verschwanden still und heimlich von der Bildfläche. Als Alternative wurde SharePoint angeboten.
- Die *TempVars*-Collection wurde eingeführt, sie ermöglichte u.a. auch einen Informationsaustausch zwischen Makros und VBA-Code¹ per globaler Variablen.

Microsoft Office Access 2010

Wieder einmal wurde die Benutzeroberfläche grundlegend überarbeitet. Die vor kurzem von Microsoft noch als "Ei des Kolumbus" gepriesene runde Office-Schaltfläche wurde durch die Registerkarte *Datei* bzw. die dahinter liegende *Backstage*-Ansicht ersetzt.

Unter der Bezeichnung "Datenmakros" wurden endlich die von SQL-Datenbanken her bekannten Trigger eingeführt. Weitere wichtige Änderungen:

- Quasi wie ein Felddatentyp können berechnete Felder definiert werden.
- Neue und überarbeitete Controls (*Webbrowser*-Steuerelement, *Navigationssteuerelement*, neue Formate für Befehlsschaltfläche)
- Datenzugriffsseiten sind **nicht** mehr funktionsfähig!
- Das bereits in Office Access 2007 eingeführte neue Sicherheitsmodell wurde weiter verbessert. Die entsprechenden Einstellungen werden im *Microsoft Office-Sicherheitscenter* vorgenommen.

¹ Leider lässt sich die *TempVars*-Collection nicht auch für Datenmakros verwenden.

- Mit den Access Services in Microsoft SharePoint Server 2010 können Sie Webdatenbanken erstellen, auf welche Benutzer per Webbrowser zugreifen können.

Microsoft Office Access 2013

Die ganzen Entwicklungsressourcen wurden für die neuen Web Apps aufgewendet. Auch die Zusammenarbeit mit SharePoint fällt seitdem etwas komfortabler aus. Damit lagen die größten Neuerungen in der Anbindung von Cloud-Diensten, welche das Datenbank-Hosting auch online erlauben.

1.4.2 Microsoft Access 2016 – viel Lärm um nichts?

Endlich werden lang gehegte Wünsche wahr – eine maximale Datenbankgröße von 2 TB, Unterstützung für Volltextsuche, OData, .NET 4.5-Integration und eine Hilfe, die den Namen auch verdient ...

Hallo, Hallo, wachen Sie auf, Sie sollten nicht zu viel träumen!

Natürlich sind diese Features auch in Access 2016 **nicht** enthalten, dafür aber die "beruhigende" Nachricht:

Es gibt keine vorhandenen Features und Funktionen aus früheren Versionen, die in Access 2016 nicht mehr unterstützt werden!

D.h., nach einem Upgrade von Access 2013 auf Access 2016 stehen Ihnen auch weiterhin alle gewohnten Features zur Verfügung. Neben dem farbenfrohen Rahmen der Bedienoberfläche werden Sie aber auch weitere "wichtige" neue Features entdecken:

Was möchten Sie tun?

Es gibt jetzt im Menüband das Textfeld "Was möchten Sie tun?" bzw. "Sie wünschen?" (Glühbirnen-Symbol). Hier können Sie Suchbegriffe eingeben und dazu entsprechende Hilfestellungen erhalten.

Neue Designs

Auf Access können Sie jetzt zwei Office-Designs anwenden: "Farbig" und "Weiß" (Zugriff über *Datei/Optionen/Allgemein/Office-Design*).

Datenquellendaten nach Excel exportieren

Dieses in das Dialogfeld "Tabellenverknüpfungs-Manager" integrierte Feature ist dann besonders hilfreich, wenn Sie die Access-Anwendung nicht selbst entworfen haben.

Modernisierte Vorlagen

Fünf der beliebtesten Datenbankvorlagen weisen nun ein moderneres Aussehen und Verhalten auf.

Erleichterungen für Kunden mit einer SharePoint-Implementierung

Nutzer von SharePoint 2016 mit Access Services können auf verbesserte Features zugreifen.

1.4.3 Der inoffizielle Access-Friedhof (Access 2013/2016)

Alle Jahre wieder – Microsoft entstaubt seine Produkte und dabei brechen ab und zu einige Ecken und Verzierung ab, die funktionell mehr oder weniger wichtig waren. Begleiten Sie uns also abschließend auf einem kleinen Rundgang durch den über die Jahre recht stattlich gewordenen "Microsoft Access-Friedhof":

- Die noch aus Access 2010 bekannten **Web-Datenbanken** wurden nach kurzer Lebensphase sang- und klanglos wieder beerdigt. Microsoft hatte wieder mal "das Ruder herumgerissen" und so stehen alle, die dem Marketinggeblubber der Experten vertraut haben, im Regen ... Sie können zwar noch die alten Projekte zu Tode pflegen (Öffnen geht, Neu erstellen nicht), aber wer will das noch bei dieser Perspektive?
- Dass die **ADP** dem Tode geweiht waren, ist schon lange bekannt, seit Access 2013 wurde konsequenter Weise auch damit endgültig aufgeräumt.
- Wer gerne Daten mit **dBASE-Dateien** ausgetauscht hat (im Desktop-Bereich entgegen aller Erwartung noch immer weit verbreitet), ist mit Access 2013/2016 nicht gut beraten, die Unterstützung dafür ist ersatzlos weggefallen. Da ist es zu verschmerzen, dass auch das **Access 97-Format** nicht mehr gelesen werden kann.
- Vermutlich haben Sie die **Access Data Collection**-Funktion (sammeln von Daten per E-Mail) nie verwendet, so fällt es nicht schwer, darauf ebenfalls zu verzichten.
- Etwas anders sieht es bei **PivotChart und PivotTabellen** aus, diese waren ein praktisches Feature, dass man jetzt mit Excel ersetzen darf.
- Sie können die **Access 2003-Symboleleisten und -Menüs** nicht mehr sinnvoll verwenden.
- Last but not least, auch die **Access Source Code-Verwaltung**, der **Access Upsizing-Wizard** und die **Replikationsfunktionen** sind den Weg alles Irdischen gegangen¹ – R.I.P.

1.5 Übersichten und Ergänzungen

Praktische Nachschlagemöglichkeiten für die tägliche Arbeit des VBA-Programmierers bieten Ihnen die folgenden Tabellen.

1.5.1 Deutsche und englische Bezeichner

Die folgende Tabelle zeigt eine alphabetische Auflistung von wichtigen Eigenschaften und Ereignissen, wie sie im Eigenschaftenblatt für Formulare, Berichte und Steuerelemente aufgeführt sind,

¹ Erstaunlich, dass es die Desktop-Datenbanken noch geschafft haben, wo doch jetzt alles in die Cloud ausgelagert wird.

und ihre äquivalenten VBA-Bezeichner, wie sie zum Teil im Eigenschaftenfenster des VBA-Editors zu sehen sind.

HINWEIS: Am einfachsten erfahren Sie Näheres zum originalen (englischen) Bezeichner, wenn Sie auf den entsprechenden Eintrag im Eigenschaftenfenster klicken und dann die Funktionstaste **[F1]** drücken.

Deutsch	Access-Basic	Deutsch	Access-Basic
Aktiviert	<i>Enabled</i>	Hilfedatei	<i>HelpFile</i>
Anfügen zulassen	<i>AllowAdditions</i>	Hilfekontext	<i>HelpContext</i>
Automatisch zentrieren	<i>AutoCenter</i>	Hintergrundart	<i>BackStyle</i>
Bearbeitungen zulassen	<i>AllowEdit</i>	Hintergrundfarbe	<i>BackColor</i>
Bei Änderung	<i>Change</i>	Höhe	<i>Height</i>
Bei Entladen	<i>UnLoad</i>	In Reihenfolge	<i>TabStop</i>
Bei Fokuserhalt	<i>GotFocus</i>	Links	<i>Left</i>
Bei Fokusverlust	<i>LostFocus</i>	Löschen zulassen	<i>AllowDeletions</i>
Bei Größenänderung	<i>Resize</i>	Marke	<i>Tag</i>
Bei Laden	<i>Load</i>	Menüleiste	<i>MenuBar</i>
Bei Mausbewegung	<i>MouseMove</i>	Nach Aktualisierung	<i>AfterUpdate</i>
Bei Maustaste Ab	<i>MouseDown</i>	Nach Eingabe	<i>AfterEnter</i>
Bei Maustaste Auf	<i>MouseUp</i>	Nach Löschbestätigung	<i>AfterDelConfirm</i>
Bei Taste	<i>KeyPress</i>	Name	<i>Name</i>
Bei Taste Ab	<i>KeyDown</i>	Navigationsschaltflächen	<i>NavigationButtons</i>
Bei Taste Auf	<i>KeyUp</i>	Oben	<i>Top</i>
Beim Doppelklicken	<i>DbtClick</i>	PopUp	<i>PopUp</i>
Beim Klicken	<i>Click</i>	Rahmenart	<i>BorderStyle</i>
Beim Öffnen	<i>Open</i>	Rahmenbreite	<i>BorderWidth</i>
Beim Schließen	<i>Close</i>	Rahmenfarbe	<i>BorderColor</i>
Beschriftung	<i>Caption</i>	Raster X	<i>GridX</i>
Bild	<i>Picture</i>	Raster Y	<i>GridY</i>
Bildlaufleisten	<i>ScrollBars</i>	Reihenfolgenposition	<i>TabIndex</i>
Breite	<i>Width</i>	Schneller Laserdruck	<i>FastLaserPrinting</i>
Daten eingeben	<i>DataEntry</i>	Schriftart	<i>FontName</i>
Datensatzquelle	<i>RecordSource</i>	Schriftbreite	<i>FontWeight</i>
Datensatzgruppentyp	<i>RecordsetType</i>	Schriftgröße	<i>FontSize</i>
Datensatzmarkierer	<i>RecordSelectors</i>	Sichtbar	<i>Visible</i>
Eingabeformat	<i>InputMask</i>	Sortiert nach	<i>OrderBy</i>
Filter	<i>Filter</i>	Standardansicht	<i>DefaultView</i>
Filter anwenden	<i>ApplyFilter</i>	Standardwert	<i>DefaultValue</i>

Deutsch	Access-Basic	Deutsch	Access-Basic
Filter zulassen	<i>AllowFilters</i>	Steuerelementinhalt	<i>ControlSource</i>
Format	<i>Format</i>	Textausrichtung	<i>TextAlign</i>
Gebunden	<i>Modal</i>	Trennlinien	<i>DividingLines</i>
Gesperrt	<i>Locked</i>	Vor Aktualisierung	<i>BeforeUpdate</i>

1.5.2 DoCmd-Objekt

Dieses Objekt baut dem Makro-Programmierer eine "Goldene Brücke" zu VBA. Gewissermaßen als Mittler zwischen Makro- und Code-Programmierung gestatten es seine Methoden nahezu alle vordefinierten Aktionen auszuführen. Bislang haben Sie z.B. ausgiebig von der *Close*-Methode des *DoCmd*-Objekts Gebrauch gemacht, um das Formular zu schließen.

Methoden des DoCmd-Objekts

Methode	Makro-Aktion
<i>AddMenu</i>	HinzufügenMenü
<i>ApplyFilter</i>	AnwendenFilter
<i>Beep</i>	Signalton
<i>BrowseTo</i>	GeheZuSeite
<i>CancelEvent</i>	AbbrechenEreignis
<i>Close</i>	Schließen
<i>CopyObject</i>	KopierenObjekt
<i>DeleteObject</i>	LöschenObjekt
<i>DoMenuItem</i>	AusführenMenübefehl
<i>Echo</i>	Echo (besser Echo-Methode des <i>Application</i> -Objekts verwenden)
<i>FindNext</i>	SuchenWeiter
<i>FindRecord</i>	SuchenDatensatz
<i>GoToControl</i>	GeheZuSteuerelement (besser <i>SetFocus</i> -Methode verwenden)
<i>GoToPage</i>	GeheZuSeite (besser gleichnamige <i>Form</i> -Methode nehmen)
<i>GoToRecord</i>	GeheZuDatensatz
<i>Hourglass</i>	Sanduhr
<i>Maximize</i>	Maximieren
<i>Minimize</i>	Minimieren
<i>MoveSize</i>	Positionieren
<i>OpenForm</i>	ÖffnenFormular
<i>OpenModule</i>	ÖffnenModul
<i>OpenQuery</i>	ÖffnenAbfrage
<i>OpenReport</i>	ÖffnenBericht
<i>OpenTable</i>	ÖffnenTabelle

Methode	Makro-Aktion
<i>OutputTo</i>	AusgabeIn
<i>PrintOut</i>	Drucken
<i>Quit</i>	Beenden
<i>RefreshRecord</i>	AktualisierenDatensatz
<i>Rename</i>	UmbenennenObjekt
<i>RepaintObject</i>	AktualisierenObjekt
<i>Requery</i>	AktualisierenDaten
<i>Restore</i>	Wiederherstellen
<i>RunMacro</i>	AusführenMakro
<i>RunSQL</i>	AusführenSQL
<i>Save</i>	Speichern
<i>SelectObject</i>	AuswählenObjekt
<i>SendObject</i>	SendenObjekt
<i>SetFilter</i>	FestlegenFilter
<i>SetMenuItem</i>	SetzenMenüelement
<i>SetOrderBy</i>	FestlegenSortiertNach
<i>SetParameter</i>	Parameterübergabe
<i>SetWarnings</i>	Warnmeldungen
<i>ShowAllRecords</i>	AnzeigenAlleDatensätze
<i>ShowToolbar</i>	EinblendenSymbolleiste
<i>TransferDatabase</i>	TransferDatenbank
<i>TransferSpreadSheet</i>	TransferArbeitsblatt
<i>TransferText</i>	TransferText

Nicht unterstützte Aktionen

Die folgenden Makro-Aktionen werden durch das *DoCmd*-Objekt nicht unterstützt:

Aktion	Basic-Programmierung
<i>AusführenAnwendung</i>	Verwenden Sie die <i>Shell</i> -Funktion
<i>AusführenCode</i>	Führen Sie die entsprechende Funktion direkt in VBA aus
<i>HinzufügenMenü</i>	Programmieren Sie ein Menü-Makro
<i>Meldung</i>	Verwenden Sie die <i>MsgBox</i> -Funktion
<i>SetzenWert</i>	Legen Sie den Wert direkt in VBA fest!
<i>StopAlleMakros</i>	Stoppt die Ausführung aller Makros
<i>StopMakro</i>	Stoppt die Ausführung des aktuellen Makros
<i>Tastaturbefehle</i>	Verwenden Sie die Anweisung <i>SendKeys</i>

Bemerkungen

- Die meisten Methoden des *DoCmd*-Objekts verfügen über Argumente. Einige sind erforderlich, während andere optional sind. Verzichten Sie auf optionale Argumente, so setzen die Argumente die Standardwerte für die jeweilige Aktion voraus. Die Methode *OpenForm* verwendet z.B. sieben Argumente, jedoch ist lediglich das erste Argument (Formularname) unbedingt erforderlich.
- In vielen Fällen, wo zwar entsprechende Methoden des *DoCmd*-Objekts bereitstehen, kann man durch direkte VBA-Programmierung effektiveren Quellcode schreiben. Andererseits gibt es jedoch auch eine Reihe von Aktionen, die man mit VBA nicht ohne weiteres nachbilden kann und die sich nur über entsprechende Methoden des *DoCmd*-Objekts umsetzen lassen.

Programmieren mit VBA

Um bei der Vielzahl von VBA-Sprachelementen den Überblick nicht zu verlieren, sollen in diesem Kapitel zunächst die elementaren Visual Basic-Sprachkonstrukte vorgestellt werden. Die objekt-orientierten und datenbankspezifischen Erweiterungen wollen wir erst in späteren Kapiteln behandeln.

HINWEIS: Viele Beispiele dieses Kapitels werden im *Direktfenster* durchgeführt bzw. mittels *MsgBox*-Anweisung ausgegeben.

2.1 Datentypen, Variablen und Konstanten

Jede Programmiersprache "lebt" in erster Linie von den zur Verfügung stehenden Variablentypen und Konstanten, die wiederum bestimmten Datentypen entsprechen.

2.1.1 Übersicht

Die folgende Tabelle gibt eine Übersicht der standardmäßig zur Verfügung stehenden Datentypen. Neben den genannten Datentypen lassen sich auch benutzerdefinierte (strukturierte) Datentypen mit der *Type*-Anweisung deklarieren (siehe Seite 96).

Datentyp	Speicherbedarf	Erläuterung	Typkennzeichen
<i>Byte</i>	1 Byte	Ganzzahl zwischen 0 und 255	
<i>Boolean</i>	2 Byte	Wahrheitswert (<i>True</i> = -1/ <i>False</i> = 0)	
<i>Integer</i>	2 Byte	Ganzzahl zwischen -32.768 und +32.767	%
<i>Long</i>	4 Byte	Lange Ganzzahl (<i>Long Integer</i>) zwischen -2.147.483.648 und +2.147.483.647	&
<i>Single</i>	4 Byte	Einfachgenaue Gleitkommazahl 7-stelliger Genauigkeit zwischen $10E^{-38}$ und $10E^{+38}$	!
<i>Double</i>	8 Byte	Doppeltgenaue Gleitkommazahl 16-stelliger Genauigkeit zwischen $10E^{-308}$ und $10E^{+308}$	#

Datentyp	Speicherbedarf	Erläuterung	Typkennzeichen
<i>Currency</i>	8 Byte	Währung (ohne Rundungsfehler!) mit 15 Stellen vor und vier Stellen nach dem Dezimalpunkt	@
<i>Date</i>	8 Byte	1. Januar 100 0:00:00 bis 31. Dezember 9999 23:59:59	
<i>String</i>	1 Byte/Zeichen + 10 Byte	Zeichenfolge mit max. 2.000.000.000 Zeichen	\$
<i>Variant (mit Zahlen)</i>	16 Byte	Universeller Datentyp mit numerischen Werten im Bereich von <i>Double</i> -Zahlen	
<i>Variant (mit Zeichen)</i>	1 Byte/Zeichen + 22 Byte	Universeller Datentyp wie <i>String</i> variabler Länge	
<i>Object</i>	4 Byte	Verweis auf ein Objekt	

Bemerkungen

- Zum *String*-Datentyp siehe Seite 87 (Zeichenkettenfunktionen)
- Zum *Object*-Datentyp siehe Kapitel 6 (Programmieren mit Objekten)
- Zu allen anderen Typen siehe Seite 83 (Einzelheiten zu den Datentypen)

2.1.2 Variablendeklaration

Deklarieren bedeutet so viel wie "bekannt machen". Dazu lassen Sie sich einen Variablennamen bzw. Bezeichner einfallen und weisen diesem einen bestimmten Datentyp (bzw. ein Typkennzeichen) zu.

Dim-Anweisung

In vielen Fällen werden Sie zur Deklaration von Variablen die *Dim*-Anweisung einsetzen¹, wobei der Datentyp mit *As* zugewiesen wird. Geschieht dies innerhalb einer Funktion oder Prozedur, so beschränkt sich die Gültigkeit der Variablen logischerweise nur auf diesen Bereich.

BEISPIEL: Deklaration einer *Integer*-Variablen *a*:

```
Sub test()
    Dim a As Integer
    ...           ' Gültigkeitsbereich von a
End Sub
```

HINWEIS: Zweckmäßigerweise sollten die Variablendeklarationen am Anfang stehen, müssen aber nicht.

Sie können auch innerhalb einer Zeile mehrere Variablen deklarieren.

¹ Auf die anderen Möglichkeiten kommen wir später zu sprechen.

BEISPIEL:

Drei Variablen werden deklariert:

```
Dim a As Integer, b As Integer, c As Single
```

Falls die Angabe des Datentyps weggelassen (oder "vergessen") wird, ist die Variable automatisch vom *Variant*-Datentyp.

BEISPIEL: Falls Sie das Vorgängerbeispiel so schreiben:

```
Dim a, b As Integer, c As Single
```

ist *a* nicht, wie vielleicht erwartet, vom Datentyp *Integer*, sondern vom Datentyp *Variant*!

Bezeichner

Bei der Benennung von Variablen sind bestimmte Konventionen einzuhalten. Variablennamen müssen:

- mit einem Buchstaben beginnen
- kürzer als 256 Zeichen sein
- ohne eingeschlossene Leerzeichen, Punkte und Typkennzeichner auskommen
- innerhalb ihres Gültigkeitsbereichs eindeutig sein (also keine Mehrfachvergabe des gleichen Namens)

HINWEIS: Die gleichen Konventionen gelten auch bei der Namensgebung für beliebige andere nutzerdefinierte Sprachelemente (Konstanten, nutzerdefinierte Typen, Funktionen/Prozeduren und andere).

BEISPIEL: Gültige Deklarationen:

```
Dim s As String  
Dim anzahl As Integer  
Dim Netto_Betrag As Currency
```

BEISPIEL: Ungültige Deklarationen:

```
Dim i% As Integer  
Dim 3Viertel As Single  
Dim Netto Betrag As Currency
```

Typkennzeichen

In den meisten Fällen kann man durch Einsatz von Typkennzeichen die Variablendeklarationen verkürzen¹.

¹ Für den Datentyp *Variant* gibt es kein Typkennzeichen, ebenso wenig für die seit Microsoft Access 95 neu hinzugekommenen Datentypen.

Anstatt:

```
Dim i As Integer, st As String
```

kann man auch

```
Dim i%, st$
```

schreiben.

HINWEIS: Es ist eine weit verbreitete Unsitte, das Typkennzeichen auch außerhalb der Variablen Deklaration anzugeben.

BEISPIEL: Man sollte die Anweisungen

```
For i% = 1 To 5  
    st$ = "Achtung!"  
Next i%
```

wie folgt ersetzen:

```
For i = 1 To 5  
    st = "Achtung!"  
Next i
```

Vor der allzu gedankenlosen Verwendung von Typkennzeichen sei jedoch gewarnt:

HINWEIS: Typkennzeichen sind dann nicht zu empfehlen, wenn auch Programmierer anderer Sprachen (C, Delphi, Java, ...) Ihren VBA-Code nutzen sollen. Für diese Programmierer bedeutet die ausführliche Deklaration mit *As* eine deutlich verbesserte Lesbarkeit und Transparenz. Aus diesem Grund werden wir im vorliegenden Buch nur in wenigen Fällen von Typkennzeichen Gebrauch machen.

DefType-Anweisungen

Wer eine größere Anzahl von Variablen gleichen Typs verwendet, dem fällt unter Umständen deren einzelne Deklaration lästig. Um dies zu vereinfachen, sollte man auf *DefType*-Anweisungen zurückgreifen.

BEISPIEL: Die Anweisung

```
DefInt i-n
```

deklariert explizit alle Variablen, die mit den Buchstaben *i* bis *n* beginnen¹, als *Integer*, sodass beispielsweise die folgende Zuweisungen möglich ist:

```
nummer = 25  
i = 100
```

¹ Die Groß-/Kleinschreibung spielt keine Rolle!

Anweisung	Datentyp	Anweisung	Datentyp
<i>DefBool</i>	<i>Boolean</i>	<i>DefDbf</i>	<i>Double</i>
<i>DefByte</i>	<i>Byte</i>	<i>DefDate</i>	<i>Date</i>
<i>DefInt</i>	<i>Integer</i>	<i>DefStr</i>	<i>String</i>
<i>DefLng</i>	<i>Long</i>	<i>DefObj</i>	<i>Object</i>
<i>DefCur</i>	<i>Currency</i>	<i>DefVar</i>	<i>Variant</i>
<i>DefSng</i>	<i>Single</i>		

DefType-Anweisungen können mit *Dim* außer Kraft gesetzt werden.

BEISPIEL: Die beiden folgenden Anweisungen

```
DefStr a-v
Dim preis As Currency
```

deklarieren alle Variablen, die mit den Buchstaben A bis V beginnen, als Stringtypen. Einzige Ausnahme ist die Währungsvariable *preis*.

DefType-Anweisungen beziehen sich nicht nur auf Variablen, sondern gleichermaßen auch auf Funktionstypen und *PropertyGet*-Prozeduren.

BEISPIEL: Zum vorangegangenen Beispiel könnte man folgende Anweisungen hinzufügen:

```
Function addiereName(name, vorname)
    addiereName = vorname & " " & name
End Function
```

```
vollerName = addiereName("Mueller", "Heinz")
```

Die Funktion *addiereName* beginnt mit dem Buchstaben "a", d.h., ihr Rückgabewert ist vom Typ *String*.

Bemerkungen

- *DefType*-Anweisungen dürfen nur auf Modulebene (also nicht innerhalb von Funktionen und Prozeduren) eingesetzt werden. Sie können nicht öffentlich (*Public*) deklariert werden.
- Die Buchstabenbereiche mehrerer *DefType*-Anweisungen dürfen sich nicht überschneiden, da sonst die Eindeutigkeit verloren geht.

Option Explicit

VBA erlaubt so genannte implizite Variablendeklarationen (ein Rudiment der alten Basic-Ära). Im Gegensatz zur expliziten Deklaration von Variablen mit *Dim*, *Private*, *Public* oder *Static* können neue Variablen auch durch einfache Zuweisungen entstehen.

BEISPIEL: Die Anweisung:

```
zahl = 100
```

erzeugt eine Variable mit dem Namen *zahl* und weist ihr den Wert 100 zu.

Implizit deklarierte Variablen sind immer vom Datentyp *Variant*. Variablen dieses universellen Typs belegen in der Regel mehr Speicherplatz als andere Variablen. Ein Programm arbeitet aber effizienter, wenn seine Variablen explizit und mit dem geeigneten Datentyp deklariert wurden.

Die Möglichkeit der impliziten Variablendeklaration war sicherlich einer der Gründe, warum die Sprache Basic (und deren Derivate) in der Vergangenheit einen zweifelhaften Ruf hatte. Eine solche Großzügigkeit provoziert geradezu Programmierfehler, die allein durch reine Schreibfehler entstehen können.

BEISPIEL: Stellen Sie sich vor, Sie haben anstatt:

```
person = "Mueller"
```

versehentlich:

```
personn = "Mueller"
```

eingetippt.

Es geistert nun eine unbekannte Variable *personn* im Programm herum, von deren Existenz Sie keine Ahnung haben. Ihr Programm scheint zwar zu funktionieren, aber es verarbeitet bestimmte Informationen fehlerhaft, was böse Auswirkungen haben kann, deren Ursachen man nur durch zermürende Fehlersuche auf die Spur kommt.

Negativen Erlebnissen obiger Art können Sie ein für allemal aus dem Weg gehen, wenn Sie die Anweisung *Option Explicit* an den Beginn des Deklarationsteils eines jeden Moduls setzen. Vom Compiler werden dann nur explizit deklarierte Variablen akzeptiert. Andernfalls erfolgt eine Fehlermeldung.

BEISPIEL: Die Anweisung:

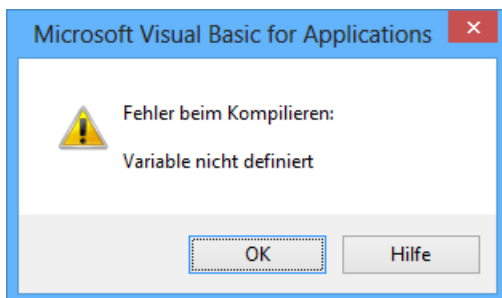
```
Option Explicit
```

```
Dim person As String
```

sorgt dafür, dass bei der Zuweisung:

```
personn = "Mueller"
```

der Compiler die Übersetzung mit einer Fehlermeldung stoppt:



HINWEIS: Vergesslichen sei empfohlen, den Menübefehl *Extras/Optionen* zu wählen und die Registerkarte *Editor* anzuklicken. Aktivieren Sie nun das Kontrollkästchen *Variablendeklaration erforderlich*. Die Anweisung *Option Explicit* erscheint dann automatisch zu Beginn eines jeden neuen Moduls.

2.1.3 Konstantendeklaration

Nachdem wir uns bereits mit der Variablendeklaration ausführlich auseinander gesetzt haben, ist die Anwendung der *Const*-Anweisung schnell erklärt:

BEISPIEL: Die nachfolgenden Anweisungen definieren Konstanten:

```
Const Pi As Single = 3.14159
Public Const hinweis1 = "ACHTUNG!"
Private Const zahl1 As Integer = 7
Const hinweis2 = "VORSICHT!"
Const zahl2 As Double = 125.987
```

Wie Sie sehen, sind auch hier Typkennzeichen, Mehrfachanweisungen in einer Zeile sowie private und öffentliche Deklarationen möglich. Standardmäßig gelten alle einfachen *Const*-Anweisungen als *Private*. Auch für die Namensgebung sind die gleichen Konventionen wie bei Variablen anzuwenden.

Beachten Sie jedoch folgende Unterschiede:

- Zulässige Typen sind *Byte*, *Boolean*, *Integer*, *Long*, *Currency*, *Single*, *Double*, *Date*, *String* oder *Variant*.
- Verwenden Sie für jede deklarierte Variable einen separaten *As Type*-Abschnitt. Falls Sie die Typangabe weglassen, wird automatisch ein geeigneter Datentyp zugewiesen.
- Es dürfte jedem klar sein, dass der Wert einer einmal definierten Konstante während des Programms nicht verändert werden darf.
- Nur in Standardmodulen (also nicht in Klassenmodulen, d.h. Form- bzw. Berichtsmodulen oder gar auf Prozedurebene) können mit dem Schlüsselwort *Public* öffentliche Konstanten deklariert werden.

In VBA sind bereits viele "vorgefertigte" Konstanten integriert, die im Allgemeinen mit den Buchstaben *vb* beginnen.

BEISPIEL: Das Textfeld wird rot eingefärbt (*vbRed*) und der Text umbrochen (*vbCrLf*).

```
Text0.BackColor = vbRed
Text0.Value = "Das ist die erste " & vbCrLf & "das ist die zweite Zeile!"
```

2.1.4 Gültigkeitsbereiche

Bevor Sie eine Variable/Konstante deklarieren (und ihr damit Speicherplatz zuweisen), sollten Sie sich Klarheit über deren Gültigkeitsbereich verschaffen. Dieser kann sich auf eine der folgenden Ebenen beziehen:

- Aktuelle und andere Access-Datenbankapplikationen
- Nur die aktuelle Access-Datenbankapplikation
- Module
- Prozeduren

Der Gültigkeitsbereich einer oberen Ebene umfasst auch die darunter liegenden Ebenen.

Private- und Public-Deklarationen

Diese Anweisungen können anstatt *Dim* verwendet werden, allerdings nur auf Modulebene, d.h., nicht innerhalb von Funktionen/Prozeduren. In VBA unterscheiden wir zwischen zwei Arten von Modulen:

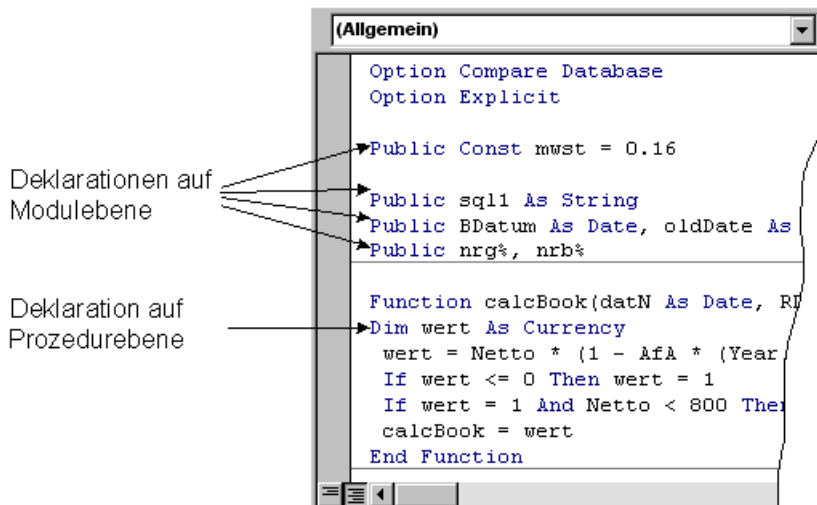
- Klassenmodule (Formular-/Berichtsmodule, eigenständige Klassenmodule)
- Standardmodule (Code-Module)

Sie erreichen das entsprechende Codefenster über das Symbol *Code anzeigen* der Befehlsgruppe *Tools* in der Hauptregisterkarte *Entwurf*:



Sämtlicher Code außerhalb von Funktionen/Prozeduren – und dazu gehört auch die Variablendeklaration auf Modulebene – steht im Deklarationsabschnitt ganz am Anfang des Moduls. Wenn Sie hier die einfache *Dim*-Anweisung verwenden, gelten die damit deklarierten Variablen nur innerhalb des Moduls, also nur für die dort angesiedelten Funktionen/Prozeduren.

Wenn Sie möchten, dass die auf Modulebene deklarierten Variablen auch noch in allen anderen Modulen der aktuellen Datenbank zur Verfügung stehen sollen, so müssen sie mit *Public* als "öffentlich" gekennzeichnet werden.



BEISPIEL: Die auf Modulebene angewendete Anweisung:

```
Public a As Integer
```

deklariert eine öffentliche *Integer*-Variable *a*, die auch in allen anderen Modulen der Datenbank zur Verfügung steht.

Beachten Sie folgenden Unterschied: Während *Public*-Variablen eines Formular-/Berichtsmoduls nur innerhalb von Funktionen/Prozeduren der aktuellen Datenbank verwendet werden können, darf auf *Public*-Variablen eines Standardmoduls auch von anderen Datenbanken aus, die mit ihrer Deklaration auf die aktuelle Datenbank verweisen, zugegriffen werden.

HINWEIS: Eine noch effektivere Möglichkeit zur Definition globaler Variablen wurde bereits unter Access 2007 mit der neuen *TempVars*-Auflistung des *Application*-Objekts eingeführt (siehe "Eigenschaften und Methoden des *Application*-Objekts" in Kapitel 6).

Eine mit *Private* gekennzeichnete Variablendeklaration weist ausdrücklich darauf hin, dass die Variable nur innerhalb des Moduls zur Verfügung steht. Standardmäßig sind alle mit *Dim* auf Modulebene deklarierten Variablen privat, sodass man hier auch ohne das Schlüsselwort *Private* auskommt.

BEISPIEL: Die auf Modulebene eingesetzte Anweisung:

```
Private a As Integer
```

ist äquivalent zur Anweisung:

```
Dim a As Integer
```

und deklariert eine *Integer*-Variable *a*, die nur den Funktionen/Prozeduren dieses Moduls zur Verfügung steht.

Static-Anweisung

Neben den erwähnten Gültigkeitsbereichen (Prozedurebene, Modulebene, Datenbankebene) von Variablen ist auch noch die Dauer, während der eine Variable ihren Wert beibehält, von Interesse.

Beim Aufruf einer Prozedur werden alle dort deklarierten Variablen initialisiert. Numerische Variablen erhalten den Wert 0, eine Zeichenkette variabler Länge wird zur leeren Zeichenfolge (""), und eine Zeichenfolge fester Länge wird zu dem Zeichen, das vom ASCII-Zeichencode als 0 bzw. *Chr(0)* interpretiert wird. Variablen des Datentyps *Variant* werden mit *Empty* initialisiert. Das Gleiche trifft auch auf jedes Element von Variablen eines benutzerdefinierten Datentyps (*Type-Anweisung*) zu.

Etwas anders verhält es sich bei Objektvariablen. Zwar reserviert Microsoft Access dafür Speicherplatz, die Variable erhält aber erst dann einen Wert, wenn ihr mit Hilfe der *Set*-Anweisung eine Objektreferenz zugewiesen wurde (siehe Kapitel 6).

Eine mit *Dim* auf Prozedurebene deklarierte Variable behält ihren Wert nur so lange, bis die Ausführung der Prozedur beendet ist.

BEISPIEL: Platzieren Sie auf einem Formular eine Befehlsschaltfläche und hinterlegen Sie das *Click*-Ereignis mit folgendem Code:

```
Sub Befehl0_Click()  
    Dim anzahl As Integer  
    anzahl = anzahl + 1  
    MsgBox anzahl  
End Sub
```

Nach dem Öffnen des Formulars und wiederholtem Klicken auf die Schaltfläche zeigt das Meldungsfeld (auch als Messagebox bezeichnet) unverändert den Wert 1 an.

Wird *Dim* hingegen auf Modulebene eingesetzt, so behalten die dort deklarierten Variablen ihren Wert, bis das Modul zurückgesetzt oder neu gestartet wird.

BEISPIEL: Im Unterschied zum Vorgängerbeispiel erfolgt jetzt die Variablendeklaration auf Modulebene:

```
Dim anzahl As Integer  
  
Sub Befehl0_Click()  
    anzahl = anzahl + 1  
    MsgBox anzahl  
End Sub
```

Sie werden feststellen, dass sich der im Meldungsfeld angezeigte Wert bei jedem Klicken um 1 erhöht.

Um den gleichen Effekt wie auf Modul- auch auf Prozedurebene zu erreichen, muss anstatt *Dim* das Schlüsselwort *Static* verwendet werden.

BEISPIEL: Verwendung von *Static*:

```
Sub Befehl0_Click()
```

Beachten Sie, das *anzahl* jetzt innerhalb der Methode definiert wird:

```
Static anzahl As Integer  
anzahl = anzahl + 1  
MsgBox anzahl  
End Sub
```

Nach jedem Aufruf erhöht sich der Wert von *anzahl* um 1.

Bemerkungen

- Die *Static*-Anweisung ist nur auf Prozedurebene anwendbar. Ansonsten gelten die gleichen Konventionen wie für *Dim*.
- *Static* lässt sich auch auf Daten-Arrays anwenden und wird auch in Verbindung mit Funktionen bzw. Prozeduren eingesetzt (siehe Seite 123).

2.2 Einzelheiten zu den Datentypen

Da VBA englischsprachige Konventionen verwendet, sind aufgrund der deutschsprachigen Windows-Systemeinstellung einige Besonderheiten zu beachten.

2.2.1 Single- und Double-Datentypen

Wenn Sie Fließkommazahlen im Quelltext zuweisen, dürfen Sie nicht das Komma, sondern müssen den Punkt als Dezimaltrennzeichen verwenden.

BEISPIEL: Dezimaltrennzeichen

```
a = 0.45
```

Andererseits dürfen Sie sich nicht wundern, wenn z.B. bei Zahleneingaben in ein Textfeld (bzw. bei der Anzeige) nur das Komma als Dezimaltrennzeichen akzeptiert wird.

2.2.2 Integer-, Long- und Boolean-Datentypen

Farbwerte sind im Allgemeinen vom Datentyp *Long*.

BEISPIEL: Der Formularhintergrund wird gelb eingefärbt.

```
Dim farbe As Long  
farbe = 65535 ' entspricht der Konstanten vbYellow  
Me.Section(0).BackColor = farbe
```

Auch hinter den Wahrheitswerten *True* und *False* verbergen sich Integer-Zahlen (0 für *False*, sonst *True*). Während Sie im Quellcode *True* und *False* (oder eine Integer-Zahl) zuweisen müssen, erscheint bei der Anzeige "Wahr" bzw. "Falsch".

BEISPIEL: Für die *Boolean*-Variable *bVar* = -1 ergibt sich die Anzeige "Wahr".

```
Dim bVar As Boolean
bVar = -1
MsgBox bVar
```

2.2.3 Date-Datentyp

Variablen dieses Typs werden intern als 64-Bit-Zahlen (8 Bytes) gespeichert und können ein Datum im Bereich vom *01. Januar 100* bis zum *31. Dezember 9999* und eine Uhrzeit im Bereich von *0:00:00* bis *23:59:59* darstellen.

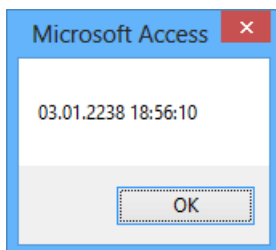
HINWEIS: Beachten Sie, dass eine Variable vom Datentyp *Date* im Allgemeinen ein Datum und eine Zeit speichert!

Ein Datums- oder Zeitliteral muss links und rechts durch das Zeichen "#" (Nummernzeichen) eingeschlossen sein, z.B. *#January 31, 2011#* oder *#31 Jan 11#* oder *#01/31/2011#*. Besonders am letzten Ausdruck sehen Sie deutlich, dass hier grundsätzlich die englische Schreibweise Anwendung findet, wo zuerst der Monat steht und erst dann der Tag folgt.

Hinter dem Datentyp *Date* versteckt sich eigentlich eine Gleitkommazahl (*Double*), bei der die Vorkommastellen das Datum abbilden.

BEISPIEL: Die Zahl *123456.789* repräsentiert den (weit in der Zukunft liegenden) Termin *3.1.2238 18:56:10*.

```
Dim z As Double, d As Date
z = 123456.789
d = z
MsgBox d
```

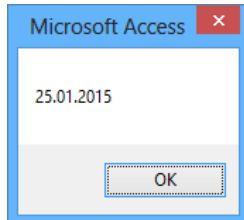


- Der Wert 0 entspricht dem Datum *30.12.1899*, der Wert 1 entspricht einem Tag (24 Stunden).
- Negative ganze Zahlen beziehen sich auf ein Datum vor dem *30. Dezember 1899*.
- Die Nachkommastellen bilden die Uhrzeit ab ($1 \text{ Stunde} = 1/24 = 0.0417$). Mitternacht hat demzufolge den Nachkommawert 0 und Mittag = $12 \times 1/24 = 0,5$.

Einfache Beispiele

BEISPIEL: Der 25.1.2015 wird einer Datumsvariablen zugewiesen und mit einem Meldungsfenster angezeigt.

```
Dim dat As Date
dat = #1/25/2015#
MsgBox dat
```



Der Tag vor dem Wert 0.

```
Dim dat As Date
dat = -1
MsgBox dat ' liefert den 29.12.1899
```

BEISPIEL: Berechnung eines Termins, der 14 Tage und drei Stunden vor dem 13.1.2014 12:45 liegt

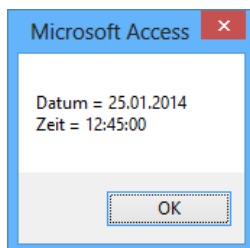
```
Dim dat As Date
dat = #1/13/2014 12:45:00 PM# - 14 - 3 / 24 ' liefert "30.12.2013 09:45:00"
```

Herausfiltern von Datum und Zeit

Mit Hilfe der *Int*-Funktion (siehe Seite 116) ist es einfach, eine *Date*-Variable in ihren ganzzahligen und gebrochenen Anteil zu zerlegen, um damit Datum und Zeit herauszufiltern.

BEISPIEL: Der Termin 25.1.2014 12:45 soll "zerlegt" werden.

```
Dim dat As Date, datum As Date, zeit As Date
dat = #1/25/2014 12:45#
datum = Int(dat)
zeit = dat - Int(dat)
MsgBox "Datum = " & datum & vbCrLf & "Zeit = " & zeit
```



Weitere Funktionen

Im Zusammenhang mit dem *Date*-Datentyp sind die *CDate*-Funktion und die vielfältigen Datums-/Zeitfunktionen von Bedeutung (siehe Seite 120).

BEISPIEL: Anstatt der Zuweisung

```
dat = #1/13/2014 12:45#
```

könnte man im obigen Beispiel auch schreiben

```
dat = CDate("13.1.2014 12:45")
```

Siehe dazu auch das Praxisbeispiel "Das Wochenende feststellen" (Seite 152).

Kurzeingabe

Wenn Sie ein Datum eingeben, das nur den Monat sowie ein oder zwei Ziffern umfasst, nimmt Access an, dass die Zahlen 1 bis 31 den Tag angeben und dass das Jahr das aktuelle Jahr ist.

HINWEIS: *December 03* wird als 1. Dezember des aktuellen Jahres und nicht des Jahres 2003 interpretiert.

Jahr 2000-Problematik

Zweistellige Jahresangaben sind mit Vorsicht zu genießen, denn sie werden unterschiedlich interpretiert, wobei als Stichtag der 1.1.1930 festgelegt ist:

Abgekürztes Jahresformat	Interpretation
1.1.00 bis 31.12.29	1.1.2000 bis 31.12.2029
1.1.30 bis 31.12.99	1.1.1930 bis 31.12.1999

BEISPIEL: Wenn Sie das Geburtsdatum Ihrer Urgroßmutter mit 28.11.26 angeben, dann haben Sie eine "Zeitreise" in die Zukunft unternommen.

```
Dim dat As Date
```

```
dat = #11/28/26#
```

```
MsgBox dat ' liefert den 28.11.2026
```

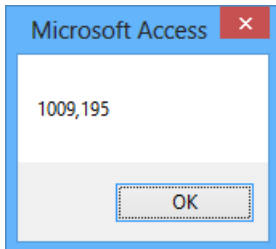
HINWEIS: Um sicherzustellen, dass Access die Datumswerte auch in ferner Zukunft nicht verfälscht, sollten Sie Jahreszahlen grundsätzlich vierstellig abspeichern.

2.2.4 Currency-Datentyp

Bei Währungswerten handelt es sich um einen Festkomma-Datentyp (8 Byte), welcher 15 Ziffern links und 4 Ziffern rechts vom Komma darstellen kann.

BEISPIEL: Anzeige eines Währungsbetrags (ohne Formatierung)

```
Dim gehalt As Currency  
gehalt = 1003.571 + 5.624  
MsgBox gehalt
```

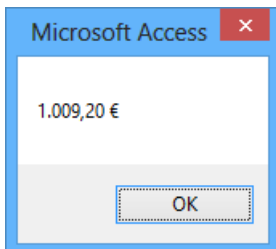


Wie Sie obigem Beispiel entnehmen, erfolgt die Anzeige ungerundet. Probleme können auch im Zusammenhang mit dem Dezimaltrennzeichen auftreten (im Quelltext als Punkt, in der Anzeige aber als Komma). Aus diesen Gründen sollten Sie zur Anzeige die *Format\$*- oder die *FormatCurrency\$*-Funktion verwenden (siehe Seite 110).

BEISPIEL: Zwei Möglichkeiten zur formatierten Anzeige eines Währungsbetrags.

```
Dim gehalt As Currency  
gehalt = 1003.571 + 5.624  
MsgBox Format$(gehalt, "Currency") ' Variante A  
MsgBox FormatCurrency$(gehalt)     ' Variante B
```

Das Ergebnis ist in beiden Fällen gleich.



2.2.5 String-Datentyp

Strings (auch Zeichenketten genannt) werden in Visual Basic grundsätzlich in doppelte Anführungszeichen ("Gänsefüßchen") eingeschlossen.

BEISPIEL: Deklarieren und Zuweisen einer Stringvariablen

```
Dim s As String  
s = "Ich bin ein String!"
```

Strings fester Länge

Zeichenketten konstanter Länge werden mit ** Länge* deklariert.

BEISPIEL: Eine Stringvariable mit 10 Zeichen

```
Dim vorname As String * 10  
vorname = "Siegbast"
```

BEISPIEL: Quasi als Ersatz für den in VBA nicht enthaltenen Datentyp *Char* kann man ein einzelnes Zeichen darstellen.

```
Dim zeichen As String * 1  
zeichen = "A"
```

Stringaddition

Obwohl man auch mit dem arithmetischen "+"-Operator Zeichenketten verbinden kann, sollte man dafür grundsätzlich den "&"-Operator ("Kaufmännisches Und") verwenden.

BEISPIEL: Addieren von Strings.

```
Dim a As String, b As String  
a = "Access und"  
b = "VBA"  
Debug.Print a & " " & b      ' liefert "Access und VBA"
```

Beachten Sie:

- Ausdrücke, die keine Zeichenfolge sind, werden in den *Variant*-Untertyp *String* umgewandelt.
- Das Ergebnis ist vom *String*-Datentyp, wenn beide Ausdrücke den Typ *String* haben, andernfalls ist das Ergebnis ein *Variant* vom Untertyp *String*.
- Sind beide Ausdrücke *Null*, ist das Ergebnis ebenfalls *Null*. Wenn jedoch nur ein Ausdruck *Null* ist, wird dieser Ausdruck bei der Verkettung mit einem anderen Ausdruck als Leerstring ("") interpretiert. Hat ein Ausdruck den Wert *Empty*, so wird er ebenfalls als Leerstring interpretiert.

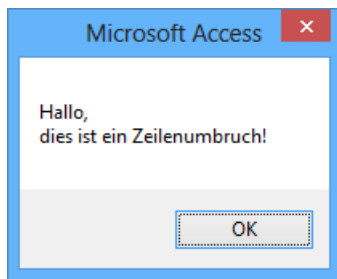
Zeilenumbruch

Um einen Zeilenumbruch innerhalb eines Strings zu erzwingen kann man die Steuerzeichen *Chr\$(10)* bzw. *Chr\$(13)* oder die VB-Konstante *vbCrLf* einfügen.

HINWEIS: Mehr zur Arbeit mit Strings lesen Sie ab Seite 110 (Zeichenkettenfunktionen).

BEISPIEL: Zeilenumbruch einfügen

```
MsgBox "Hallo," & vbCrLf & "dies ist ein Zeilenumbruch!"
```



2.2.6 Variant-Datentyp

Dieser Datentyp fällt etwas aus der Reihe, da er – einem Chamäleon gleich – beliebige Daten, mit Ausnahme von Zeichenketten (Strings) fester Länge und benutzerdefinierten Typen, enthalten kann.

Obwohl man schon in der Grundschule lernt, dass man "Äpfel und Birnen" nicht zusammenzählen darf – ein *Variant* macht's trotzdem möglich, denn wenn eine *Variant*-Variable Ziffern enthält, so können diese (je nach Zuweisung) entweder als Stringausdruck oder als Zahl interpretiert werden.

BEISPIEL: Die Anweisungsfolge:

```
Dim v As Variant
Dim st As String
Dim z As Integer

v = 5784
st = "16"
z = v + st           ' liefert die Integer-Zahl 5800
```

Arithmetische Operationen lassen sich mit allen *Variant*-Variablen durchführen, wenn diese numerische Daten enthalten oder Stringausdrücke, die als numerische Daten interpretiert werden können.

HINWEIS: Beim Umgang mit dem Datentyp *Variant* ist Vorsicht geboten, verliert man doch hier sehr leicht den Überblick. Allgemein behalten *Variant*-Variablen ihren erstmalig zugewiesenen Sub-Datentyp bei. Wenn Sie zum Beispiel einem *Variant* einen *Integer*-Wert zuweisen, interpretieren alle nachfolgenden Operationen den *Variant* als Datentyp *Integer*.

Mit den Funktionen *VarType* bzw. *TypeName* können Sie ermitteln, wie die Daten in einer *Variant*-Variablen interpretiert werden.

BEISPIEL: Das folgende kleine Testprogramm demonstriert die Verwendung der Funktionen *TypeName* (gibt eine Zeichenfolge zurück) und *VarType* (gibt einen Wert zurück).

```
Private Sub Form_Load()
' Einige Variablen deklarieren:
Dim i As Integer
```

```

Dim sng As Single
Dim st As String
Dim obj As Object
Dim v As Variant
' Tabellenkopf anfertigen:
Debug.Print "Variable", "TypeName", "VarType"
Debug.Print "-----"
' Typinfos feststellen und ausdrucken:
Debug.Print "i", TypeName(i), VarType(i)
Debug.Print "sng", TypeName(sng), VarType(sng)
Debug.Print "st", TypeName(st), VarType(st)
Debug.Print "v", TypeName(v), VarType(v)
End Sub

```

Öffnen Sie das Formular und betätigen Sie **[Sng]+[G]**. Oder schließen Sie das Formular wieder, wechseln in das Codefenster und aktivieren den Menüpunkt *Ansicht/Direktfenster*:

Variable	TypeName	VarType
i	Integer	2
sng	Single	4
st	String	8
v	Empty	0

HINWEIS: Im Übersichtsteil des Kapitels (Seite 148) finden Sie eine Zusammenstellung der Rückgabewerte der *VarType*-Funktion.

Spezielle Werte

Einer *Variant*-Variablen können auch die speziellen Werte *Empty*, *Error*, *Nothing* und *Null* zugewiesen werden:

- *Empty* kennzeichnet eine nicht initialisierte Variant-Variable (d.h., ihr wurde noch kein Wert zugewiesen). Im Zusammenhang mit Zahlenoperationen entspricht dies dem Wert 0, bei Stringoperationen dem Leerstring ("").
- *Null* zeigt an, dass die Variable vom Datentyp *Variant* absichtlich keine gültigen Daten enthält, was z.B. im Zusammenhang mit dem Auslesen von Datenbankfeldern von Bedeutung sein kann. *Null* darf nicht mit der Zahl *Null* (0) oder mit *Empty* verwechselt werden!
- *Error* dient der Kennzeichnung von Fehlerzuständen in einer Prozedur. Im Unterschied zu anderen Fehlern findet jedoch keine normale Fehlerbehandlung durch die Anwendung statt. Deshalb können Sie als Programmierer selbst den Fehler auswerten und geeignete Maßnahmen ergreifen. Ob eine *Variant*-Variable den Wert *Null* hat, kann mit der *IsNull*-Funktion getestet werden.

HINWEIS: Verwenden Sie den Datentyp *Variant* nur dort, wo es unbedingt nötig ist (Auflistungen, API, optionale Parameter, Parameter-Arrays).

Array- und IsArray-Funktionen

Mit der *Array*-Funktion können Sie einer *Variant*-Variablen eine Aufzählung zuweisen.

BEISPIEL: Geben Sie im Direktfenster nacheinander die beiden folgenden Anweisungen ein:

```
aWoche = Array("Montag","Dienstag","Mittwoch","Donnerstag","Freitag")
Debug.Print aWoche(3) ' liefert Donnerstag im Direktfenster
```

Konvertieren von Datentypen

Bei Zuweisungen für den Datentyp *Variant* ist es wichtig, Klarheit über die Interpretation des Wertes zu schaffen. Um diese und ähnliche Probleme zu lösen, wird für (fast) jeden Datentyp eine Funktion zur Verfügung gestellt, mit welcher man Umwandlungen des Ergebnistyps von Ausdrücken vornehmen kann.

BEISPIEL: Konvertierung

```
Dim datum1 As String, datum2 As Date
datum1 = "28.November 2016"
datum2 = CDate(datum1) ' ergibt für datum2 den Wert 28.11.16
```

In der folgenden Tabelle finden Sie eine Zusammenstellung typischer Konvertierungsfunktionen:

Umwandlungsfunktion	Datentyp	Umwandlungsfunktion	Datentyp
<i>CInt (Ausdruck)</i>	<i>Integer</i>	<i>CBool (Ausdruck)</i>	<i>Boolean</i>
<i>CLng (Ausdruck)</i>	<i>Long</i>	<i>CByte (Ausdruck)</i>	<i>Byte</i>
<i>CSng (Ausdruck)</i>	<i>Single</i>	<i>CDate (Ausdruck)</i>	<i>Date</i>
<i>CDBl (Ausdruck)</i>	<i>Double</i>	<i>CStr (Ausdruck)</i>	<i>String</i>
<i>CCur (Ausdruck)</i>	<i>Currency</i>	<i>CVar (Ausdruck)</i>	<i>Variant</i>

Beachten Sie bitte Folgendes:

- Das Argument *Ausdruck* kann ein beliebiger numerischer Wert oder ein String (Zeichenkette) sein.
- Zwischen der *CStr*- und *Str*-Funktion bestehen erhebliche Unterschiede. Das Gleiche gilt für *CInt*, *CLng*, *CSng* etc. bezüglich der *Val*-Funktion.
- Weitere Informationen bezüglich der Anwendung von *CDate* und *CLng* entnehmen Sie bitte den Abschnitten zu Datums- und Zeitfunktionen (siehe Seite 120) bzw. Vorzeichen- und Rundungsfunktionen (siehe Seite 116).

BEISPIEL: Anwenden einer Konvertierungsfunktion (Direktfenster).

```
? CInt(52 / 15) * 15 ' ergibt 45
```

BEISPIEL: Vergleichende Berechnungen

```
Dim i As Integer

i = 52 / 15 * 15
MsgBox i

... ergibt 52 !!!

i = (52 / 15)
i = i * 15
MsgBox i

... ergibt 45

i = (52 \ 15) * 15
MsgBox i

... ergibt 45
```

2.3 Datenfelder (Arrays)

VBA erlaubt die Deklaration statischer und dynamischer Datenfelder. Beide müssen explizit deklariert werden. Auch hierzu werden *Dim*-, *Private*-, *Public*- und *Static*-Anweisung eingesetzt.

2.3.1 Statische Arrays

Der Speicherplatzbedarf liegt bei 20 Byte für den Overhead plus vier Byte für jede Felddimension. Hinzu kommt die Anzahl von Bytes für die eigentlichen Daten.

Option Base-Anweisung

In VBA ist der untere Feldindex standardmäßig null.

BEISPIEL: Arraydefinition

```
Dim a(50) As Currency

und

Dim a@(50)

sind gleichwertige Anweisungen und definieren beide ein Array mit 51 Währungswerten, also:

a(0), a(1), a(2), ... a(50)
```

Manchmal möchte man aber, dass der unterste Feldindex mit eins beginnt. In diesem Fall entspricht die Anzahl der Feldelemente exakt der Dimension des Arrays. Um das zu erreichen, setzen Sie die *Option Base*-Anweisung ein.

BEISPIEL: Die folgenden Zeilen:

```
Option Base 1
Dim a(50) As Currency
```

definieren ein Feld (Array) mit 50 Währungswerten, d.h.:

```
a(1), a(2), a(3), ... a(50)
```

Der Zugriff auf *a(0)* verursacht dann einen Fehler.

BEISPIEL: Die Anweisungen:

```
Option Base 1
Const c1 = 10, c2 = 20
Dim b(c1,c2) As Single
```

deklarieren ein zweidimensionales Array mit *Single*-Zahlen, wobei *c(1,1)* das erste Element der ersten Zeile und *b(10,20)* das letzte Element der letzten Zeile bezeichnen. Beachten Sie:

- *Option Base* kann nur mit den Argumenten 0 (Standardwert) oder 1 aufgerufen werden.
- *Option Base* ist nur auf Modulebene anwendbar (also nicht innerhalb von Funktionen und Prozeduren!).

Bereichsgrenzen mit To

Wesentlich flexiblere Möglichkeiten zur Festlegung der Dimensionsgrenzen ergeben sich durch Verwendung von *To*:

BEISPIEL: Die Anweisung:

```
Dim a(1 To 5, 10 To 15)
```

deklariert ein zweidimensionales Array mit folgendem Aufbau:

	10	11	12	13	14	15
1	a(1,10)	a(1,11)	A(1,12)	a(1,13)	a(1,14)	a(1,15)
2	a(2,10)	a(2,11)	A(2,12)	a(2,13)	a(2,14)	a(2,15)
3	a(3,10)	a(3,11)	A(3,12)	a(3,13)	a(3,14)	a(3,15)
4	a(4,10)	a(4,11)	A(4,12)	a(4,13)	a(4,14)	a(4,15)
5	a(5,10)	a(5,11)	A(5,12)	a(5,13)	a(5,14)	a(5,15)

Private- und Public-Deklarationen

Ähnlich wie bei "normalen" Variablen können Sie auch Arrays als privat oder öffentlich deklarieren. Wenn Sie *Dim* auf Modulebene verwenden, sollten Sie es der besseren Lesbarkeit wegen durch *Private* ersetzen.

BEISPIEL: Eine Deklaration auf Modulebene

```
Private matrixA(n+1)
```

Öffentliche Deklarationen funktionieren allerdings nur in Standardmodulen (also nicht innerhalb von Form-/Berichtsmodulen und schon gar nicht innerhalb von Funktionen und Prozeduren).

BEISPIEL: Die Deklaration innerhalb eines Standardmoduls stellt das zweidimensionale *Variant*-Array *matrixB* auch allen anderen Modulen der aktuellen Datenbank zur Verfügung:

```
Public matrixB(50,z)
```

2.3.2 Dynamische Arrays

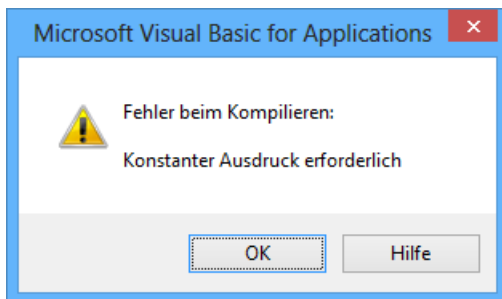
Die Anzahl der Feldelemente eines dynamischen Arrays braucht erst zur Laufzeit festgelegt zu werden und ist im Allgemeinen zur Entwurfszeit noch unbekannt. Dynamische Arrays werden unter anderem auch als Übergabeparameter in so genannten *Parameter-Arrays* eingesetzt.

ReDim-Anweisung

Die bei der normalen Array-Deklaration angegebenen Bereichsgrenzen müssen Konstanten sein, Variablen sind also unzulässig.

BEISPIEL: Die folgende Sequenz erzeugt die gezeigte Fehlermeldung.

```
Dim x As Integer
Dim y As Integer
x = 5
y = 17
Dim a(1 To x, 10 To y) As Double
```



Aber das ist noch lange kein Grund zur Resignation! Sie brauchen nur die *Dim*- (oder auch *Private*-)Anweisung auf Modulebene mit leeren Klammern auszuführen, und schon haben Sie ein dynamisches Array. Durch *ReDim*, welches allerdings nur auf Prozedurebene einsetzbar ist, erfolgt die Festlegung bzw. Änderung der Dimensionen.

BEISPIEL: Um das fehlgeschlagene Vorgängerbeispiel doch noch zum Laufen zu bringen, ist zunächst auf Modulebene Folgendes zu deklarieren:

```
Private a() As Double
Private x As Integer
Private y As Integer
```

Auf Prozedurebene werden die Dimensionen festgelegt:

```
x = 10  
y = 17  
ReDim a(1 To x, 10 To y)
```

Preserve-Option

Mit Hilfe der *Preserve*-Option kann man dafür sorgen, dass trotz mehrmaligem Umdimensionieren der alte Feldinhalt erhalten bleibt.

BEISPIEL: Die Anweisung:

```
ReDim Preserve a(1 To x+1, 10 To y)
```

fügt eine weitere Zeile zum oben deklarierten Array hinzu, ohne den alten Inhalt zu zerstören.

Beachten Sie:

- Sie können mit *Preserve* nur die obere Grenze ändern.
- Die Anzahl der Dimensionen kann mit *Preserve* nicht geändert werden.

Abfrage der Dimensionen mit LBound und UBound

Falls durch häufiges Umdimensionieren die aktuellen Abmessungen eines dynamischen Arrays nicht mehr bekannt sind, können diese mit den Funktionen *LBound* und *UBound* ermittelt werden. Außer dem Feldnamen muss diesen Funktionen der Index der abzufragenden Dimension übergeben werden.

BEISPIEL: Der Befehl:

```
UBound(a,2)
```

liefert für das Vorgängerbeispiel den Wert 17 (obere Grenze der zweiten Dimension).

Löschen von Arrays mit Erase

Wenden Sie die *Erase*-Anweisung auf ein dynamisches Datenfeld an, so wird das gesamte Array gelöscht und der belegte Speicher wieder freigegeben. Eine Neudimensionierung ist mit *ReDim* möglich.

BEISPIEL: Die folgende Anweisungsfolge löscht das Array a und legt danach die Abmessungen neu fest.

```
Erase a  
ReDim a(10,10)
```

Wird *Erase* auf ein statisches Array angewendet, so werden alle Elemente auf null (0) gesetzt. In String-Arrays werden Nullstrings ("") zugewiesen.

2.4 Benutzerdefinierte Datentypen

Neben den Standard-Datentypen können Sie auch eigene Typen erstellen.

2.4.1 Type-Anweisung

Die benutzerdefinierten Datentypen werden auch als Struktur- bzw. Verbundvariablen bezeichnet.

Die Typdefinition erfolgt mit *Type*. Die "nackte" *Type*-Anweisung (*Public*) wird nur in Standardmodulen verwendet, in einem Klassenmodul bzw. Form-/Berichtsmodul muss das Schlüsselwort *Private* vorangestellt werden.

BEISPIEL: In einer Variablen zur Erfassung von Studenten sollen Name, Immatrikulationsdatum und Höhe des Stipendiums abgespeichert werden. In einem Standardmodul definieren Sie den Typ:

```
Type TStudent
    name As String
    immatDatum As Date
    stipendium As Currency
End Type
```

Die Variablendeklaration für zwei Studenten erfolgt z.B. mit:

```
Dim student1 As TStudent, student2 As TStudent
```

Beachten Sie bei der Anwendung der *Type*-Anweisung bitte Folgendes:

- Die *Type*-Anweisung ist nur auf Modulebene zulässig (also nicht innerhalb von Funktionen und Prozeduren). Nach der Definition eines benutzerdefinierten Typs mit *Type* können Sie eine Variable dieses Typs nur innerhalb des Gültigkeitsbereichs der Definition deklarieren, und zwar mit *Dim*, *Private*, *Public*, *ReDim* oder *Static*.
- Zulässige Typen der Datenelemente sind: *Byte*, *Boolean*, *Integer*, *Long*, *Currency*, *Single*, *Double*, *Date*, *String*, *String * Länge* (für Zeichenfolgen fester Länge), *Object*, *Variant*, ein Objekttyp oder ein anderer benutzerdefinierter Typ.
- Typkennzeichen (*%*, *!* usw.) sind innerhalb von *Type* nicht erlaubt.
- Die Typdefinition erfolgt standardmäßig öffentlich (*Public* kann weggelassen werden) und ist (wie in obigem Beispiel) so nur in Standardmodulen zulässig.
- In Klassen-, Formular- und Berichtsmodulen dürfen nur private Typdefinitionen erfolgen.

HINWEIS: Ein einfaches Beispiel zu benutzerdefinierten Datentypen und dynamischen Arrays finden Sie auf Seite 128!

2.4.2 With-Anweisung

Um auf den Wert eines benutzerdefinierten Datentyps zuzugreifen, müssen Variablen- und Feldbezeichner durch einen Punkt voneinander getrennt sein.

BEISPIEL: Das Lesen bzw. Schreiben einzelner Einträge der Variablen *student1* des Vorgängerbeispiels könnte so aussehen:

```
Dim n As String, id As Date, geld As Currency
student1.name = "Sorglos"
student1.immatDatum = #11/28/72#
student1.stipendium = 880.65
n = student1.name
id = student1.immatDatum
geld = student1.stipendium
```

Das lästige Voranstellen des Variablenbezeichners können Sie sich sparen, wenn Sie sich der *With*-Anweisung bedienen:

BEISPIEL: Die Schreibweise des Vorgängerbeispiels könnte wie folgt vereinfacht werden:

```
With student1
    .name = "Sorglos"
    .immatDatum = #11/28/72#
    .stipendium = 880.65
    n = .name
    id = .immatDatum
    geld = .stipendium
End With
```

Natürlich ist es auch möglich, nicht nur auf die einzelnen Feldelemente, sondern auf die Strukturvariable insgesamt zuzugreifen:

BEISPIEL: Die folgende Anweisung "klont" die Variable *student1*.

```
student2 = student1
```

HINWEIS: Nachdem mit der Ausführung eines *With*-Blocks begonnen wurde, kann das spezifizierte Objekt nicht mehr geändert werden. Sie können daher in einer einzelnen *With*-Anweisung nicht mehrere verschiedene Objekte bearbeiten.

2.4.3 Strings innerhalb Type

In einigen Anwendungsfällen (z.B. Randomdateien) sind Strings fester Länge zu verwenden.

BEISPIEL: Verwenden der *Type*-Anweisung in einem Standardmodul, um einen benutzerdefinierten Datentyp zu erzeugen:

```
Type TMitarbeiter
    PersNr As Integer
    Name As String * 25
```

```
Adresse As String * 50
Einstellungsdatum As Date
End Type
```

Die Variablendeklaration:

```
Dim Mitarbeiter1 As TMitarbeiter
```

Das Füllen der Variablen *Mitarbeiter1*:

```
With Mitarbeiter1
.PersNr = 12345
.Name = "Hoffmann"
.Adresse = "04610 Wintersdorf Buchenring 19"
.Einstellungsdatum = 15.09.2010
End With
```

2.4.4 Enumerationen

Sammlungen von miteinander verwandten Konstanten können in so genannten *Enums* (Enumerations) zusammengefasst werden. Der *Enum*-Datentyp wird ähnlich wie ein strukturierter Datentyp deklariert.

```
[Public|Private] Enum typeName
    mitgliedsName [=konstanterAusdruck]
    mitgliedsName [=konstanterAusdruck]
    ...
End Enum
```

BEISPIEL: Eine Enumeration für drei Konstanten:

```
Public Enum erstesQuartal
    cJanuar = 1
    cFebruar
    cMärz
End Enum
```

Es genügt, wenn nur der ersten Mitgliedskonstanten (*cJanuar*) ein (*Long*-)Wert zugewiesen wird, der Wert der Nachfolger wird automatisch um 1 erhöht.

Auf die deklarierten Konstanten kann direkt zugegriffen werden.

BEISPIEL: Verwendung der oben deklarierten Enumeration:

```
Dim m As Long
m = cFebruar           ' m erhält den Wert 2

oder ausführlicher

m = erstesQuartal.cFebruar
```

BEISPIEL: Auch Variablen vom Datentyp einer *Enum* sind möglich.

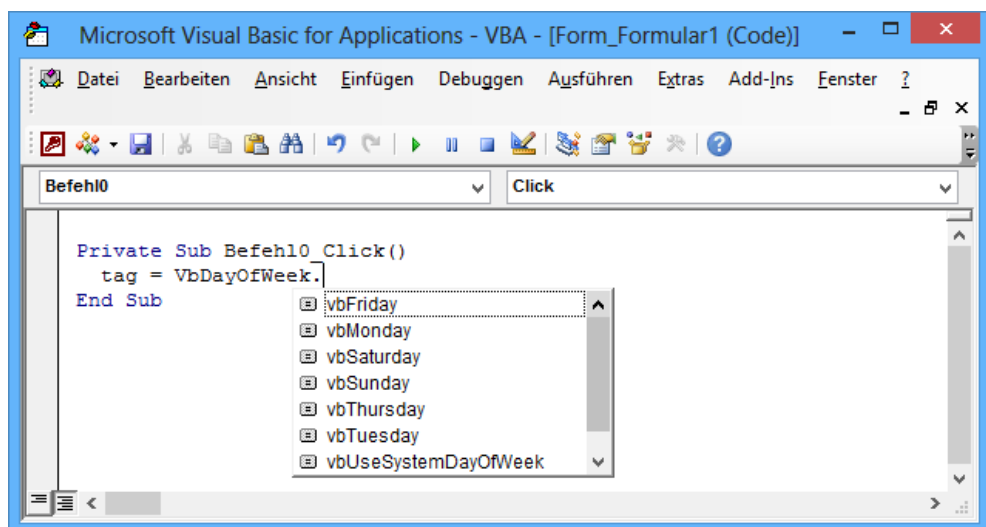
```
Dim monat As erstesQuartal  
monat = cFebruar
```

BEISPIEL: Auch einer Prozedur können Sie einen *Enum*-Parameter übergeben:

```
Private Sub listenEintrag(mName$, monat As erstesQuartal)  
    List1.AddItem mName & " = " monat  
End Sub
```

HINWEIS: Das Zuweisen negativer *Long*-Werte ist z.B. für Enumerationen mit Fehlerkonstanten üblich!

VBA hat zahlreiche "eingebaute" Enumerationen, z.B. *vbDayOfWeek*. Nutzen Sie die IntelliSense des Quelltexteditors, um die verfügbaren Mitgliedskonstanten zu "besichtigen":



2.4.5 Arrays in benutzerdefinierten Typen

Statische Datenfelder

Wenn Sie ein Datenfeld fester Größe innerhalb eines benutzerdefinierten Typs deklarieren, müssen die Dimensionen mit Zahlen oder Konstanten angegeben werden. Variablen sind an dieser Stelle unzulässig.

BEISPIEL: Gezeigt wird die Verwendung eines zweidimensionalen Datenfeldes fester Größe (*m*) in einem benutzerdefinierten Typ (*TMyType*), der selbst wiederum als Vorlage für ein eindimensionales Datenfeld (*arr*) dient.

In den Deklarationsabschnitt eines *Form*-Moduls kopieren Sie folgenden Code:

```
Const a = 10, b = 20

Private Type TMyType
    m(1 To a, 2 To b) As Integer    ' statisches Array
    ...
End Type
Dim arr(1 To 100) As TMyType
```

Auf Prozedurebene wäre dann z.B. folgende Zuweisung möglich:

```
arr(90).m(3,14) = 10000
```

Dynamische Datenfelder

Es spricht für die Leistungsfähigkeit der *Type*-Anweisung, dass in ihr auch dynamische Arrays definiert sein dürfen, deren Dimensionen während der Laufzeit verändert werden können.

BEISPIEL: Mit folgendem Quellcode wird das gleiche Ergebnis wie im Vorgängerbeispiel erreicht.

```
Dim a As Integer, b As Integer

Private Type TMyType
    m() As Integer                ' dynamisches Array
    ...
End Type

Dim arr(1 To 100) As TMyType
```

Die Veränderung der Felddimension geschieht auf Prozedurebene und bezieht sich immer nur auf ein einzelnes Element der Variablen *arr*:

```
a=10
b=20
ReDim arr(90).m(1 To a, 2 To b)
arr(90).m(3,14) = 10000
```

HINWEIS: Wenn nicht explizit eine Untergrenze für das in *Type* definierte Datenfeld angegeben wird (wie in den Beispielen dieses Abschnitts), so wird die Untergrenze durch die *Option Base*-Anweisung gesteuert.

2.5 Operatoren

VBA unterscheidet zwischen vier Typen von Operatoren:

- Arithmetische Operatoren (für mathematische Berechnungen)
- Logische Operatoren (für logische Vergleiche)

- Vergleichsoperatoren (für sonstige Vergleiche)
- Verkettungsoperatoren (zum Aneinanderhängen von Zeichenfolgen)

Die Wirkung der einzelnen Operatoren können Sie bequem im Direktfenster nachvollziehen, welches Sie beispielsweise mit **Strg**+**G** aufrufen:

```

Direktbereich
Print 3*2.5^3
46,875
? 10\4
2
? (10>8)
Wahr

```

HINWEIS: Als Abkürzung für die *Print*-Anweisung können Sie das Fragezeichen (?) verwenden.

2.5.1 Arithmetische Operatoren

Folgende Operatoren steht zur Verfügung:

Operator	Syntax	Erklärung
^	Ergebnis = Zahl ^ Exponent	Potenzieren
+	Ergebnis = Operand1 + Operand2	Addition
–	Ergebnis = Operand1 – Operand2 –Zahl	Subtraktion Negatives Vorzeichen
*	Ergebnis = Operand1 * Operand2	Multiplikation
/	Ergebnis = Operand1 / Operand2	Gleitkomma-Division
\	Ergebnis = Operand1 \ Operand2	Integer-Division
Mod	Ergebnis = Operand1 Mod Operand2	Modulo (Rest aus Division)

HINWEIS: Der Operator "+" kann auch zur Stringverkettung eingesetzt werden, aber Sie vermeiden Mehrdeutigkeiten, wenn Sie dazu grundsätzlich den Operator "&" verwenden.

Bei *Null*- (ungültig) und *Empty*-Werten (leer) gilt: Wenn mindestens ein Ausdruck ein *Null*-Ausdruck ist, enthält das Ergebnis ebenfalls *Null*. Wenn ein Ausdruck *Empty* ist, so wird er als Zahl 0 interpretiert.

Bezüglich der zahlreichen Sonderfälle und Abweichungen von dieser Regel (insbesondere im Zusammenhang mit *Variant*-Datentypen) informieren Sie sich bitte in der Online-Hilfe.

BEISPIEL: Addition unterschiedlicher Datentypen:

```
a = "24": b = 7
c = a + b
? c           ' liefert 31
```

BEISPIEL: Integer-Division

```
? 10 \ 4      ' liefert 2
```

BEISPIEL: Mod-Division

```
? 25 Mod 5    ' ergibt 0
? 15 Mod 5.3   ' ergibt 0
? 12.6 Mod 5   ' liefert 3
```

BEISPIEL: Potenzieren

```
Wert1 = 2 ^ 2 ^ 3      ' liefert 64
a = (-5) ^ 3           ' liefert -125
```

BEISPIEL: Quadratwurzel

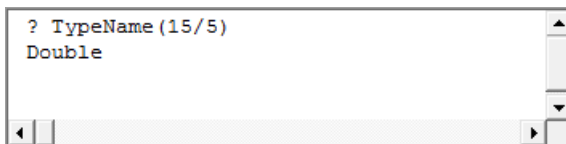
```
a = 2 ^ -0.5           ' liefert 0,7071068
```

BEISPIEL: Kubikwurzel

```
a = 1/3
b = 2 ^ a              ' weist b den Wert 1,259921 zu (= dritte Wurzel aus 2)
```

Ergebnistyp bei Division

Bei der Gleitkommadivision (/) ist der zurückgegebene Wert immer vom Datentyp *Double*, gleichgültig, ob es sich bei den Operanden um Gleitkomma- oder Integer-Zahlen handelt.



Die Modulo-Division (*Mod*) liefert hingegen immer einen Integer-Wert als Ergebnis.

Zusammenhang zwischen Modulo- und Integer-Division

Es gilt die allgemeine Beziehung:

$$(a \setminus b) * b + a \text{ Mod } b = a$$

Obige Formel gilt aber nur, wenn *a* und *b* Integer-Zahlen sind!

Die Überprüfung obiger Formel im Direktfenster für $a = 12$ und $b = 5$ bestätigt die Richtigkeit:

```
a = 10
b = 5
? (a \ b) * b + a Mod b = a
Wahr
```

2.5.2 Logische Operatoren

Folgende Operatoren für die logische Verknüpfung von Ausdrücken bzw. die bitweise Verknüpfung von Zahlen stehen zur Verfügung:

Operator	Bedeutung	Operator	Bedeutung
Not	NEGATION	Xor	EXKLUSIV ODER
And	UND	Eqv	ÄQUIVALENZ
Or	ODER	Imp	IMPLIKATION

Da der Wert *Null* (ungültiger Ausdruck) ebenfalls beim Vergleich von Ausdrücken eine Rolle spielt, ergibt sich eine Art "dreiwertige Logik". Die folgende Tabelle zeigt eine entsprechende Zusammenstellung für die vier wichtigsten Operatoren:

Operator	Verknüpfen von Ausdrücken			Bitweise Verknüpfung		
	Ausdruck1	Ausdruck2	Ergebnis	Bit1	Bit2	Ergebnis
Not	True	–	False	0	–	1
	False	–	True	1	–	0
	Null	–	Null	–	–	–
And	False	False	False	0	0	0
	False	True	False	0	1	0
	True	False	False	1	0	0
	True	True	True	1	1	1
	True	Null	Null	–	–	–
	Null	False	False	–	–	–
	Null	Null	Null	–	–	–
Or	False	False	False	0	0	0
	False	True	True	0	1	1
	True	False	True	1	0	1
	True	True	True	1	1	1
	True	Null	True	–	–	–
	Null	False	Null	–	–	–
	Null	Null	Null	–	–	–

Operator	Verknüpfen von Ausdrücken			Bitweise Verknüpfung		
Xor	False	False	False	0	0	0
	False	True	True	0	1	1
	True	False	True	1	0	1
	True	True	False	1	1	0
	True	Null	Null	–	–	–
	False	Null	Null	–	–	–
	Null	Null	Null	–	–	–

BEISPIEL: Ausdrucksvergleiche:

Not (10 > 8)

' liefert False

Not (6 > Null)

' liefert Null

8 > 10 Eqv 8 > 6

' liefert False

True And True Or False And False

' liefert True

BEISPIEL: Bitweise Vergleiche:

Not (0)

' liefert -1

Not (20)

' liefert -21

8 And 10

' liefert 8, weil

1000 (binäre 8)

And 1010 (binäre 10)

1000 (binäre 8)

8 Xor 10

' liefert 2, da

1000 (binäre 8)

Xor 1010 (binäre 10)

0010 (binäre 2)

2.5.3 Vergleichsoperatoren

Diese dienen zum Vergleich von Ausdrücken, die sowohl Zahlen als auch Zeichenketten enthalten können.

Operator	Erklärung	True, wenn	False, wenn	Null, wenn
<	Kleiner als	a < b	a >= b	a oder b = Null
>	Größer als	a > b	a <= b	a oder b = Null
=	Gleich	a = b	a <> b	a oder b = Null
<=	Kleiner oder gleich	a <= b	a > b	a oder b = Null
>=	Größer oder gleich	a >= b	a < b	a oder b = Null
<>	ungleich	a <> b	a = b	a oder b = Null

Bemerkungen

- Wird ein Wert vom Typ *Currency* mit einem Wert vom Typ *Single* oder *Double* verglichen, so wird letzterer in *Currency* umgewandelt. Dabei gehen die fünfte und alle weiteren Nachkommastellen des *Single*- oder *Double*-Wertes verloren. Vorsicht: Zwei in Wirklichkeit verschiedene Werte können damit als gleich interpretiert werden!
- Beim Vergleich eines Werts vom Typ *Single* mit einem Wert vom Typ *Double* wird letzterer auf *Single*-Genauigkeit gerundet.
- In Klammern gesetzte Operationen haben grundsätzlich Vorrang (Taschenrechnerprinzip!).

2.6 Kontrollstrukturen

Mit diesen Anweisungen unterbrechen Sie den linearen Programmablauf, indem – in Abhängigkeit von bestimmten Bedingungen – "Weichen" gestellt oder Schleifen durchlaufen werden.

2.6.1 Bedingte Verzweigungen

Verzweigung	Erläuterungen
If <i>Bedingung</i> Then <i>Anweisungen</i> [Else <i>Anweisungen</i>]	Bedingte Verzweigung (muss in einer Zeile stehen!)
If <i>Bedingung1</i> Then <i>Anweisungen</i> [ElseIf <i>Bedingung2</i> Then <i>Anweisungen</i> ElseIf <i>Bedingung3</i> Then <i>Anweisungen</i>] [Else <i>Anweisungen</i>] End If	Blockstruktur <i>If...ElseIf...End If</i> Jede Zeile muss mit <i>Then</i> enden. <i>Else</i> -Anweisungen werden nur dann ausgeführt, wenn keine der <i>If</i> - bzw. <i>ElseIf</i> -Bedingungen zutreffen.
Select Case <i>Ausdruck</i> Case <i>Ausdruck1</i> <i>Anweisungen</i> [Case <i>Ausdruck2</i> <i>Anweisungen</i>] [Case Else <i>Anweisungen</i>] End Select	Blockstruktur <i>Select Case...Case Else...End Select</i> Der Ausdruck kann eine Variable oder ein beliebiger Ausdruck sein, der mit den hinter <i>Case</i> angeführten Ausdrücken verglichen wird.
Choose (<i>Index</i> , <i>Ausdruck1</i> [, <i>Ausdruck2</i> ...])	<i>Choose</i> -Funktion Abhängig vom Index <i>i</i> wird der <i>i</i> -te Ausdruck zurückgegeben.
IIf (<i>Ausdruck</i> , <i>True-Wert</i> , <i>False-Wert</i>)	<i>IIf</i> -Funktion Ist Ausdruck wahr, wird <i>True</i> zurückgegeben, sonst <i>False</i> .
Switch (<i>Ausdruck1</i> , <i>Var1</i> [<i>Ausdruck2</i> , <i>Var2</i>])	<i>Switch</i> -Funktion Wenn Ausdruck1 wahr ist, wird <i>Var1</i> zurückgeliefert usw. (max. 7 Einträge)

BEISPIEL: *If*-Verzweigungen

```
If note = 1 Then
    ? "Gratuliere!"
Else
    ? "Verbessern!"
End If
```

```
If note = 1 Then
    ? "Sehr gut!"
Elseif note = 2 Then
    ? "Gut"
Elseif note = 3 Then
    ? "Befriedigend"
' usw.
End If
```

BEISPIEL: *Select Case*-Verzweigungen

```
Select Case note
    Case 1: ? "Sehr gut"
    Case 2: ? "Gut"
    Case 3: ? "Befriedigend"
' usw.
End Select
```

```
Select Case monat
    Case 3,4,5: ? "Frühling"
    Case 6,7,8: ? "Sommer"
    Case 9,10,11: ? "Herbst"
    Case 12,1,2: ? "Winter"
Case Else
    ? "Ungültiger Monat!"
End Select
```

```
Select Case monat
    Case 3 To 5: ? "Frühling"
    Case Is < 4: ? "Erstes Quartal"
    Case Is > 9: ? "Viertes Quartal"
' usw.
End Select
```

Im letzten Beispiel wurde auch von der Bereichseingrenzung mit *To* und dem relativ selten benutzten *Is*-Operator Gebrauch gemacht.

2.6.2 Schleifenanweisungen

Hierzu gibt es fünf Grundkonstruktionen, die wohl kaum noch Wünsche offen lassen dürften:

Schleifenanweisung	Erläuterungen
For <i>Zähler = Anfang To Ende</i> [<i>Step Schritt</i>] <i>Anweisungen</i> [Exit For] <i>Anweisungen</i> Next [<i>Zähler</i>]	<i>For...Next</i> -Zählschleife vorzeitiger Abbruch mit <i>Exit For</i> ohne <i>Step</i> ist Schritt gleich 1
While <i>Bedingung</i> <i>Anweisungen</i> Wend	<i>While...Wend</i> -Bedingungsschleife gleichbedeutend mit <i>Do While...Loop</i>
For Each <i>Element In Datenfeld</i> <i>[Anweisungen]</i> [Exit For] <i>[Anweisungen]</i> Next [<i>Element</i>]	<i>For Each...Next</i> -Durchlaufschleife wiederholt eine Reihe von Anweisungen für alle Elemente eines Datenfeldes (bzw. einer Auflistung von Objekten)
Do [While Until <i>Bedingung</i>] <i>Anweisungen</i> [Exit Do] <i>Anweisungen</i> Loop	<i>Do While...Loop</i> -Bedingungsschleife Abbruchbedingung am Schleifenanfang
Do <i>Anweisungen</i> [Exit Do] <i>Anweisungen</i> Loop [While Until <i>Bedingung</i>]	<i>Do...Loop While</i> -Bedingungsschleife Abbruchbedingung am Schleifenende

Das vorzeitige Verlassen einer Schleife ist mit den Anweisungen *Exit For* bzw. *Exit Do* möglich.

BEISPIEL: Alle folgenden sieben Schleifenkonstruktionen führen zum gleichen Ergebnis. Im Direktfenster wird achtmal untereinander *Achtung!* ausgedruckt.

```

For i = 1 To 8
    Debug.Print "Achtung!"
Next i
i = 1
Do While i <= 8
    Debug.Print "Achtung!"
    i = i + 1
Loop

```

```
i = 1
Do Until i > 8
    Debug.Print "Achtung!"
    i = i + 1
Loop

i = 1
Do
    Debug.Print "Achtung!"
    i = i + 1
Loop Until i > 8

i = 1
Do
    Debug.Print "Achtung!"
    If i = 8 Then Exit Do
    i = i + 1
Loop

i = 1
While i <= 8
    Debug.Print "Achtung!"
    i = i + 1
Wend
```

For Each...Next-Durchlaufschleife

Diese Form der Schleifenbildung fällt etwas aus dem Rahmen, da sie nur für Datenfelder (Arrays) und Auflistungen von Objekten (*Collection*, siehe Kapitel 6) eingesetzt wird.

BEISPIEL: Kopieren Sie den folgenden Code in die Prozedur des *Click*-Ereignisses einer Schaltfläche:

```
Dim a(4) As String      ' statisches Array
Dim m As Variant
a(1) = "Müller"
a(2) = "Schultze"
a(3) = "Lustig"
a(4) = "Viertel"

For Each m In a
    If m = "Lustig" Then m = "Heinrich"
    Beep
    Exit For
Next m

MsgBox m                ' zeigt "Heinrich" an
MsgBox a(3)             ' zeigt "Lustig" an
```

Wie man sieht, kann die Zählvariable *m* nur gelesen werden, ein Überschreiben von Feldelementen durch Neuweisung von *m* ist nicht möglich.

Beachten Sie Folgendes:

- Die Anweisungen werden für jedes Element im Array ausgeführt.
- *Exit For* kann an beliebiger Stelle und beliebig oft in der Schleife verwendet werden und bewirkt einen sofortigen Abbruch. Es wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel *If...Then*) eingesetzt.
- Sie können *For Each...Next*-Schleifen verschachteln, die Zählvariable *Element* muss jedoch für alle Schleifen eindeutig sein.
- Die Angabe der Zählvariablen *Element* nach *Next* kann auch entfallen.

2.6.3 GoTo und GoSub

Beide Anweisungen führen Sprungbefehle zu einer Marke aus. Während *GoTo* eine einfache unbedingte Verzweigung darstellt, ruft *GoSub* ein Unterprogramm auf, welches mit *Return* abzuschließen ist.

BEISPIEL: *GoTo*

```
If k > 5 Then GoTo markel
Debug.Print "Das ist der Code für k <= 5 !"
Exit Sub
markel:
Debug.Print "Das ist der Code für k > 5 !"
Exit Sub
```

BEISPIEL: *GoSub*

```
If k > 5 Then GoSub markel Else Debug.Print "Das ist der Code für k <= 5 !"
Exit Sub
markel:
    Debug.Print "Das ist ein Unterprogramm für k > 5!"
Return
```

Im Zusammenhang mit dem Schlüsselwort *On* können *GoTo* und *GoSub* auch bedingte Sprunganweisungen ausführen.

BEISPIEL: *On...GoTo*

```
On monat GoTo m1, m2, m3      ' usw.
m1:
    Debug.Print "Januar": Exit Sub
m2:
    Debug.Print "Februar": Exit Sub
' usw.
```

HINWEIS: Beachten Sie den obligatorischen Doppelpunkt (:) nach einer Sprungmarke. Sie sollten deshalb in dieser Zeile auf Mehrfachanweisungen (die ja bekanntlich ebenfalls durch Doppelpunkte voneinander getrennt werden) verzichten. Andernfalls kann es im Zusammenhang mit verkürzten Prozeduraufrufen zu böartigen und schwer auffindbaren Laufzeitfehlern kommen.

Bemerkungen

- Der hemmungslose Gebrauch von *GoTo* und *GoSub* war nicht ganz schuldlos am schlechten Ruf des klassischen Basic. Will man strukturiert und übersichtlich programmieren, sollte man weitestgehend auf *GoTo* bzw. *GoSub* verzichten und stattdessen blockorientiert, also mit Schleifenanweisungen, arbeiten.
- Allerdings können konsequente Gegner von *GoTo* und *GoSub* dann nicht ganz ernst genommen werden, wenn sie allzu freizügig von Anweisungen wie *Exit Sub*, *Exit Function*, *Exit For* und *Exit Do* Gebrauch machen, dies sind ja auch nichts anderes als "verkappte" Sprungbefehle zum Blockende.
- Sinnvoll und in Maßen angewendet, können *GoTo* und *GoSub* durchaus ihren Beitrag zu einem effektiven Programmcode leisten (z.B. für Fehlerbehandlungsroutinen).

2.7 Zeichenkettenfunktionen

VBA bietet eine Vielzahl von Funktionen zur Bearbeitung bzw. Formatierung von Zeichenketten (Strings). Die meisten dieser Funktionen trifft man in zweifacher Ausführung an, nämlich mit oder ohne angehängtes \$-Zeichen, z.B. *Left\$* und *Left*.

HINWEIS: Wenn das \$ am Ende einer Zeichenkettenfunktion fehlt, so hat deren Rückgabewert den Datentyp *Variant*, andernfalls den Datentyp *String*.

2.7.1 Stringverarbeitung

Übersicht

Die folgende Tabelle gibt einen Überblick über die wichtigsten Zeichenkettenfunktionen:

Funktion/Anweisung	Erläuterung
<i>Asc(string)</i>	Liefert den ANSI-Code des ersten Zeichens
<i>Chr\$(code)</i>	Liefert das Zeichen für ANSI-Code
<i>CStr(zahl)</i>	Konvertiert <i>zahl</i> in einen String
<i>Format\$(ausdruck, Format-string)</i>	Formatiert einen Ausdruck
<i>InStr([start], string, string1)</i>	Liefert Position des ersten Vorkommens von <i>string1</i> innerhalb <i>string</i> ab <i>start</i>

Funktion/Anweisung	Erläuterung
InStrRev ([start], string, string1)	Wie <i>InStr</i> , allerdings mit umgekehrter Suchrichtung
Join (array [, trenn])	Liefert eine Zeichenkette aus <i>array</i> (mit Trennzeichen <i>trenn</i>)
Len (string)	Liefert die Zeichenanzahl von <i>string</i>
LCase (string); UCase (string)	Konvertiert <i>string</i> in Klein- bzw. in Großbuchstaben
Trim (string); LTrim (string); RTrim (string)	Entfernt alle führenden und angehängten Leerzeichen oder nur die führenden bzw. angehängten Leerzeichen aus <i>string</i>
Mid (string, start, [n]); Left (string, n); Right (string, n)	Liefert <i>n</i> Zeichen ab <i>start</i> oder die <i>n</i> von links beginnenden bzw. <i>n</i> von rechts endenden Zeichen (ohne <i>n</i> werden alle Zeichen geliefert)
Mid (string1, start, [n]) = string2	Ersetzt <i>n</i> Zeichen von <i>string1</i> ab <i>start</i> durch <i>string2</i>
Replace (string, x, y)	Ersetzt alle Zeichen <i>x</i> durch <i>y</i>
Space (n)	Liefert <i>n</i> Leerzeichen
Split (string[, trenn[, limit[, compare]]])	Zerlegt <i>string</i> an den Trennzeichen <i>trenn</i> und liefert nullbasiertes Array
Str (zahl)	Konvertiert eine Zahl in einen String (mit führendem Leerzeichen!)
String (n, code string)	Liefert <i>n</i> Zeichen von <i>code</i> oder von dem führenden Zeichen von <i>string</i>
StrReverse (string)	Liefert String mit umgekehrter Reihenfolge der Zeichen
Val (string)	Liefert den numerischen Wert einer Zeichenfolge (als <i>Double</i> -Zahl)

Einige Beispiele

BEISPIEL: Im Direktfenster können Sie sich leicht von der Richtigkeit der folgenden Codezeilen überzeugen:

```
? Asc("A")      ' liefert 65
? Chr$(65)      ' liefert "A"
? Val("27")     ' liefert 27
? Str$(27)      ' liefert " 27" (mit einem führenden Leerzeichen!)
```

BEISPIEL: Unter der Annahme, dass die Stringvariable *txt* den Wert "Access" hat, gilt:

```
? UCase$(txt)      ' liefert "ACCESS"
? Len(txt)         ' liefert 6
? Left$(txt, 2)    ' liefert "Ac"
? Mid$(txt, 3, 4)   ' liefert "cess"
? String$(5, txt)   ' liefert "AAAAA"
? InStr(txt, "e")   ' liefert 4
? Replace(txt, "c", "b") ' liefert "Abbess"
? Space(5) & txt    ' liefert "    Access"
```

BEISPIEL: Die ersten zehn Zeichen eines Strings werden beseitigt.

```
Dim s1 As String
s1 = "Das Große Access-Buch"
s1 = Mid$(s1,11)           ' "Access-Buch"
```

BEISPIEL: Ein String wird am Trennzeichen "-" in Teilstrings zerlegt und anschließend (ohne Trennzeichen) wieder zusammengefügt.

```
Dim a() As String           ' dynamisches Array (unterster Feldindex = 0 !)
Dim s As String
s = "Alle Vögel - sind - schon da!"
a = Split(s, "-")
MsgBox a(2)                 ' "schon da!"
s = Join(a, "")
MsgBox s                     ' "Alle Vögel sind schon da!"
```

Bemerkungen

- Der (optionale) *start*-Parameter der *Instr*-Funktion beginnt immer mit 1.
- Vorsicht beim Umgang mit der *Val*-Funktion, denn sie berücksichtigt nicht die Ländereinstellungen.

2.7.2 Format-Funktion

Mit dieser leistungsfähigen Funktion können Sie die unterschiedlichsten Datentypen in eine Zeichenkette verwandeln. Aufgrund der großen Vielfalt an Formatierungsstrings ergeben sich verschiedenste Darstellungsmöglichkeiten für Zahlen, Strings, Währungen und Datums-/Zeitangaben. Wir werden hier nur die wichtigsten Formatierungsstrings vorstellen (mehr dazu siehe Online-Hilfe).

Die Syntax:

```
Format(Ausdruck[, Formatstring])
```

Wird der Formatstring weggelassen, so liefert die *Format*-Funktion dasselbe Ergebnis wie die *Str*-Funktion. Allerdings fehlt bei positiven Zahlen das (ansonsten für das Vorzeichen reservierte) führende Leerzeichen.

BEISPIEL: Ein Vergleich zeigt den Unterschied.

```
? Str$(35)           ' liefert " 35" (man beachte das Leerzeichen am Anfang!)
? Format$(35)         ' ergibt "35"
```

Standardformate

Für die am häufigsten benötigten Zahlen-Formatierungen stellt VBA eine Reihe von vordefinierten Stringkonstanten bereit:

Formatstring	Erläuterung
<i>Standard</i>	Darstellung mit Tausender-Trennzeichen (.) und mindestens zwei Nachkommastellen
<i>General Number</i>	Normaldarstellung, ohne Tausender-Trennzeichen (.)
<i>Fixed</i>	Darstellung mit mindestens einer Vor- und mindestens zwei Nachkommastellen
<i>Currency</i>	Währungsformat
<i>Percent</i>	Prozentdarstellung
<i>Scientific</i>	Wissenschaftliche Notation
<i>Yes/No</i>	0 → Nein, sonst Ja
<i>True/False</i>	0 → Falsch, sonst Wahr
<i>On/Off</i>	0 → Aus, sonst Ein

BEISPIEL: Währungsformatierung

```
Dim geld As Double
```

```
geld = 12.6857
```

```
MsgBox Format$(geld, "Currency") ' ergibt "12.69 Euro"
```

Vordefinierte Datum-Zeit-Formatierungen

Auch für die Anzeige von *Date*-Variablen (siehe Seite 84) gibt es eine Reihe vorgefertigter Formate.

Formatname	Erläuterung
<i>General Date</i>	Darstellung einer <i>Date</i> -Variablen als Datum und/oder Zeit. Für reale Zahlen werden Datum und Zeit angezeigt, z.B. <i>06.05.2016 16:28:37</i> Gibt es keine Nachkommastellen, so wird nur ein Datum angezeigt, z.B. <i>06.05.2016</i> . Steht vor dem Komma eine Null, so wird nur die Zeit angezeigt, z.B. <i>16:28:37</i> .
<i>Long Date</i>	Darstellung entsprechend des langen Datumsformats des Systems, z.B. <i>Montag, 6.Mai 2016</i>
<i>Medium Date</i>	Darstellung entsprechend des Medium-Formats, z.B. <i>06. Mai 16</i>
<i>Short Date</i>	Darstellung entsprechend des kurzen Datumsformats, z.B. <i>06.05.2016</i>
<i>Long Time</i>	Zeitanzeige entsprechend des langen Zeitformats des Systems (Stunden, Minuten, Sekunden), z.B. <i>16:28:37</i> .
<i>Medium Time</i>	Zeitanzeige im 12-Stunden-Format (Stunden, Minuten, AM/PM), z.B. <i>04:28 PM</i> .
<i>Short Time</i>	Zeitanzeige im 24-Stunden-Format, z.B. <i>16:28</i> .

BEISPIEL: Testen Sie im Direktfenster:

```
? Format$(41343.951, "General Date")
```

```
10.03.2016 22:49:26
```

```
? Format$(41343, "General Date")
```

```
10.03.2016
```

```
? Format$(0.951, "General Date")
22:49:26
```

Nutzerdefinierte Datum-/Zeit-Formate

Komfortable Möglichkeiten ergeben sich mit den Formatstrings der folgenden Tabelle.

Format-string	Erläuterung
c	Siehe <i>General Date</i>
d oder dd	Anzeige des Tags als Zahl ohne (1 – 31) oder mit führender Null (01 – 31).
ddd	Anzeige des Wochentags als Abkürzung (So, Mo, Di, Mi, Do, Fr, Sa).
dddd	Anzeige des Wochentags mit vollem Namen (Sonntag – Sonnabend).
w	Anzeige des Wochentags als Zahl (1 für Sonntag bis 7 für Sonnabend).
ww	Anzeige der Woche im Jahr als Zahl (1 – 54).
m oder mm	Anzeige des Monats als Zahl ohne (1 – 12) oder mit führender Null (01 – 12). Achtung: Wenn <i>m</i> unmittelbar auf <i>h</i> oder <i>hh</i> folgt, wird anstatt des Monats die Minute angezeigt.
mmm	Anzeige des Monats als Abkürzung (Jan – Dez).
mmmm	Anzeige des vollen Monatsnamens (Januar – Dezember).
q	Anzeige des Quartals (1 – 4).
y	Anzeige des Tags im Jahr als Zahl (1 – 366).
yy	Zweistellige Anzeige der Jahreszahl (00 – 99).
yyyy	Vierstellige Anzeige der Jahreszahl (100 – 9999).
H oder Hh	Stundenanzeige ohne (0 – 23) oder mit führender Null (00 – 23).
N oder Nn	Minutenanzeige ohne (0 – 59) oder mit führender Null (00 – 59).
S oder Ss	Sekundenanzeige ohne (0 – 59) oder mit führender Null (00 – 59).
tttt	Komplette Zeitanzeige (Stunden, Minuten, Sekunden) unter Verwendung des in den Systemeinstellungen festgelegten Trennzeichens.

BEISPIEL: Tests im Direktfenster (angenommen, es ist der 15. Januar 2016)

```
? Format$(Date,"d/m/yy")      ' ergibt z.B. 15.1.16
? Format$(Date,"d-mmmm-yy")    ' ergibt z.B. 15-Januar-16
```

BEISPIEL: Aktuelles Datum in einem Textfeld anzeigen

```
Text1.Value = Format$(Now, "ddd, dd/mm/yy hh:mm")
```

Das Ergebnis könnte so aussehen:

Mi, 23.3.16 17:00

Eine alternative Realisierung wäre mittels der *Format*-Eigenschaft möglich:

```
Text1.Format = "ddd, dd/mm/yy hh:mm"
Text1.Value = Now
```

HINWEIS: Weitere Varianten der Formatierung von Datum und Zeit finden Sie ab Seite 120 (Datums-/Zeitfunktionen).

Nutzerdefinierte Formate

Fast beliebige formatierte Anzeigen kann man sich unter Verwendung der in folgender Tabelle gezeigten Formatzeichen selbst "zusammenbasteln":

Zeichen	Erläuterung
"" (Leerstring)	formatfreie Zahlenausgabe
0	Platzhalter, zeigt die Zahl 0 oder ein anderes Zeichen
#	Platzhalter, zeigt nichts oder ein Zeichen
. (Punkt)	Dezimaltrennzeichen, bestimmt die Vor- und Nachkommastellen
%	Prozentplatzhalter, multipliziert die Anzeige mit 100
Zeichen	Erläuterung
, (Komma)	Tausender-Trennzeichen, für Zahlen mit vier und mehr Stellen vor dem Dezimaltrennzeichen
E- E+ e- e+	wissenschaftliches Format
- + \$ () Leerzeichen	Diese Zeichen werden direkt angezeigt.
\	Das nachfolgende Zeichen wird angezeigt.

BEISPIEL: Testen Sie im Direktfenster:

```
? Format$(334.9, "##0.00")           ' liefert 334,90 (Punkt im Formatstring wird zum Komma!)
? Format$(334.9, "#0.00E+00")         ' ergibt 33,49E+01.
? Format$(Date, "dddd, \d\e\n dd.mm.yyyy") ' liefert z.B. "Montag, den 25.01.2016"
? Format$(Date, "\#m\d\yyyy\#")       ' liefert z.B. "#1/25/2016#" (englisches Datumsliteral!)
? Format$(Now, "\#m\d\yy hh:mm\#")    ' ergibt z.B. "#1/25/16 15:25#"

```

FormatNumber-, FormatPercent- und FormatCurrency-Funktion

Hier handelt es sich um spezialisierte Formatierungs-Funktionen, die meist noch über weitere (optionale) Argumente verfügen (siehe Online-Hilfe). Falls Sie eine Zahl mit dem Punkt (.) als Tausender-Trennzeichen formatieren wollen, können Sie die *FormatNumber*-Funktion verwenden.

BEISPIEL: Die Zahl 12045,23 wird mit drei Nachkommastellen formatiert.

```
? FormatNumber(12045.23, 3)           ' liefert "12.045,230"
```

Mit der *FormatPercent*-Funktion lässt sich eine Zahl als Prozentzahl ausdrücken (standardmäßig mit zwei Nachkommastellen).

BEISPIEL: Wie Sie sehen, kann man im Argument auch rechnen.

```
? FormatPercent(3/10 + 0.1)           ' liefert "40,00%"
```

Die *FormatCurrency*-Funktion formatiert eine Zahl in der Währung, wie sie in der Systemsteuerung ("Ländereinstellungen") eingestellt wurde (für uns in der Regel *Euro*).

BEISPIEL: Die Zahl 1205.36 wird in *Euro* ausgegeben.

? FormatCurrency(1205.36) ' liefert "1.205,36 Euro"

2.8 Vordefinierte Funktionen

VBA stellt für Sie ein Sammelsurium vordefinierter Funktionen bereit, von denen wir hier nur die wichtigsten aufführen wollen.

2.8.1 Mathematische Funktionen

Die Sammlung an wissenschaftlichen Funktionen nimmt sich relativ bescheiden aus, dürfte aber den Alltagsproblemen des "normalen" Datenbankprogrammierers durchaus gewachsen sein.

Funktion	Erklärung
<i>Abs(x)</i>	Liefert den Absolutwert der Zahl x
<i>Atn(x)</i>	Berechnet den Arcustangens der Zahl x im Bogenmaß
<i>Cos(x)</i>	Berechnet den Cosinus eines Winkels x, der im Bogenmaß vorliegt
<i>Exp(x)</i>	Liefert die x-te Potenz zur Basis e (Basis des natürlichen Logarithmus)
<i>Fix(x)</i>	Gibt den ganzzahlig abgerundeten Anteil der Zahl x zurück
<i>Int(x)</i>	Gibt den ganzzahligen Anteil der Zahl x zurück
<i>Log(x)</i>	Liefert den natürlichen Logarithmus (Basis e) einer Zahl x
<i>Randomize[zahl]</i>	Startet den Zufallszahlengenerator
<i>Rnd[(zahl)]</i>	Generiert eine Zufallszahl
<i>Round(zahl)</i>	Rundet auf den nächsten ganzzahligen Wert
<i>Sgn(x)</i>	Liefert 1 oder 0 oder -1 für $x > 0$, $x = 0$, $x < 0$ (Vorzeichenfunktion)
<i>Sin(x)</i>	Berechnet den Sinus eines Winkels x, der im Bogenmaß angegeben ist
<i>Sqr(x)</i>	Berechnet die Quadratwurzel einer Zahl x
<i>Tan(x)</i>	Berechnet den Tangens eines Winkels x, der im Bogenmaß vorliegt

Winkelfunktionen

Die Werte für Winkel sind durchgängig im Bogenmaß (Maßeinheit *Radian*) angegeben. Benutzen Sie z.B. die folgenden Anweisungen, um zwischen *Grad* und *Rad* umzurechnen:

```
Const Pi = 3.141592654
Dim rad As Double
Dim grad As Double

rad = Pi * grad / 180 = 0.0175 * grad
grad = 180 * rad / Pi = 57.296 * rad
```

Weitere trigonometrische Funktionen lassen sich aus den Standardfunktionen ableiten. Die nachfolgende Tabelle zeigt eine Auswahl, die Sie mit Hilfe einer "mathematischen Formelsammlung" beliebig ergänzen können:

Winkelfunktion	Berechnung mit Standardfunktionen
<i>CoTangens</i> (x)	$1/\text{Tan}(x)$
<i>ArcSin</i> (x)	$\text{Atn}(x/\text{Sqr}(1-x*x))$
<i>ArcCos</i> (x)	$\text{Atn}(\text{Sqr}(1-x*x)/x)$
<i>ArcCoTangens</i> (x)	$\text{Atn}(x) + 2 * \text{Atn}(1)$
<i>Sekans</i> (x)	$1/\text{Cos}(x)$
<i>CoSekans</i> (x)	$1/\text{Sin}(x)$
<i>SinusHyperbolicus</i> (x)	$(\text{Exp}(x) - \text{Exp}(-x))/2$
<i>CosinusHyperbolicus</i> (x)	$(\text{Exp}(x) + \text{Exp}(-x)) / 2$
<i>ArcSinHyperbolicus</i> (x)	$\text{Log}(x + \text{Sqr}(x * x + 1))$

BEISPIEL: Der *ArcCos*(0,5) soll in der Maßeinheit Grad ermittelt werden.

```
Const Pi = 3.1416
Dim x As Double
x = 0.5
Debug.Print 180 / Pi * Atn(Sqr(1 - x * x) / x)      ' 60 Grad (59,999...)
```

Zufallszahlen

Wenn Sie die *Rnd*-Funktion ohne Argument einsetzen, wird ein Wert im Bereich 0 ... 0.99999, basierend auf der Systemzeit, zurückgeliefert. Das Argument *zahl* kann auch das Ergebnis eines Ausdrucks sein. Sein Wert legt fest, wie die Zufallszahl generiert wird:

Wert des Arguments zahl	Welche Zufallszahl wird generiert?
< 0	Immer wieder die gleiche Zufallszahl mit <i>zahl</i> als Startwert
> 0	Die nächste Zufallszahl der Folge
= 0	Die zuletzt generierte Zufallszahl
Nicht angegeben	Die nächste Zufallszahl der Folge

Um Zufallszahlen in einem frei definierten Bereich zu erzeugen, können Sie die folgende Formel verwenden:

```
z = Int((obereGrenze - untereGrenze + 1) * Rnd + untereGrenze)
```

Bevor Sie *Rnd* anwenden, sollten Sie durch Aufruf von *Randomize* dafür sorgen, dass sich immer wieder andere Zufallszahlenfolgen ergeben. Die *Randomize*-Anweisung benutzt das optionale Argument *zahl* als Startwert zum Initialisieren des Zufallszahlengenerators. Ohne *zahl* dient der von der *Timer*-Funktion zurückgegebene Wert als neuer Startwert. Verwenden Sie die *Randomize*-Anweisung ohne Argument, können Sie einen zufälligen Startwert (basierend auf der Systemzeit) erzeugen.

BEISPIEL: Durch wiederholtes Anwenden der beiden Anweisungen werden die Zufallszahlen 1 bis 6 generiert (Würfel).

```
Randomize
z = Int(6 * Rnd + 1)
```

Vorzeichen- und Rundungsfunktionen

Bei positiven Zahlen gibt es keinen Unterschied zwischen *Int*- und *Fix*-Funktion, beide liefern nur den ganzzahligen Anteil einer Dezimalzahl zurück, ohne dabei zu runden. Erst bei negativen Zahlen gibt es einen Unterschied. Hier rundet *Int* auf die niedrigere negative Zahl, *Fix* hingegen auf die höhere.

BEISPIEL: Testen Sie im Direktfenster (+

```
? Int(12.6)    ' liefert 12
? Fix(12.6)    ' liefert 12

? Int(-12.6)   ' liefert -13
? Fix(-12.6)   ' liefert -12
```

Im Zusammenhang mit der Signum- und Absolutwert-Funktion gilt die Formel:

```
Sgn(n) * Int(Abs(n)) = Fix(n)
```

HINWEIS: Die Funktionen *Int* und *Fix* geben immer den Datentyp *Double* zurück.

Bei einer Typkonvertierungsfunktion wie *CLng* oder *CInt* hängt die Art der Rundung vom Wert der unmittelbar rechts neben dem Dezimaltrennzeichen stehenden Ziffer ab. Ist diese kleiner als 5, so wird ab-, bei größer 5 aufgerundet. Das scheint normal, interessant ist hingegen der Fall, wenn diese Zahl exakt gleich 5 ist. Dann wird abgerundet, wenn die unmittelbar links neben dem Dezimaltrennzeichen stehende Zahl gerade ist, und aufgerundet, wenn diese Zahl ungerade ist.

BEISPIEL:

Im ersten Fall wird abgerundet (links steht gerade Zahl), im zweiten Fall wird aufgerundet (links steht ungerade Zahl).

```
? CLng(6.5)    ' liefert 6
? CLng(7.5)    ' liefert 8
```

Mit der Rundungsfunktion *Round* können Sie eine Gleit- oder Festkommazahl auf eine bestimmte Stellenanzahl runden.

BEISPIEL: Die Zahl 12,345 wird auf zwei Nachkommastellen gerundet.

```
? Round(12.345,2)    ' liefert 12,34
```

Kritisch ist der Fall, wenn die Rundungsziffer den Wert 5 hat (siehe obiges Beispiel), weil Sie nie genau vorhersagen können, ob auf- oder abgerundet wird.

Quadrieren

VBA stellt zum Quadrieren keine extra Funktion zur Verfügung. Zwar können Sie sich mit dem Potenzoperator behelfen, doch das kostet viel Rechenzeit. Verwenden Sie stattdessen besser die Multiplikation.

BEISPIEL: Beide Anweisungen sind identisch.

```
a = x ^ 2
a = x * x
```

Dringend gebrauchen können Sie aber den Potenzoperator für die Berechnung von beliebig gebrochenen Potenzen bzw. für weitere Wurzeln (außer der quadratischen).

BEISPIEL: Die dritte Wurzel aus 10.

```
? 10 ^ (1 / 3)      ' liefert 2,15443469003188
```

Logarithmus und Exponentialfunktionen

Leider fehlt der dekadische Logarithmus in (fast) jeder Programmiersprache, sodass man sich zu seiner Nachbildung am besten eine eigene Funktion schreiben sollte:

```
Function LogD (x As Double) As Double
    LogD = Log(x) / Log(10)
End Function
```

Allgemein gilt für den Logarithmus zur Basis N:

```
LogN(X) = Log(X) / Log(N)
```

$\text{Log}(x)$ und $\text{Exp}(x)$ sind zueinander Umkehrfunktionen mit der Basis $e = 2.718282\dots$:

```
Exp(Log(x)) = Log(Exp(x))
```

2.8.2 Finanzmathematische Funktionen

VBA stellt zahlreiche mehr oder weniger komplexe Funktionen für diverse Finanzberechnungen zur Verfügung (Tabelle 2.19).

Funktion	Erläuterung
PV (rate, nper, pmt[, fv[, type]])	Barwert einer Annuität bei regelmäßigen konstanten und zukünftig zu leistenden Zahlungsausgängen und konstantem Zinssatz
SYD (cost, salvage, life, per)	Jahresabschreibung eines Vermögenswertes über einen bestimmten Zeitraum
DDB (cost, salvage, life, period[, factor])	Abschreibung eines Vermögenswertes über einen bestimmten Zeitraum mit Hilfe der geometrisch degressiven Abschreibungsmethode oder einer von Ihnen angegebenen Methode
IRR (values()[, guess])	Interner Zinsfuß für eine Folge regelmäßiger Aus- und Einzahlungen

Funktion	Erläuterung
PPmt (rate, per, nper, pv[, fv[, type]])	Kapitalanteil einer Auszahlung für einen bestimmten Zeitraum einer Annuität bei regelmäßigen konstanten Auszahlungen und konstantem Zinssatz
SLN (cost, salvage, life)	Arithmetische Abschreibung eines Vermögenswertes über einen bestimmten Zeitraum
MIRR (values(), finance_rate, reinvest_rate)	Modifizierter interner Zinsfuß bei einer Folge regelmäßiger Aus- und Einzahlungen
NPV (rate, values())	Netto-Barwert einer Investition bei regelmäßigen Aus- und Einzahlungen und einem Diskontsatz
Pmt (rate, nper, pv[, fv[, type]])	Auszahlung für eine Annuität bei regelmäßigen konstanten Zahlungsausgängen und konstantem Zinssatz
Rate (nper, pmt, pv[, fv[, type[, guess]]])	Zinssatz einer Annuität pro Zeitraum
IPmt (rate, per, nper, pv[, fv[, type]])	Zinszahlung für einen bestimmten Zeitraum einer Annuität bei regelmäßigen konstanten Zahlungen und einem konstanten Zinssatz
FV (rate, nper, pmt[, pv[, type]])	Endwert einer Annuität bei regelmäßigen, konstanten Zahlungsausgängen und einem konstanten Zinssatz
NPer (rate, pmt, pv[, fv[, type]])	Anzahl der Zeiträume für eine Annuität bei regelmäßigen, konstanten Zahlungen und einem konstanten Zinssatz

BEISPIEL: Die Nutzungsdauer eines Computers (Neupreis 1.200 Euro) beträgt vier Jahre, danach soll er noch einen Restwert (Erinnerungswert) von 1 Euro haben. Die jährliche Abschreibung ist unter Verwendung der *SLN*-Funktion (arithmetischer bzw. linearer Wertverlust) zu ermitteln.

Rufen Sie das Direktfenster auf und tippen Sie ein:

```
? Format(SLN(1200, 1, 4), "Currency")
```

Das Ergebnis ist 299,75 Euro (Abschreibung pro Jahr).

2.8.3 Datums-/Zeitfunktionen

Zu Berechnungen mit Datum und Zeit hält VBA ein vielfältiges Repertoire bereit.

HINWEIS: Beachten Sie, dass eine Variable des Typs *Date* im Allgemeinen nicht nur ein Datum, sondern ein Datum und eine Uhrzeit speichern kann.

Im Folgenden soll deshalb eine Unterteilung nach dem Kriterium erfolgen, ob die Funktion vorrangig auf Datum, Uhrzeit oder beides angewendet wird.